VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Consensus Ascent – Beating Naive Gradient-Based Optimisation

Luis Slyfield

Supervisors: Marcus Frean and Andrew Lensen

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours.

Abstract

Gradients are the cornerstone of deep learning but are expensive to compute. Conventional gradient descent makes poor use of gradient information due to only considering the gradient of a single point at once, and requires ad hoc approaches to determine the step size. This project develops and explores two alternative approaches, in which we instead combine the information of previously seen points in the optimisation space to determine the next step. The first method combines the information into a distribution and samples from it to determine the next step. It does not perform well but provides insight about the problem. The second method provides a new way of thinking about optimisation problems as a generative model that learns off its history. The algorithm developed from this is able to outperform gradient descent in some situations.

Acknowledgements

Thank you to Marcus for sparking the ideas that made this possible, and Andrew for making sure the ideas turned into an honours project. Thank you again to Marcus and Andrew for their knowledge and guidance throughout the project. Thank you to Demelza whose support was integral to completing it. And thank you to all family and friends, whose combined support and influence have led me here.

Contents

Chapter 1

Introduction

The optimisation of mathematical functions is a common problem with applications throughout computer science and other disciplines [25][14][4]. Optimisation problems can take many forms, but the nature of the task is to find the input to the function that gives its highest (or lowest) value. Optimisation is of particular importance in machine learning, as the core component of many contemporary state-of-the-art machine learning methods is optimising an objective function or functions that represent the goal of the learning task [39].

A natural approach to optimisation is to use gradients (if they are available), as they provide a measure of the direction and amount that the function is changing. The representative example of a gradient-based optimiser is gradient descent. Gradient descent and its derivatives are some of the most popular optimisation methods in machine learning, particularly for learning the weights of neural networks. Gradient descent comes with some drawbacks such as getting stuck at sub-optimal points. Optimisers based on gradient descent such as such as RMSProp [41] and Adam [21] try to address such flaws. Most of them do so in an ad-hoc way that tries to address the symptoms rather than the cause.

The problem this project tries to address is to make a gradient-based optimiser that combines the gradient information from multiple points in the search space to make better use of such information than conventional gradient descent. The aim is also to produce a method that is backed up by a theoretically grounded model rather than developed by an empirical approach. This means the model can be improved by making changes to the model rather than trying to modify observed behaviour.

In particular we propose a new way of thinking about an optimisation problem: that optimisation is akin to an unsupervised learning problem where we have observed data in the form of queries from an objective function and its gradient, and the global optimum is a latent variable that we want to learn about. We frame this as a generative relationship where the global optimum generates the observed data. It is then possible to model our beliefs about the relationship, and then produce an optimisation algorithm that follows from that model.

1.1 Motivation

Methods based on gradient descent are often used for optimisation, particularly when training neural networks. Gradient descent by itself is a relatively naive algorithm that is only guaranteed to converge for convex functions: functions that have gradients that always point away from the minimum. For functions with more complex structure it can easily get stuck in local minima with poor objective function values, or at saddle points that are far from good minima.

The most popular improvements on gradient descent such as Adam [21] are able to overcome aspects of its issues, but tend to involve ad-hoc approaches and require large data sets to be effective. Furthermore, their effectiveness at reaching solutions that generalise well to data not seen in training may even be worse in many cases [45]. This project aims to provide an alternative gradient-based method for optimisation that uses the gradients of multiple points at once and combines their information to produce a better new point than traditional gradient descent would using the points separately.

We propose such a consensus-based approach as being able to make better use of the data available in an optimisation task, which includes all previously seen points and their gradient and objective function values. Many methods waste most of this information, such as gradient descent by only considering the most recently seen point, or adaptive learning rate derivatives of gradient descent that encode it into one or two numbers.

Note that optimising a single function can be a minimisation or maximisation task. For this project is assumed that we are minimising an objective function as this is the more common task in machine learning. A maximisation task can always be re-framed as a minimisation task by minimising the objective function multiplied by -1.

1.2 Goals

The overall aim is to develop a gradient-based method for optimisation that makes effective use of the information given by all previously observed points, which includes their positions, gradients, and objective function values. The desired outcome is to produce a method that can outperform standard gradient descent using such information, and to do so in a way they can be easily built upon to create new methods.

1.2.1 Specific Goals

This is split into the following sub-goals:

- **Goal 1:** Create a system for running experiments to compare optimisers on a variety of models and data sets in numerical optimisation and neural network training contexts.
- **Goal 2:** Develop and test an optimisation method that combines the results of many gradient descent steps into a single distribution to explore the merits of that approach to combining the available information in optimisation.
- **Goal 3:** Using the lessons from achieving Goal 2, develop and test an optimisation method that is based on a theoretical model for the relationship between the global optimum and previously observed points. This should be able to exhibit some advantages over conventional gradient descent, and have the potential to be improved upon by adjusting the underlying beliefs it is based on.

1.3 Contributions

This project provides two novel approaches to optimisation in a machine learning context and explores their performance. The first method uses a sampling based approach to taking multiple gradient descent steps at once. It does not perform well but provides insight that informs the design of the second method. The second is based on a generative model that shows improved performance over standard gradient descent when used as an optimiser for neural networks and has potential to be improved in future work. In addition, the

motivation for the second approach introduces the idea of formulating optimisation as an unsupervised learning problem where the global optimum is a latent variable. Both models are implemented as PyTorch optimisers which enables them to easily be used in machine learning tasks.

1.4 Report Organisation

Chapter 2 provides the background to place this project in context, including a literature review covering the background required to understand the later sections and their place in the artificial intelligence field, and related works that have investigated similar problems.

Chapter 3 outlines the testing system developed to meet the first goal in Section 1.2.1. Chapter 4 describes the optimisation algorithm developed to meet the second goal and presents results of experimental testing. Chapter 5 does the same for the model developed to meet the third goal.

The findings are discussed and put into context in Chapter 6. Chapter 7 summarises the report and provides possibilities for future research building upon this work.

Chapter 2

Background

This section covers the background information required to understand the methods developed in this project. It assumes the reader has an understanding of the computer science and mathematics equivalent to that gained in an undergraduate computer science degree, but not necessarily the specifics of the topics covered.

2.1 Optimisation

In the broadest sense, optimisation is defined as trying to find the best way to make use of resources or take action in a situation to achieve a goal. *Mathematical optimisation* (referred to as just *optimisation* in this work) formalises this idea as the selection of the best element of a set of options given some criterion [4][30]. Optimisation is perhaps the biggest problem in applied mathematics, with applications throughout many scientific fields [25][14][4]. The criterion for optimisation could be any number of objectives, but the most common problem in machine learning is trying to find the value that minimises a single real-valued function with multi-dimensional inputs and single- or multi-dimensional outputs.

2.1.1 Convex versus Non-Convex Optimisation

An important distinction in optimisation is whether or not the problem is *convex* (for minimisation; the analogue for maximisation is *concave*). Formally, a convex optimisation problem involves optimising a convex function, or functions, over inputs that form a convex set. A *convex function* is a function that has a positive semi-definite Hessian over its domain [5]. This is a desirable property for optimisation because it results in the function having only a single minimum (or maximum for a concave function) value with no saddle points. This means the gradient of the function always points away (or towards for a concave function) from that value.

2.1.2 Nonlinear Optimisation

Nonlinear optimisation refers to the more general case of an optimisation problem where the objective function and constraints are not linear or known to be convex [5]. Many of the tasks in machine learning, such as trying to find the appropriate weights in most neural networks, are nonlinear optimisation problems [15]. There is no best way to solve such problems, and many different approaches are available. These can be broadly split into local and global optimisation methods. *Global optimisation* refers to finding the true global solution to the problem, whereas *local optimisation* finds a locally optimal point that may not be the global

optimum. Local optimisation has the advantage of usually being faster and cheaper, at the expense of settling for a potentially inferior solution.

2.1.3 Popular Optimisation Methods

When trying to optimise a differentiable mathematical function, a natural approach is to make use of its gradient. The gradient of a function at a point is a vector made up of the partial derivatives of the function at that point [38]. This provides a direct measure of the direction in which the function is changing. We can then move against this direction (for minimisation, or with it for maximisation) in hope of finding a better value of the function. We may like to also make use of second or higher derivatives to gain more information about the way the function changes. Some functions are not differentiable or have derivatives that are expensive to evaluate, leading to other methods being required.

One way to categorise common optimisation methods is therefore: first-order methods that make use of first-order derivatives, higher-order methods that make use of second or higher derivatives, and derivative-free methods that do not use any gradient information [39].

First-order methods are those that make use of gradient information. The standard example of this in machine learning is *gradient descent* [6], as well as the methods derived from it. Gradient descent involves making iterative steps based on the gradient of a function, and is further detailed due to its importance in this work.

Higher-order methods make use of higher-order derivatives to gain more information about the behaviour of the function being optimised. This comes at the expense of increased computational overhead, even for methods where only second derivatives are used [39] . For any application with more than one dimension this involves manipulating or approximating the multi-dimensional analogue of the second derivative: the Hessian matrix, such as in Newton's method [30]. Some steps can be taken to reduce the computational load of this by approximating instead of calculating the Hessian matrix, such as in Quasi-Newton Methods [30]. Methods such as conjugate gradient [37] and the Hessian free method [24] try to use or approximate second-order gradient information without using the Hessian at all. Another approach is natural gradient method [2] which performs gradient descent in a space of distributions.

Derivative-free methods are especially valuable for difficult problems where gradient information is not available or is expensive to obtain. These methods can be further split in two main types [39]: *heuristic algorithms* that try to approximate solutions using heuristics and empirical rules or methods, and *function fitting* methods that try to fit another function to samples from the objective function and then find the optimum of that. Heuristic methods include genetic algorithms [10], simulated annealing [22], particle swarm optimisation [20], and ant colony algorithms [12]. Function fitting methods include Bayesian optimisation [36] and derivative-free coordinate descent [46].

2.2 Gradient Descent

Variations on gradient descent are among the most widely used optimisation algorithms, especially for updating neural network weights [39][15]. Gradient descent is based on the principle of solving a minimisation problem by taking successive steps in the direction opposite to the gradient of a function [6]. In the context of machine learning where we are usually trying to learn the parameters of a model, this idea is summarised by the following equation:

$$\theta_n = \theta_{n-1} - \eta \nabla_{\theta_{n-1}} L(\theta_{n-1}), \tag{2.1}$$

where θ_n represents the parameters being optimised (after n updates), η the learning rate, and L the objective (loss) function being optimised over. It provides an iterative method of optimising the parameters by updating them in discrete steps with size determined by the hyperparameter η, in the opposite direction of the gradient of the loss function at each iteration. While this definition assumes we are minimising an objective function, the same method can be used to maximise an objective function by moving in the same direction as the gradient.

When used in machine learning with a data set of training instances, there are three common variants of gradient descent [35]:

- *Batch gradient descent* involves finding the gradient of the loss function for the entire training set at each step before updating the parameters. This can lead to redundant computations being performed for large data sets due to recomputing similar gradients for similar data entries.
- *Stochastic gradient descent* updates the parameters each time it computes the gradient of a single instance of the training set. While this allows updating parameters before computing potentially redundant calculations as in batch gradient descent, it can lead to more erratic convergence due to computing potentially very different gradients at each step.
- *Mini-batch gradient descent* is a mix of the two previous methods and updates the parameters after computing the gradients of a fixed number of training instances. In many situations this allows the benefit of stochastic gradient descent while having less variance in the parameter updates.

2.2.1 Drawbacks of Gradient Descent

With a sensible choice of learning rate, basic gradient descent is guaranteed to reach a local minimum for smooth surfaces and a global minimum for convex surfaces [5]. However it can be inefficient or struggle to reach a desirable minimum for more complicated loss surfaces. The main issues with gradient descent are outlined below.

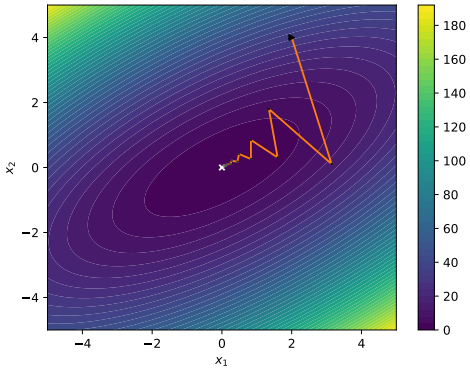
Only Using Local Information and Getting Stuck at Stationary Points

Gradient descent only makes use of the gradient at a single point to inform what the next step is. This means it is not taking full advantage of all information available and the information gained from previous steps is wasted. This exploitation-heavy approach also leads to gradient descent easily getting stuck at stationary points.

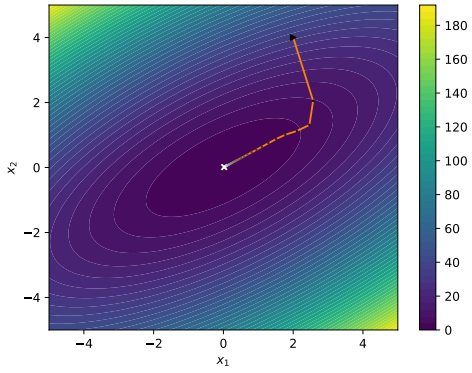
The stationary points of a function are the points at which the gradient of a function is zero [38]. This can include both optima (maxima and minima) and saddle points where the gradient is zero but the point is not an optimum. For a smooth surface, the gradient will be close to zero near this point [5]. By definition, gradient descent takes steps proportional to the gradient at that point, so it will slow down as it approaches a stationary point or stop completely if it reaches it exactly. This is desired behaviour if the stationary point is a very good local (or ideally global) optimum but can present a serious problem if the point is poor local optimum or a saddle point far from a good local optimum.

The Gradient Does Not Always Point to the Optimum

Gradient descent can easily get stuck oscillating in ravine-like features of loss surfaces, and the weight spaces of neural networks often contain such features [40]. By definition, gradi-



(a) Standard gradient descent.



(b) Gradient descent with momentum.

Figure 2.1: Example of gradient descent oscillations and how momentum addresses this. In each case the algorithm is started at the black point and finishes at the white point, with the orange line connecting the points at each step.

ent descent moves in the direction parallel to the gradient, or equivalently perpendicular to the contour lines of a function. This is often not in the direction of an optimum and can lead to undesirable behaviour. For example, gradient descent will tend to bounce between the walls of an elongated ellipse function in two dimensions rather than head directly towards the bottom as illustrated in figure 2.1a.

Setting the Learning Rate

The learning rate is a hyperparameter that must be set by the user. This can present a challenge in finding the right learning rate to allow converging in a desirable time while not fluctuating too much or even diverging. For problems where the function behaves differently over different length scales in each dimension it may not even be appropriate to apply a single learning rate to the gradient as a whole. This led to the development of adaptive learning rate methods such as AdaGrad [13] and Adam [21] that are discussed below.

2.2.2 Improved Gradient Descent Methods

Most of the attempts to improve on standard gradient descent focus on using dynamically adapting the learning rate, or ways to dampen oscillations. A common approach to damping oscillations is to make use of momentum [32]. One of the most popular adaptive learning rate methods is Adam [21], which builds on the benefits of AdaGrad [13] and RMSProp [41].

Momentum

One method to avoid the oscillations observed in Figure 2.1a is to introduce a momentum term [32]. This is inspired by the physical concept of momentum and has a damping effect that smooths the path taken, as shown in figure 2.1b. It helps to address the flaw in gradient descent only using the gradient information at a single point at a time.

AdaGrad

AdaGrad (Adaptive Gradient) [13] maintains separate learning rates for each dimension and adjusts them as optimisation proceeds. These are based on the sum of squared partial

derivatives for each dimension of all previously seen points in the search space. It also means that AdaGrad is not as sensitive to the initial choice of learning rate. A limitation of AdaGrad is that the step size for each parameter tends to get too small over time leading to progress stagnating prematurely.

RMSProp

RMSProp [41] tries to address the flaw of AdaGrad. Instead of directly using an average partial gradient like AdaGrad does, it uses a decaying average that leads to more recently seen points having more influence. This makes it less susceptible to the early halting of AdaGrad.

Adam

Adam [21] combines the ideas of AdaGrad and RMSProp by maintaining adaptive estimates of both first and second order moments of the gradients. It is often used as the default optimiser for neural networks.

2.3 Inspiration Outside Gradient Descent

2.3.1 Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES)

CMA-ES (Covariance Matrix Adaptation - Evolutionary Strategy) [17] is an evolutionary algorithm that evolves a covariance matrix for a multi-variate normal distribution that is intended to approximate a second order model of the function being optimised. While CMA-ES evolves a model by sampling points in the optimisation space, it does not sample or estimate gradients, whereas this project intends to develop a method that takes advantage of gradient information.

2.3.2 Bayesian Optimisation

Bayesian optimisation [26] is another method that may be used for inspiration, as it also involves sampling from points in the optimisation space and is able to reach a solution in relatively few steps. The method uses a surrogate model to represent the possibilities for the function being optimised, with the most popular choice of surrogate model being Gaussian processes. It then optimises an acquisition function that determines the next point to sample. The acquisition function is chosen depending on the required balance between exploration and exploitation, and is usually considerably easier to optimise than the actual objective function. [36]

Despite its benefits, some issues prevent Bayesian optimisation from being more widely used. It is difficult to scale to higher dimensions, chiefly due to covariance matrix inversion computational complexity increasing with dimensionality. There have been attempts to make approximations to get around this [9][33][44] that show promise but make trade-offs. Bayesian optimisation also requires careful choice of parameters and surrogate models.

2.3.3 Particle Swarm Optimisation (PSO)

PSO [20] approaches optimisation by having a population of candidate solutions represented as particles each with their own position and velocity. All particles in the population have their position updated iteratively according the their velocity and previous position, and their velocity according to a weighted combination of the best point the particles has

seen and the best position(s) that any particles has seen. PSO bears some similarity to the goal of this project, as it also seeks to combine information of various points in the search space to perform optimisation. The fact that it does not use any gradient information can be of benefit when such information is difficult or impossible to obtain. However this can also be a drawback when that information is available and not used.

2.4 Bayesian Inference

Bayesian inference provides a powerful method of reasoning under uncertainty, and we make use of it in the method described in Chapter 5. If we let probabilities represent degrees of certainty about events, we can then make use of Bayes' rule to update beliefs [3]. Bayes' rule gives a relationship between two random variables U and X as follows:

$$P(U|X) = \frac{P(X|U)P(U)}{P(X)} \tag{2.2}$$

Bayes' rule holds for any two random variables, but for the purposes of inference we can consider U to be an unknown quantity that we would like to gain information about, and X an observed quantity. Then each term can be thought of as follows:

- $P(X|U)$: The **likelihood** that possible values of U lead to possible values of X.
- $P(U)$: The **prior** distribution of U that represents our beliefs about it in the absence of observations.
- $P(X)$: The prior distribution of X. In the context of inference about U based on X this is reduced to a **normalising** term that does not need to be calculated explicitly.
- $P(U|X)$: The **posterior** distribution that represents our new beliefs about U having observed X

2.5 Probabilistic Graphical Models

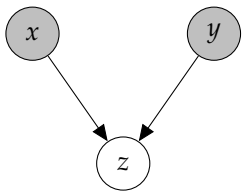


Figure 2.2: Example Probabilistic Graphical Model

Probabilistic Graphical Models (PGMs) provide a way of compactly representing conditional dependence relationships between random variables in a graph [28]. An example PGM is provided in Figure 2.2. Random variables are represented by circles, with shaded circles representing observed variables and unshaded representing latent (unobserved) variables. Arrows between variable show a dependence relationship; an array from x to y means y is dependent on x. The arrows show the only dependence relationships; all other relationships are assumed to be conditional independence.

The joint probability represented by the entire PGM is a product of all relationships implied by the arrows, along with prior probabilities of the parent variables (variables with no incoming arrows). As such, the joint probability of the example PGM in Figure 2.2 can be factorised as: $P(x, y, z) = P(z|x, y)P(x)P(y)$.

2.6 Generative Classifiers

The method to be described in chapter 5 is inspired by an analogy to generative classifiers, so an overview of generative classifiers is presented here.

The most common way of framing machine learning problems is using a *discriminative* approach. This involves trying to learn the relationship described by the conditional probability $P(Y|X)$, where X represents data instances and Y their known or unknown output (labels for classification and clustering or a real vector for regression and dimensionality reduction). *Generative* models try to learn the deeper relationship $P(X, Y) = P(X|Y)P(X)$ [29].

For classification, a discriminative classifier only learns where to put decision boundaries in X space to discriminate between different classes Y, while a generative learns the full joint distribution of X and Y. Discriminative classifiers are usually able to achieve better predictive accuracy as they learn a simpler relationship, whereas generative classifiers learn a richer model that enables them to easily handle missing input features and generate new unseen examples of a class [28].

2.7 Multivariate Normal Distribution

Multivariate normal (also called multivariate Gaussian) distributions [42] are used throughout this project. For the rest of the report, the term *normal distribution* is used to mean *multivariate normal distribution* unless explicitly stated to be univariate (one dimensional). The multivariate normal distribution generalises the normal distribution to any number of dimensions. Its probability density function is given by:

$$P(\boldsymbol{x}) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{2.3}$$

where μ is the mean vector and Σ the covariance matrix. Due to this definition containing its inverse, Σ must be an invertible (also called non-singular or non-degenerate) matrix.

A special case of the multivariate normal distribution is when Σ is a scalar matrix; it is a diagonal matrix with all entries the same. Then the covariance can be represented by a single variance (σ^2) parameter as follows:

$$\Sigma = \sigma^2 \mathbb{1}^d \tag{2.4}$$

for d-dimensions and it is referred to as a *spherical* or *isotropic* Gaussian. The probability density function given in equation 2.3 simplifies to:

$$P(\boldsymbol{x}) = (2\pi\sigma^2)^{-\frac{d}{2}} \exp\left(-\frac{1}{2\sigma^2}(\boldsymbol{x}-\boldsymbol{\mu})^T(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{2.5}$$

for a spherical Gaussian, where d is the number of dimensions.

2.8 Related Work

The general problem of gradient-based optimisation has interest across many scientific fields. This related work section focuses on work within machine learning that is relevant to the idea of combining the gradient information of multiple points in the optimisation space.

One approach to difficult high-dimensional, structured optimisation problems with expensive objective functions can be referred to as Latent Space Optimisation (LSO) [43]. This

involves a generative model that maps from the original search space to a lower-dimensional continuous latent space and performing the optimisation in that space. The authors use a weighting factor based on the rank points based on how good their objective function value us.

The authors of one paper [8] are able to improve the performance of CMA-ES by incorporating gradient information. Their results show that their combined method is able to outperform both CMA-ES and gradient descent on a handwritten digit classification task. However their combined method is only tested for a classifier designed by the authors in previous work [7] and the method was not developed further so it is unclear how successful it would be on a wider variety of optimisation tasks.

Bayesian optimisation does not traditionally make use of gradient information. A technique called derivative-enabled knowledge-gradient (d-KG) [47] incorporates gradient information into an acquisition function for Bayesian optimisation. The paper shows the method has excellent performance on a variety of optimisation tasks.

The stochastic average gradient (SAG) method [34] incorporates memory of previous gradient values into updates which leads to faster convergence on strongly-convex problems. The stochastic variance reduced gradient (SVRG) [19] improves on this method without the need to store the gradients. Saga [11] builds on this again.

A recent paper [16] attempts to unify stochastic gradient descent methods themselves. The authors present a unified theory of gradient descent and methods based on it, and claim that many existing methods are just special cases of the overall formulation.

Chapter 3

Testing Methodology

This section describes the frameworks used to test algorithms developed in this project. These are split into numerical and neural network optimisation tasks. The numerical optimisation tests include functions that can scale to any number of dimensions. The neural network tests give measures of performance in more complicated and practically useful optimisation tasks.

3.1 Numerical Optimisation

3.1.1 Functions

The following optimisation test functions are used. These are selected from a review paper of commonly used benchmark test functions [27]. All of these functions can be an arbitrary number of dimensions, with the dimensionality indicated by d. Two dimensional contour plots for each function are provided in Figure 3.1.

- Ackley

$$f(\boldsymbol{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos 2\pi x_i\right) + 20 + \exp(1) \tag{3.1}$$

  over range: $x_i \in [-32.768, 32.768] \forall i$, with global minimum: $f(0, ..., 0) = 0$

- Drop-Wave

$$f(x_1, x_2) = -\frac{1 + \cos\left(12\sqrt{x_1^2 + x_2^2}\right)}{\frac{1}{2}(x_1^2 + x_2^2) + 2} \tag{3.2}$$

  over range: $x_i \in [-5.12, 5.12] for i = 1, 2$, with global minimum: $f(0, 0) = -1$

- Rastrigin

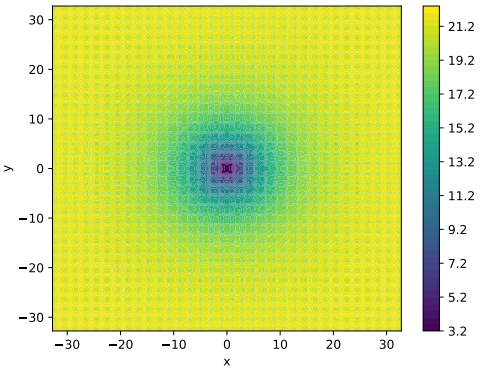
$$f(\boldsymbol{x}) = 10d + \sum_{i=1}^{d}\left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{3.3}$$

  over range: $x_i \in [-5.12, 5.12] \forall i$, with global minimum: $f(0, ..., 0) = 0$

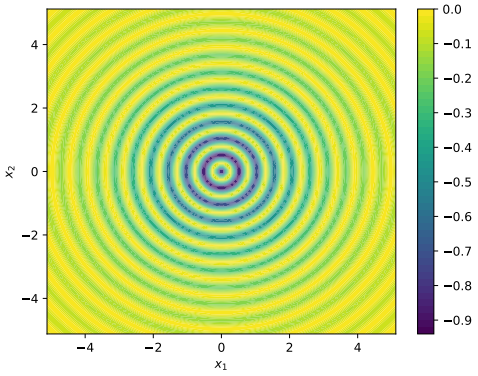
- Sphere

$$f(\boldsymbol{x}) = \sum_{i=1}^{d}(x - a)_i^2 \tag{3.4}$$

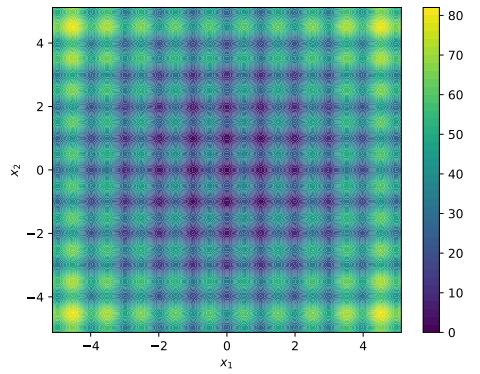
  over range: $x_i \in [-5, 5] \forall i$, with global minimum: $f(a, ..., a) = 0$.



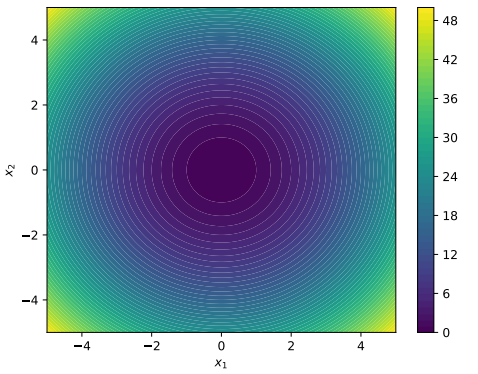
(a) Ackley function



(b) Drop-Wave function



(c) Rastrigin function



(d) Sphere function

Figure 3.1: Contour plots of two dimensional versions of the functions used.

3.1.2 Parameter Selection

An important consideration is what hyperparameters to use for optimisers that have them. To give representative performance for these algorithms, hyperparameters are tried systematically and the best performing selected. This is done by trying a range of values and adjusting the range until the best performing is in the middle of the range. For example, if 1.0, 0.1, 0.01 are chosen and 0.01 perform the best, then 0.1, 0.01, 0.001 should be tried. This is repeated until it is clear learning rates either side of the chosen one perform worse, which gives confidence that we are not missing the best performing value. The selected hyperparameters are specified with each set of results.

3.1.3 Experiment Design

The number of dimensions are chosen to be different for each function to give variety. Initial points are chosen for two dimensional functions to be in an evenly space grid over the range of the function in both dimensions. For higher dimensions, a set number of points are randomly selected from within the range of the function, as the size of an evenly spaced grid increases exponentially. For a specific function and dimensions the same initial points are used for each optimiser tested.

For optimisers with a stochastic component, thirty runs each with different random seeds are performed from each initial point. The objective function values for an optimiser on a test function are averaged over all runs and the standard deviation computed at each iteration.

3.2 Neural Networks

A common use of optimisation methods in machine learning is to learn the parameters of models that represent the task to be solved. The most popular models currently are neural networks, so the methods in this project are tested in that setting. This also provides a more complex high-dimensional setting than the synthetic functions used in the previous section

3.2.1 Test Networks

Fashion-MNIST MLP

- **Task**: Classification
- **Data set**: The Fashion-MNIST data set [48] consists of 70000 grayscale images. Each image consists of 28x28 pixels each represented by an integer grayscale value. There are 10 class labels representing different items of clothing.
- **Network**: A multi-layer perceptron with three layers and ReLU activation functions. The hidden layer contains has 512 input and output dimensions and the input and output layers match the dimensions of this and the data. This combines to a total of 669706 parameters to learn.
- **Loss Function**: Cross-Entropy loss between the predictions of the model on the training set and the true labels.

CIFAR-10 CNN

- **Task**: Classification
- **Data set**: The CIFAR-10 data set [23] consists of 60000 colour images. Each images consists of 32x32 pixels each represented by three integers for each colour channel. There are ten classes.
- **Network**: A simple convolutional neural network with two convolutional layers using max pooling followed by three linear layers. There are a total of 62006 parameters to learn.
- **Loss Function**: Cross-Entropy loss between the predictions of the model on the training set and the true labels.

3.2.2 Experimental Set-Up

Hyperparameters for optimisers that use them are selected in the same way as the numerical optimisation tasks, as described in section 3.1.2.

The comparison metric is the loss value of the loss function. This is chosen because it is the value that directly measures what is being optimised. While it is possible to present metrics such as classification accuracy or error rate for the classification tasks, this is sensitive to other factors such as over-fitting that while important problems in their own right are not directly relevant to the optimisation itself.

The layers in the neural networks are randomly initialised, and for runs where batches are used these are randomly shuffled in training. This means there is a stochastic element to the training process. To allow for this, each algorithm is run with each experiment using 30 different random seeds, which also accounts for stochastic algorithms. Results are presented with the mean and standard deviation at each epoch over the thirty runs. A variety of different batch sizes are used.

Chapter 4

First Approach: Distribution-Based Consensus

The method described in this section aims to solve the goal of combing the gradient information of multiple points by inferring a distribution from the points. Section 4.1 describes the motivation behind the method. Section 4.2 introduces the method, Section 4.3 describes the algorithm the method leads to. This algorithm is implemented and tested with experimental results of the algorithm on numerical optimisation problems presented and discussed in Section 4.4. The chapter is summarised in Section 4.5

4.1 Motivation

Standard gradient descent only follows one path through the optimisation search space, which leaves it vulnerable to being influenced by local behaviour that is not representative of the function as whole, as discussed in Section 2.2.1. This can include getting stuck at stationary points with poor objective function values, and being turned away from optima due to the surface geometry. Using a distribution of points instead of a single point at each step has the potential to overcome these issues as a distribution can cover more of the search space.

Thus we propose a solution to combining gradient information that resembles standard gradient descent in the way it takes steps but treats the current location in search space as a distribution rather than a single point. This bears some similarity to CMA-ES [17] which also builds a distribution from points in the objective space, however CMA-ES does not make use of gradients to make steps.

4.2 Consensus Sampling

To turn this into an algorithm we need to choose a distribution to use, and then determine how to take gradient descent steps with the distribution.

4.2.1 Representation

Assuming no prior knowledge about the structure of the search space, we use a multivariate normal distribution. If we consider the current location in the domain of the objective function being explored as a random variable $\boldsymbol{x}$, then it is represented as:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma) \tag{4.1}$$

where μ is the mean and Σ the covariance matrix of the distribution. These parameters need to be set initially, and then are recalculated after each step. For the purposes of the optimisation task the value of μ can be considered the model's best guess at where the optimum is at the current step.

4.2.2 Taking a Step

To determine the location of the next distribution, we perform a step of gradient descent on points sampled from the current distribution. N points are randomly sampled from the distribution described in equation 4.1 using the current values of $\boldsymbol{\mu}$ and Σ. For each point $\boldsymbol{x}_i, i \in [1, N]$, a step of gradient descent is performed:

$$x_{i_{\text{new}}} = x_i - \eta \nabla \mathcal{L}(x_i) \tag{4.2}$$

where η is the learning rate and $\mathcal{L}$ the loss function. This results in a new set of N points that cover more queries of the objective function than a single step of standard gradient descent. The number of samples to take at each step can be considered a hyperparameter for the method.

4.2.3 Reaching Consensus

The information represented by the locations of the new points is combined to produce a new multivariate normal distribution. The new mean μ of the points is simply the empirical mean $\hat{x}$:

$$\mu = \hat{x} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{4.3}$$

The new covariance matrix can be obtained by computing the sample covariance, which for a multivariate normal is equivalent to the maximum likelihood estimator (MLE) for the covariance, Σ_{MLE} [14]. Each entry of the sample covariance matrix is determined by:

$$\Sigma_{MLE_{i,j}} = \hat{\Sigma}_{i,j} = \frac{\sum_{k=1}^{N} (x_{i_k} - \bar{x}_i)(x_{j_k} - \bar{x}_j)}{N} \tag{4.4}$$

Using a sample covariance matrix directly can cause numerical issues due to the matrix being ill-conditioned. An ill-conditioned matrix is either singular or almost singular, and a singular matrix is not invertible [1]. This means it is not possible to compute the inverse and it cannot be used as the bases for a multivariate normal distribution. Initial testing found that this phenomenon was observed, with many runs failing due to ill-conditioned covariance matrices.

A simple but effective way to ensure the matrix is well-conditioned is using a technique called shrinkage estimation. This uses a prior for the covariance matrix to produce a maximum a posteriori (MAP) estimate. A common way to apply this is to use a prior of the diagonals of Σ_{MLE} multiplied by a prior sample size, which results in reducing the off-diagonals of the covariance matrix according to a set factor λ [28]. This is illustrated by the following equation:

$$\Sigma_{MAP_{i,j}} = \begin{cases} \Sigma_{MLE_{i,j}} & \text{if } i = j \\ (1 - \lambda)\Sigma_{MLE_{i,j}} & \text{otherwise} \end{cases} \tag{4.5}$$

where i and j are the row and column indices of the matrix.

The new mean and covariance matrix can then be used form a multivariate normal distribution to sample from for the next step.

4.3 Description of Algorithm

The above can be summarised in the pseudocode given in Algorithm 1. The algorithm takes as input: an initial point p_0 as the starting point for the search, a learning rate η to use for the gradient descent steps, an initial variance σ^2 which determines the size of the starting distribution, and the number of points n to sample at each step.

First the mean is initialised with the initial point in Line 1 and the covariance matrix is initialised as a diagonal matrix with values of σ^2 down the diagonal in Line 2 to form the initial distribution $\boldsymbol{x} \sim \mathcal{N}(p_0, \sigma^2 \mathbb{1}^d)$. Then at each step: the points are sampled from the distribution (Line 4), a step of gradient descent is performed on each point (Line 5), and new empirical values for the parameters of the distribution are computed (Lines 6-9). This is repeated for a fixed number of iterations.

Algorithm 1: Initial Prototype

Input: p_0 (initial point), η (learning rate), σ^2 (initial variance), N (number of points)

1 $\mu \leftarrow p_0$
2 $\Sigma \leftarrow \sigma^2 \mathbb{1}^d$
3 **for** *a set number of iterations* **do**
4 Sample N points from $\mathcal{N}(\mu, \Sigma)$
5 Perform a step of gradient descent using η on each point (equation 4.2) to produce N new points
6 Update $\mu \leftarrow$ sample mean of new points
7 Compute sample covariance Σ_{MLE} of new points
8 Compute $\Sigma_{MAP} \leftarrow \Sigma_{MLE}$ with shrinkage estimate (equation 4.5)
9 Update $\Sigma \leftarrow \Sigma_{MAP}$
10 **end**

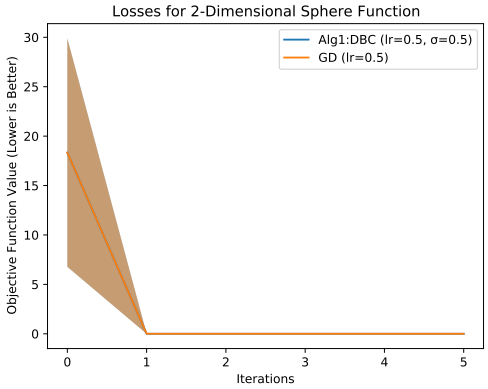
4.4 Results

4.4.1 Implementation

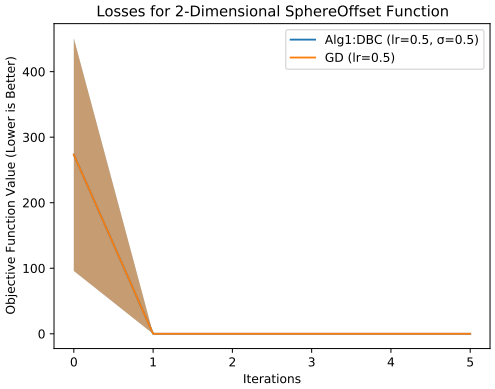
The algorithm described in Algorithm 1 is implemented as a PyTorch optimiser. PyTorch [31] is one of the most popular open source machine learning frameworks, and implementing this method as a PyTorch optimiser means it can be used in any existing optimisation tasks using PyTorch with no additional work. This implementation was used to test the algorithm on the numerical optimisation problems described in Section 3.1. These results are presented in Figure 4.1. The results for the proposed algorithm are all with $N = 10$ points sampled at each step.

4.4.2 Sphere Function

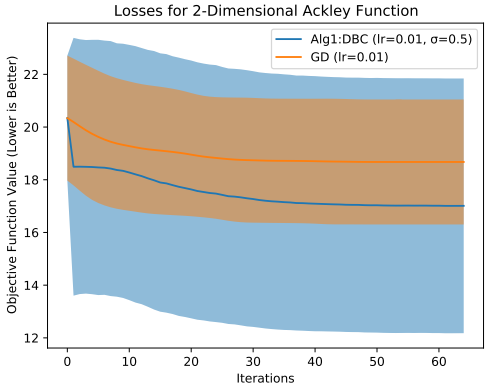
The first results are presented in Figure 4.1a for the sphere function that is described in Equation 3.4 and Figure 3.1d). As this is a quadratic bowl, its gradient is linearly related to the input value. This means that by selecting the right learning rate ($\eta = 0.5$), gradient descent will be able to reach the optimum at the centre in one step from any starting point. As the proposed method is based on taking gradient descent steps, it should also have this property. Figure 4.1a confirms that it does and the method is performing as expected in a simple test case. Figure 4.1b presents the same test but with the function shifted away from the origin to check that the method is not just jumping to the origin.



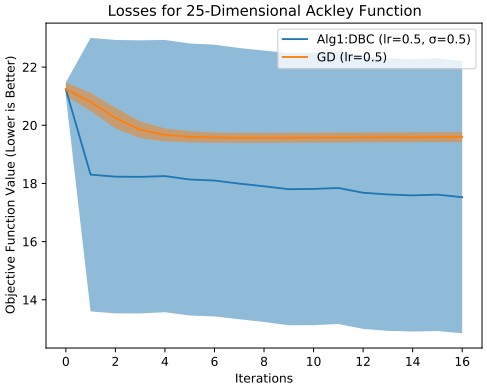
(a) 2D Sphere function



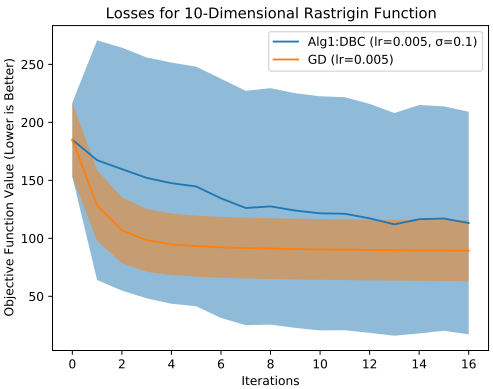
(b) $D2$ Sphere function offset from the origin



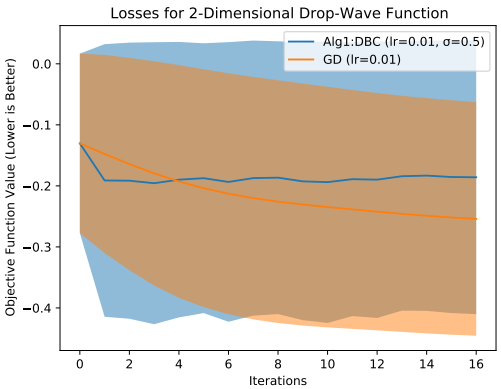
(c) 2D Ackley function



(d) 25D Ackley function



(e) 10D Rastrigin function



(f) 25D Drop-Wave function

Figure 4.1: Comparing the first approach with gradient descent on several functions. The lines represent the mean loss values averaged over 30 runs using the same random seeds for each algorithm, and the shading the standard deviation of the runs.

4.4.3 Functions with Many Local Minima

The Ackley function (Equation 3.1 and Figure 3.1a) contains many local minima that are worse than the global minimum. Given a random initial point in the search space, gradient descent will simply descend to the nearest local minimum due to the topology of the function. This can be seen in the even shape of the distribution of objective value histories in Figure 4.1c and 4.1d. This is even more of a problem for higher dimensions, as the function is symmetrical about the origin so the randomly initialised points are far more likely to be in a narrow band of objective function values, as observed by the lower standard deviation for gradient descent in the 25-dimensional case.

The proposed algorithm is able to break out of the minima due to having a wider search space, and it can be seen on both Figures 4.1c and 4.1d that it reaches lower objective functions values on average compared to gradient descent. This is particularly beneficial for higher dimensions, where it is still able to reach better points from a higher objective function value compared to gradient descent.

Despite performing better on average for both the 2- and 25-dimensional cases, a significant portion of the shaded variability for the proposed algorithm is worse than that for gradient descent. This does not matter for situations like this test, where it is relatively cheap to run the algorithm many times and just choose the best objective function value seen, but in scenarios where the objective function is expensive to evaluate it may only be possible to run the algorithm once or twice. In this limited run scenario, if it is also necessary to start with a random point, then this variability comes with increased risk of only finding worse values than just using gradient descent.

The Rastrigin function (Equation 3.3 and Figure 3.1c) contains many inferior local minima like the Ackley function, but has different structure. Just as with the Ackley function, Figure 4.1e shows the result of gradient descent descending to the nearest local minimum and staying there. The proposed method is again able to break out of these minima, but in this case it often does so to even worse areas of the search space.

The results on the Drop-Wave function described in Equation 3.2 and Figure 3.1b are presented in Figure 4.1f. Rather than having many local point minima like the Ackley and Rastrigin function, it has many rings of lower objective function values radiating out from the true global optimum in the centre. Again we see that gradient descent simply falls to the bottom of the valley it starts in, while the proposed method is able to get over the ridges. However as with the Rastrigin function, this extra explorative ability does not help it reach better objective function values on average.

4.4.4 Lessons

The overall theme is that this first approach is able to explore the objective function space better than gradient descent, but it does not make use of this information effectively to exploit more promising areas. In addition, it is taking several (ten for these results) samples at each step compared to gradient descents single sample. It takes ten times the computational power in addition to the overhead of computing a distribution and sampling from it to gain extra information about the search space, but does not make use of the extra information.

The results on numerical functions are sufficient to show how this method performs and what we can learn from it, so it was not tested on the neural network tests. Taking what was learnt in developing this first approach, the rest of development focused on the second approach in the next chapter.

4.5 Summary

This method shows some promise as it is less prone to getting stuck at stationary points than standard gradient descent. Sampling several points in an area effectively enables it to see past such points. However the flaws of the method render it unlikely to be useful in many practical problems, especially as it comes with considerably increased computational requirements compared to standard gradient descent.

These drawbacks were used to inform the design of the algorithm described in the next section, and provide some possibilities for future work. In particular, it shows that exploration is not useful if a method does not also take into account the objective function values at the points it is exploring.

Chapter 5

Second Approach: Generative Gradient Consensus

The method described in this chapter aims to solve the overall goal of the project using a generative model. Section 5.1 outlines the motivation behind choosing this path for development. Section 5.2 formalises this motivation into a general structured probabilistic model then section 5.3 gives the specific components of the model chosen for implementation. Section 5.4 shows how the model can be used to determine where to search next in optimisation space and 5.5 describes how this is used to create an iterative optimisation algorithm from the model. Section 5.6 presents the results of testing the method. A summary is given in Section 5.7.

5.1 Motivation

In this section we describe how optimisation is essentially an unsupervised learning problem and motivate why solving this problem using a generative model is an appealing approach.

5.1.1 The Data Set for Optimisation

In a black box optimisation problem with gradients available, a point in the search space can be queried for its loss value and gradient. The full information available for an optimisation algorithm to use is in principle all previously seen points along with their loss and gradient values. This can be thought of as a data set to learn from; a set of N tuples:

$$\mathcal{D} = \{(\boldsymbol{x}_i, l_i, \boldsymbol{g}_i), i = 1...N\} \tag{5.1}$$

where $\boldsymbol{x}$ is a vector describing the location of the point in the space being optimised over (such as the parameter space in a neural network), $l = \mathcal{L}(\boldsymbol{x})$ is the value of the loss (or objective) function $\mathcal{L}$ at that point, and $\boldsymbol{g} = \nabla_x \mathcal{L}(\boldsymbol{x})$ is the gradient of the objective function at that point.

This can also be written as:

$$\mathcal{D} = \{X, \boldsymbol{l}, G\} \tag{5.2}$$

where the ith row of the matrix X is $\boldsymbol{x}_i$, the ith entry of the vector $\boldsymbol{l}$ is l_i, and the ith row of the matrix G is $\boldsymbol{g}_i$. Given this data set, we consider what can be learnt from it.

Many popular optimisers do not fully exploit this data. Standard gradient descent makes updates based on the most recently seen point only, without any regard for points observed

in earlier steps beyond the fact those points led to the current point. Methods that extend gradient descent with momentum effectively encode the information of previously seen points into the momentum term [32]. Adaptive learning rate methods such as AdaGrad and Adam instead encode the information into parameters with past gradient information [13][21]. We propose that a better approach is to fully consider the data available and how to learn from it, which we consider in the remainder of this section.

5.1.2 Optimisation is Unsupervised Learning

As established in Section 2.1, the goal of optimisation is to find the global optimum of a function by querying the function. For black box optimisation, we don't know the function itself but can try to learn about properties of the function that inform the values we are seeing. These properties represent unknown qualities we want to learn more about, which is the same task as in unsupervised learning; we have observed data and want to learn underlying properties of it that are not directly observed. We can therefore frame optimisation as an unsupervised learning problem, and to solve this problem we need to determine the properties we want to learn.

There are many ways to represent useful features of the function. The most complete relationship is the function itself, and there are many intermediate properties such as smoothness and periodicity, but ultimately we are only concerned about the global optimum for optimisation. As a result, we develop a method here that simply considers the relationship between the observed data from Section 5.1.1 and the global optimum, with any other relevant properties of the function to be learnt as part of this relationship.

For a loss function $\mathcal{L}(\boldsymbol{x})$, let $\hat{\boldsymbol{x}} \in \mathbb{R}^d$ be the point in the d-dimensional space being optimised over that produces the global optimum value when used as input. The unsupervised learning task can therefore be thought of as trying to infer information about a latent variable $\hat{\boldsymbol{x}}$ given an observed data set $\mathcal{D}$ as in Equation 5.2. As shown in Figure 5.1, there are two ways to frame this relationship, which we discuss in the next section.

5.1.3 Framing Optimisation as a Generative Problem

(a) Discriminative (b) Generative

Figure 5.1: Optimisation as a probabilistic graphical model. $\hat{\boldsymbol{x}}$ represents the global optimum value of the objective function, and $\mathcal{D}$ the data set of observed points described in equations 5.1 and 5.2.

We propose that the task of optimisation can be summarised by learning trying to learn the relationship between the global optimum $\hat{\boldsymbol{x}}$ and all observed points $\mathcal{D}$. There are two ways to consider the direction of dependency in this relationship, as shown in the probabilistic graphical models in Figure 5.1. The circles represent random variables, with shaded variables representing observations. An arrow represents a dependency relationship; the variable at the tip of the arrow is dependent on the variable at the base.

Consider an analogy where standard gradient descent is like a discriminative classifier that focuses on the relationship that leads from the data to the goal of the task (from values

of the objective function to the optimum point for gradient descent, and from instances of a data set to a class label for classification), as in Figure 5.1a. By contrast, a generative classifier learns the relationship where the instances in a data set are considered to be generated by the class labels; rather than just learning to tell the difference between the classes, it learns how to actually produce an instance of the class. To apply this in an optimisation context, we try to learn a relationship where the objective function values are generated by the optimum point as illustrated in Figure 5.1b.

The method described in this chapter was developed by exploring this approach to the problem. We attempt to discover if it is possible to gain improvements over simply considering the data to optimum relationship directly as in standard gradient descent.

5.2 Model Description

The preceding section establishes our motivation for framing optimisation as a generative problem with the global optimum as a latent variable. In this section we formalise this idea into a model.

5.2.1 Establishing Dependencies

We propose a generative model where $\hat{\boldsymbol{x}}$ is a latent variable that generates the gradients $\boldsymbol{g}$ and loss values l we observe in the data at a point x. We can then try to learn this relationship and use it to determine where the optimum point is likely to be. The model makes the following dependence assumptions:

- $\boldsymbol{g}$ and l are dependent on $\hat{\boldsymbol{x}}$: This represents the optimum generating the gradient and loss information which is the assumption the model is based on.
- l is dependent on $\boldsymbol{x}$: In combination with the dependency on $\hat{\boldsymbol{x}}$, the loss value also depending on the point it is at in the search space $\boldsymbol{x}$ gives the information about position relative to $\hat{\boldsymbol{x}}$.
- $\boldsymbol{g}$ is dependent on $\boldsymbol{x}$ and l: The gradient value also depends on both the position it is at and the loss at that point, as they give information about position relative to $\hat{\boldsymbol{x}}$ and the size of the slope at that point.
- individual data points $(\boldsymbol{x}_i, l_i, \boldsymbol{g}_i)$ are independent of each other given $\hat{\boldsymbol{x}}$: This is a key assumption of the model that is inspired by a similar assumption in naive Bayes classifiers and is discussed in more depth below.

All other combinations of variables are assumed to be independent. This is illustrated in the probabilistic graphical model in Figure 5.2. In the figure circles represent random variables and the shaded variables are observed. Arrows mean the variable at the tip is conditionally dependent on the variable at the base; the absence of an arrow between two variables represents conditional independence. The plate represents the variables inside it being repeated N times over, with each ith repetition being the ith entry in the data set from equation 5.1. The plate also represents an assumption of independence between each repetition.

This assumption that each data point $(\boldsymbol{x}_i, l_i, \boldsymbol{g}_i)$ is independent of the others given $\hat{\boldsymbol{x}}$ is important to scrutinise further. In particular, this means that loss values given $\boldsymbol{x}_i$ and $\hat{\boldsymbol{x}}$ are independent given any other point, and gradients are independent given the same. This assumption is clearly not true in practice; the relationship between different $(\boldsymbol{x}, l)$ points in the space is the objective function itself. We would expect gradients at different points to

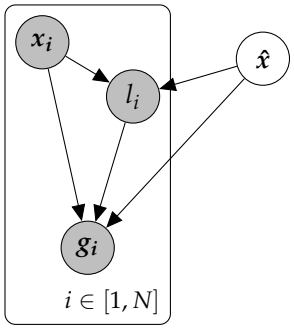


Figure 5.2: Generative model of optimisation $P(\mathcal{D}|\hat{\boldsymbol{x}})$. $\hat{\boldsymbol{x}}$ represents the global optimum, which generates the loss value l_i and gradient g_i at the point in space we are observing x_i for each of the N points. All plates together represent $\mathcal{D}$

be dependent on each other for any smooth function; points closer together should have similar gradient values.

This is similar to the assumption made in a naive Bayes classifier. It greatly simplifies calculations as it enables computing the likelihood by summing over data entries rather than considering an exponential number of complicated dependencies between every pair of points. Naive Bayes classifiers are often able to achieve excellent performance despite the incorrect assumption they are based on [49], which gives support to making that assumption here.

5.2.2 Inverting the Model

For the purposes of optimisation we want to establish our beliefs about the location of the optimum point. Using the model for $P(\mathcal{D}|\hat{\boldsymbol{x}})$ such beliefs are obtained by computing the posterior distribution over the optimum $P(\hat{\boldsymbol{x}}|\mathcal{D})$. This is done using Bayes' Rule (equation 2.2):

$$P(\hat{\boldsymbol{x}}|\mathcal{D}) = \frac{P(\mathcal{D}|\hat{\boldsymbol{x}})P(\hat{\boldsymbol{x}})}{P(\mathcal{D})} \tag{5.3}$$

where $P(\hat{\boldsymbol{x}})$ represents prior beliefs about $\hat{\boldsymbol{x}}$, and $P(\mathcal{D})$ about the data.

Following the dependence relationships given by probabilistic graphical model in Figure 5.2, the likelihood $P(\mathcal{D}|\hat{\boldsymbol{x}})$ is factorised as:

$$P(\mathcal{D}|\hat{\boldsymbol{x}}) = P(G, \boldsymbol{l}, X, |\hat{\boldsymbol{x}}) = P(X|\hat{\boldsymbol{x}})P(L|X, \hat{\boldsymbol{x}})P(G|X, \boldsymbol{l}, \hat{\boldsymbol{x}}) \tag{5.4}$$

Substituting equation 5.4 in equation 5.3:

$$P(\hat{\boldsymbol{x}}|\mathcal{D}) = \frac{P(X|\hat{\boldsymbol{x}})P(L|X, \hat{\boldsymbol{x}})P(G|X, \boldsymbol{l}, \hat{\boldsymbol{x}})P(\hat{\boldsymbol{x}})}{P(\mathcal{D})} \tag{5.5}$$

Note that $P(X|\hat{\boldsymbol{x}}) = P(X)$ due to the lack of direct dependence in our model in Figure 5.2. Substituting this and taking the logs gives:

$$\log P(\hat{\boldsymbol{x}}|\mathcal{D}) = \log P(X) + \log P(L|X, \hat{\boldsymbol{x}}) + \log P(G|X, \boldsymbol{l}, \hat{\boldsymbol{x}}) + \log P(\hat{\boldsymbol{x}}) - \log P(\mathcal{D}) \tag{5.6}$$

Note that $\log P(X)$ $\log P(\mathcal{D})$ are constant with respect to $\hat{\boldsymbol{x}}$ so they don't need to be calculated. They will disappear when differentiated (or if the explicit value is needed, it can be found by normalising). As a result the relationship we are interested in is:

$$\log P(\hat{\boldsymbol{x}}|\mathcal{D}) = \log P(L|X, \hat{\boldsymbol{x}}) + \log P(G|X, \hat{\boldsymbol{x}}) + \log P(\hat{\boldsymbol{x}}) + \text{constant} \tag{5.7}$$

where the constant term is $\log P(X) + \log P(\mathcal{D})$.

The naive Bayes inspired assumption that data points are independent of each other we made in Section 5.2.1 enables us to simplify this further. The joint probability of independent variables is just the product of their individual probabilities [3], so we can replace the probabilities of the losses and gradients over the whole data set with a product of probabilities of each individual data point. The product becomes a sum for the log probabilities, and we get

$$\log P(\hat{\boldsymbol{x}}|\mathcal{D}) = \sum_{(\boldsymbol{x},l,\boldsymbol{g})\in\mathcal{D}} \log P(l|\boldsymbol{x},\hat{\boldsymbol{x}}) + \sum_{(\boldsymbol{x},\boldsymbol{g},l)\in\mathcal{D}} \log P(\boldsymbol{g}|\boldsymbol{x},\hat{\boldsymbol{x}}) + \log P(\hat{\boldsymbol{x}}) + \text{constant} \tag{5.8}$$

as our posterior for the optimum. We now need to be specific about the three individual terms in this equation.

5.3 Model Components

The general model proposed in Figure 5.2 and the result posterior for the optimum (equation 5.8 leave three things to be determined for a specific implementation: a prior on the global optimum $P(\hat{\boldsymbol{x}})$, a likelihood of the gradients $P(\boldsymbol{g}|\boldsymbol{x},l,\hat{\boldsymbol{x}})$, and a likelihood of the loss $P(l|\boldsymbol{x},\hat{\boldsymbol{x}})$. For this first version of the algorithm, these were chosen with the aim of making the computation of the posterior available analytically and cheaply. This is motivated wanting to test the most simple and tractable version of the model as a starting point.

5.3.1 Prior on Optimum

In absence of any prior knowledge about the optimisation problem to be solved, a multivariate normal distribution centred on the origin is used:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{0},\tau^2\mathbb{1}^d) \tag{5.9}$$

with a large value for τ to reflect our initial state of high uncertainty about the location of the optimum point.

Following equation 2.5, this gives a probability distribution as follows:

$$P(\hat{\boldsymbol{x}}) = (2\pi\tau^2)^{-\frac{d}{2}} \exp\left(-\frac{1}{2\tau^2}\hat{\boldsymbol{x}}^T\hat{\boldsymbol{x}}\right) \tag{5.10}$$

Taking the logs:

$$\log P(\hat{\boldsymbol{x}}) = -\frac{1}{2\tau^2}\hat{\boldsymbol{x}}^T\hat{\boldsymbol{x}} - \frac{d}{2}\log(2\pi\tau^2) \tag{5.11}$$

If there is some prior knowledge about constraints or a likely area in the search space that contains the optimum this could be updated accordingly. For example the mean and covariance could be set to represent an area of the search space likely to contain the optimum.

5.3.2 Likelihood of the Gradients

In keeping with our goal of keeping the model as tractable as possible, we again use a normal distribution for the likelihood of gradients:

$$\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{g}|\boldsymbol{\mu},\boldsymbol{\Sigma}) \tag{5.12}$$

This makes the choice of the mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ for the gradient likelihood distribution a key part of designing the model.

Note that the direction of a function's gradient vector describes the direction of steepest ascent of the function. For a minimisation problem, we are interested in the direction of steepest descent. Therefore we are framing a likelihood for the negative gradient $-\boldsymbol{g}$ in this section.

Mean

The vector used for $\boldsymbol{\mu}$ was designed with the motivation of having it on a straight line between $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$, with some measure of prioritising closeness to $\hat{\boldsymbol{x}}$. A straight line between $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ is based on the assumption that the negative gradient at a point $\boldsymbol{x}$ is in the direction of $\hat{\boldsymbol{x}}$. There are two issues with this assumption.

This first issue is that even for a convex problem the negative gradient doesn't always point toward the global minimum, and we hope to solve difficult non-convex problems with many local minima and therefore structure that could cause the gradient to point in any arbitrary direction when not close to the global minimum. It is intended that combining the lines between many $\boldsymbol{x}$ points and $\hat{\boldsymbol{x}}$ will overcome this issue, or at least provide enough extra information to outperform standard gradient descent.

The second issue is that the search space that $\boldsymbol{x}$ points lie in is different to the space that the gradients $\boldsymbol{g}$ lie in. The search space is all the possible loss function values across all possible $\boldsymbol{x}$ values, and the gradient space is the gradients of the loss function with respect to $\boldsymbol{x}$ values. While they have the same dimensionality, they have different units of measurement, so directly using them in arithmetic is not correct. However this is a flaw present in all gradient descent based methods and not the flaw we are trying to address with this project, so we just note it here. This is discussed further in Section 6.1.2.

To measure closeness to $\hat{\boldsymbol{x}}$ we incorporate the loss information l. It is convenient to keep the mean a linear function with respect to $\hat{\boldsymbol{x}}$ so that the mode of the posterior distribution can be easily calculated analytically and meet the target of keeping the method computationally light. A simple way to do this is to first use l to calculate a ranking R for each $(\boldsymbol{x}, \boldsymbol{g}, l) \in \mathcal{D}$, where $R = 1$ is the data point with the best l, $R = 2$ the second best, and so on. Then is then used to formulate:

$$\boldsymbol{\mu} = \frac{1}{R}(\hat{\boldsymbol{x}} - \boldsymbol{x}) \tag{5.13}$$

In order to improve the readability of equations derived from this we introduce the term

$$\alpha = \frac{1}{R} \tag{5.14}$$

to represent the *relevance* of a point. This also has the benefit of being more intuitive; higher α means higher relevance. Our mean is thus given by:

$$\boldsymbol{\mu} = \alpha(\hat{\boldsymbol{x}} - \boldsymbol{x}) \tag{5.15}$$

Covariance

The simplest option for $\boldsymbol{\Sigma}$ is a scalar matrix that results in the likelihood being an spherical Gaussian:

$$\boldsymbol{\Sigma} = \sigma^2 \mathbb{1}^d \tag{5.16}$$

with a single variance value σ^2. This avoids the matrix inversion in the probability distribution function for the normal distribution (equation 2.3 becomes equation 2.5). This is

desirable because matrix inversion can become computationally intensive in higher dimensions and lead to numerical computation issues [18].

Distribution

Substituting the mean (5.15) and covariance (5.16) into the general equation for the normal distribution (2.5) gives the likelihood:

$$P(\boldsymbol{g}|\boldsymbol{x}, R, \hat{\boldsymbol{x}}) = (2\pi\sigma^2)^{-\frac{d}{2}} \exp\left(-\frac{1}{2\sigma^2}\left(-\boldsymbol{g} - \alpha(\hat{\boldsymbol{x}} - \boldsymbol{x})\right)^T \left(-\boldsymbol{g} - \alpha(\hat{\boldsymbol{x}} - \boldsymbol{x})\right)\right) \tag{5.17}$$

and log likelihood:

$$\log P(\boldsymbol{g}|\boldsymbol{x}, R, \hat{\boldsymbol{x}}) = -\frac{1}{2\sigma^2}\left(-\boldsymbol{g} - \alpha(\hat{\boldsymbol{x}} - \boldsymbol{x})\right)^T \left(\boldsymbol{g} - \alpha(\hat{\boldsymbol{x}} - \boldsymbol{x})\right) - \frac{d}{2}\log\left((2\pi\sigma^2)\right) \tag{5.18}$$

Note that $\boldsymbol{g}$ is negated as described at the start of the section.

5.3.3 Likelihood of the Relevancy

The choice to use the relevancy α instead of the loss l in the likelihood of the gradients (5.18) means we should consider likelihood of the relevancy $P(\alpha|\boldsymbol{x}, \hat{\boldsymbol{x}})$ as a proxy for likelihood of the loss $P(l|\boldsymbol{x}, \hat{\boldsymbol{x}})$. A simplifying assumption is that the relevancy is uniformly distributed. This is consistent with the Naive Bayes inspired assumption that data points are independent of each other given $\hat{\boldsymbol{x}}$; the rank that the relevancy is based on is a comparison between points, and if we are assuming they are independent given $\hat{\boldsymbol{x}}$ then we can't say anything about such a comparison given $\hat{\boldsymbol{x}}$. That is to say given the global optimum $\hat{\boldsymbol{x}}$ and a particular point $\boldsymbol{x}$, there is an even chance of it having any possible rank. This assumption is discussed further in Section 6.2. This means that $P(R|\boldsymbol{x}, \hat{\boldsymbol{x}})$ is constant with respect to $\hat{\boldsymbol{x}}$.

5.3.4 Posterior Distribution

Now we can substitute equations 5.11 and 5.18 into equation 5.8 (noting that the likelihood of the relevancy being uniformly distributed means that it is constant with respect to $\hat{\boldsymbol{x}}$):

$$\log P(\hat{\boldsymbol{x}}|\mathcal{D}) = -\frac{1}{2\sigma^2}\sum_{(\boldsymbol{x},\alpha,\boldsymbol{g})\in\mathcal{D}}\left(-\boldsymbol{g} - \alpha(\hat{\boldsymbol{x}} - \boldsymbol{x})\right)^T \left(-\boldsymbol{g} - \alpha(\hat{\boldsymbol{x}} - \boldsymbol{x})\right) - \frac{1}{2\tau^2}\hat{\boldsymbol{x}}^T\hat{\boldsymbol{x}} + \text{constant} \tag{5.19}$$

5.4 Calculating the Mode of the Posterior

The posterior given by equation 5.19 represents where the model believes the optimum is likely to be. Accordingly the mode of this distribution is the most likely location for the optimum. Using isotropic Gaussian distributions makes finding an analytic result for the mode of this posterior relatively straightforward because it is possible to just take the gradient with respect to $\hat{\boldsymbol{x}}$, set to zero, and solve.

The gradient of the posterior with respect to $\hat{\boldsymbol{x}}$ is:

$$\nabla_{\hat{\boldsymbol{x}}} \log P(\hat{\boldsymbol{x}}|\mathcal{D}) = \frac{1}{2\sigma^2}\sum_{(\boldsymbol{x},\alpha,\boldsymbol{g})\in\mathcal{D}}\alpha\left(-\boldsymbol{g} - \alpha(\hat{\boldsymbol{x}} - \boldsymbol{x})\right) - \frac{1}{\tau^2}\hat{\boldsymbol{x}} \tag{5.20}$$

Setting $\nabla_{\hat{\boldsymbol{x}}} \log P(\hat{\boldsymbol{x}}|\mathcal{D}) = 0$ and rearranging to solve for $\hat{\boldsymbol{x}}$ results in:

$$\hat{\boldsymbol{x}} = \frac{\sum \left(\alpha^2 \boldsymbol{x} - \alpha \boldsymbol{g}\right)}{\sum \alpha^2 + \frac{\sigma^2}{\tau^2}} \tag{5.21}$$

where $\sum$ refers to $\sum_{(\boldsymbol{x},\alpha,\boldsymbol{g})\in\mathcal{D}}$.

Now that we have specified each part of the model, we can use it to formulate an optimisation algorithm.

5.5 Description of Algorithm

To produce an algorithm we use the mode of the posterior (Equation 5.21) as an informed guess about where to explore next. Note that there is a potential flaw in using this; going to the expected location of the optimum at every step is potentially prone to getting stuck prematurely. This is discussed further in section 6.1.1. We justify the assumption as it is the same one gradient descent makes, and the goal here is to beat gradient descent.

The overall idea of the algorithm is to, at each step, take the data set of all previously observed points given in equation 5.1, then compute the new mode of the posterior as given by equation 5.21 and add that as a new point for the next step. For the first iteration, the initial point may be either a fixed point or randomly initialised. This is illustrated in Algorithm 2.

In the main loop of the algorithm the new posterior is computed in line 3, the gradient and loss evaluated and stored for next step in lines 4-6, and the relevancies are computed by ranking the data in lines 7-8.

Algorithm 2: GGC: Generative Gradient Consensus

Input: $(\boldsymbol{x}_0, \boldsymbol{g}_0, l)$: initial point, $f(\boldsymbol{x})$: objective function

1 History $\leftarrow$ new list containing initial point
2 **for** *number of iterations* **do**
3 $\quad \boldsymbol{x} \leftarrow$ result of equation 5.21 over History
4 $\quad \boldsymbol{g} \leftarrow \nabla \boldsymbol{x}$
5 $\quad l \leftarrow f(\boldsymbol{x})$
6 $\quad$ Append $(\boldsymbol{x}, \boldsymbol{g}, l)$ to History
7 $\quad$ Sort History by increasing l-values to produce ranking R
8 $\quad$ Assign $\alpha = \frac{1}{R}$ values to all points
9 **end**

5.6 Results

To compare the proposed algorithm, we generate results by following the testing methodology outlined in Section 3.

5.6.1 Implementation

The algorithm described in Algorithm 2 is implemented as a PyTorch [31] optimiser. As stated in the results for the first method, PyTorch is a widely used machine learning framework and implementing this method as a PyTorch optimiser means it can be used in any existing optimisation tasks using PyTorch with no additional work.

5.6.2 Comparison Optimisers

Standard gradient descent is used as a comparison optimiser for both the numerical and neural network sections, as the goal for this section is to beat gradient descent. Adam is also used in the neural network section as a representative canonical method for neural network optimisation. As described in Chapter 3, the learning rates for gradient descent and Adam are chosen systematically to give representative performance.

5.6.3 Numerical

Here we present the results of the proposed algorithm GGC against standard gradient descent on numerical optimisation tasks. To produce these results, a very high value of τ was used for GGC effectively removing the prior on the origin. This is because most of the functions have their global optimum at the origin and it is an unfair comparison to apply a prior on the origin when we are assuming a black box optimisation task.

Modelling the objective function a using multivariate normal distribution is akin to assuming its structure is a quadratic bowl. The sphere function has this shape so provides verification that equation 5.19 does represent such a model. Figures 5.3a and 5.3c show that GGC only requires three evaluations of the sphere function to find the optimum for any number of dimensions. To illustrate that the method is not simply going to the origin, results are also presented in Figure 5.3b with the sphere function translated to have its minimum away from the origin.

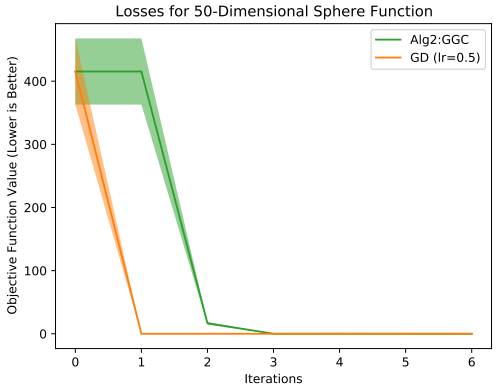
Note that despite taking one more iteration to get close to the optimum (two more for higher dimensions) than gradient descent, the behaviour of GGC is arguably more interesting. Making gradient descent reach the optimum in one step required the selection of the correct learning rate, so it is the result of hyperparameter tuning (or externally computing the gradient and selecting the learning rate accordingly). By contrast the proposed model has no hyperparameters for step size. It had the shape built into the model, and requires two evaluations to work out where each side of the bowl is before jumping to the bottom in the next step. Because it is making steps based on updating its posterior beliefs about a generative model, the decision at the third step is based on its beliefs about the entire surface. It has therefore learnt the entire surface in two steps. Note that it does not quite reach the minimum in the third step; this is due to the weighting introduced by the relevancy term.

Next we see how GGC performs on numerical functions that do not match its model in Figures 5.3d-5.3f. Only on the Ackley function (Figure 5.3d) is GGC able to beat gradient descent. Figure 5.3e shows very high initial fluctuations for the Rastrigin function, before remaining at very high values. The raw data confirms that the objective values are very high across many runs and this graph is not just the result of a small number of runs raising the mean values drastically. Finally Figure 5.3f shows that GGC does not beat gradient descent on the Drop-Wave function.

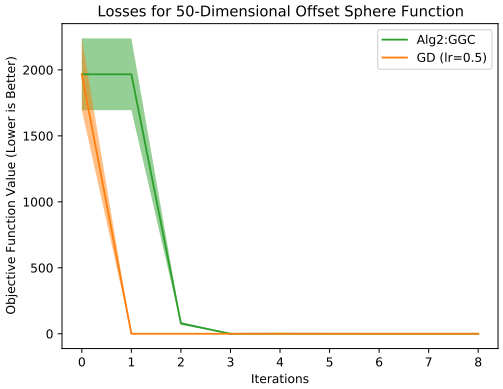
In summary, the proposed method GGC performs very well on the sphere function which matches the assumptions of the model it is based upon. It does not perform well on synthetic benchmark functions with many local minima.

5.6.4 Neural Network

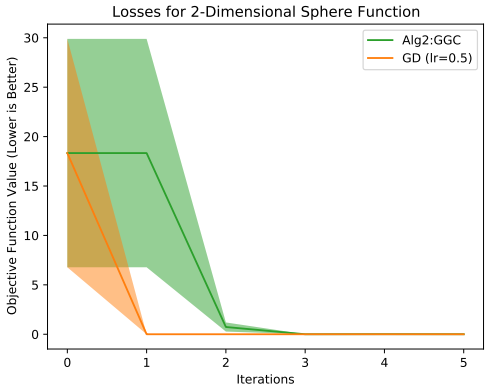
For the first run of experiments, the variance of the prior τ^2 for GGC was chosen to be sufficiently large compared to the variance for the likelihood of the gradients σ^2 such that $\frac{\sigma^2}{\tau^2} \approx 0$ and that term disappears from equation 5.21. This represents no prior beliefs about the location of $\hat{\mathbf{x}}$.



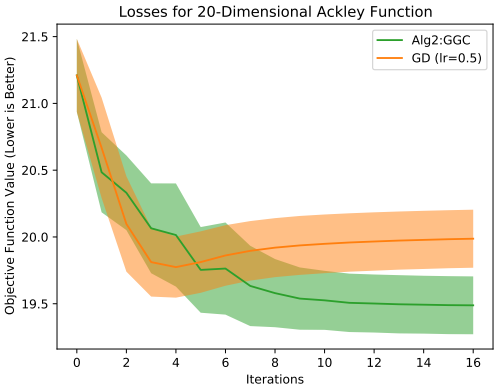
(a) 50D sphere centred on the origin.



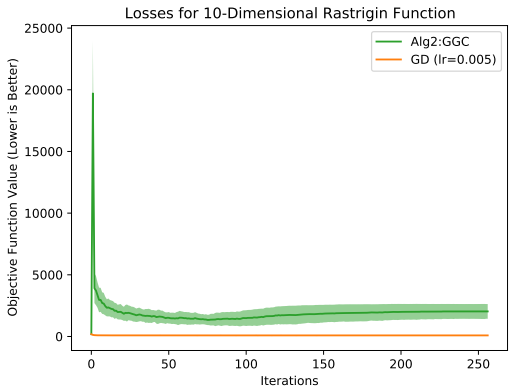
(b) 50D sphere offset by 10 in all dimensions.



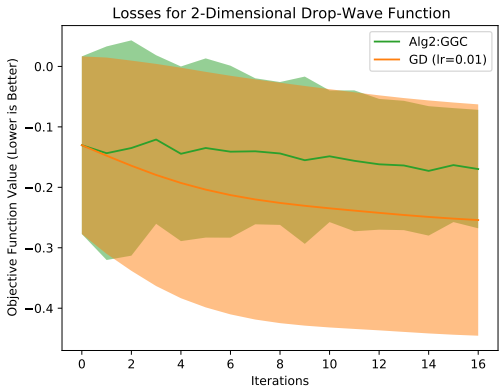
(c) 2D Sphere function.



(d) 20D Ackley function.

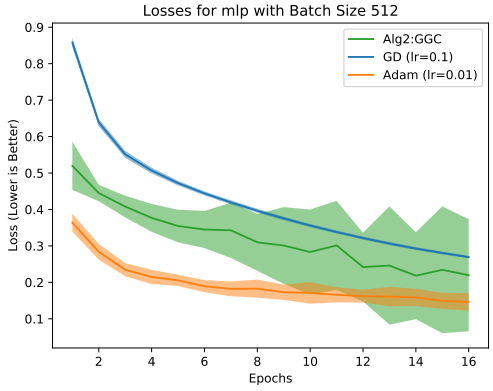


(e) 10D Rastrigin Function

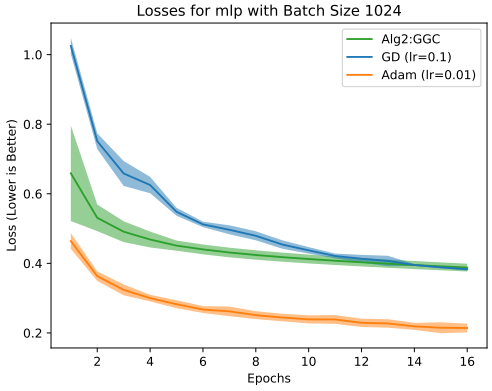


(f) Drop-Wave function.

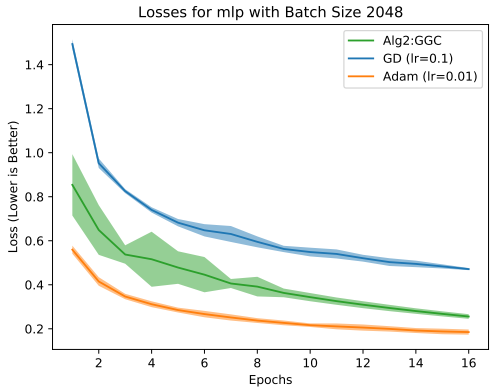
Figure 5.3: Comparisons on numerical optimisation tasks. The lines represent the mean loss values averaged over 30 runs using the same random seeds for each algorithm, and the shading the standard deviation of the runs.



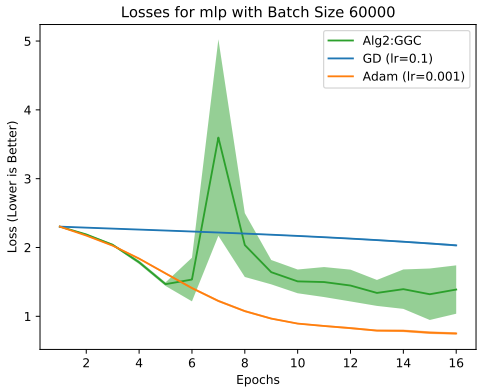
(a) Batch size = 512



(b) Batch size = 1024



(c) Batch size = 2048



(d) Batch size = 60000 (full training set)

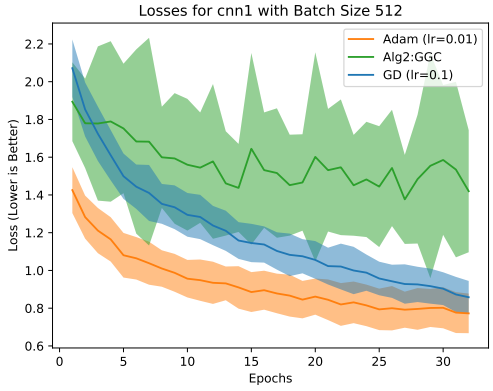
Figure 5.4: Results for the MLP network on Fashion-MNIST for various batch sizes with the proposed method having no prior (very large τ). The lines represent the mean loss values averaged over 30 runs using the same random seeds for each algorithm, and the shading the standard deviation of the runs. Note the optimisers all start with the same weights for each seed, but the plots start with the loss values after the first epoch.

No Prior, MLP

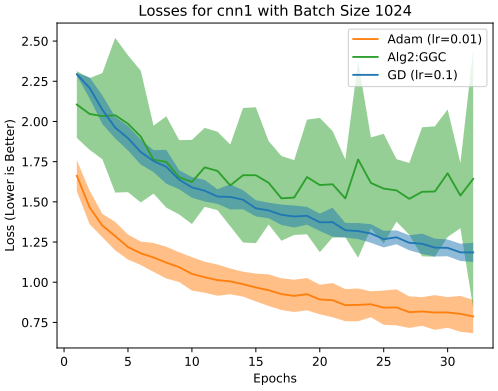
Figure 5.4 shows comparisons between the proposed method, standard gradient descent, and Adam, for a variety of batch sizes on the Fashion-MNIST MLP experiment described in Section 3.2.1. Initial inspection shows the relative performance of the proposed method appears to be better than gradient descent and worse than Adam, although there are some interesting features of the results to note.

For all tested batch sizes, the GGC has higher variance between the 30 random runs than the established methods. Given that it is a deterministic algorithm, this suggests it is more sensitive to initial parameters than the other methods. This is usually not a desirable property as it may require multiple runs to achieve representative results when used in practice.

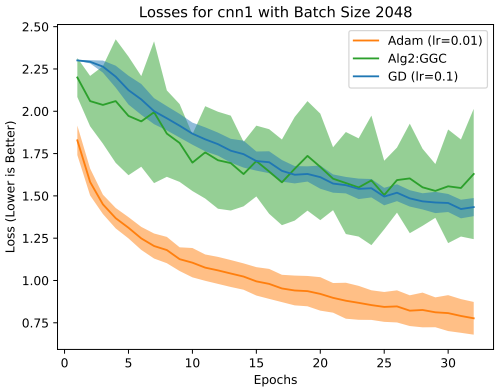
The performance for GGC when using full batch size (Figure 5.4d) starts out on average superior to both Adam and gradient descent for several epochs, before jumping erratically. This graph is of particular interest for both of these reasons. Beating Adam at any point surpasses the expectations set out in this project, even if it is only in one particular situation, and shows promise for developing the method further. The sudden jump after 6 epochs is



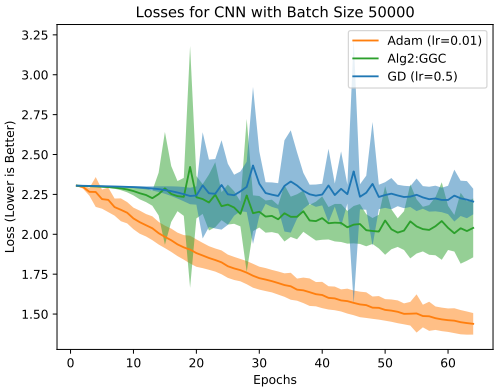
(a) Batch size = 512



(b) Batch size = 1024



(c) Batch size = 2048



(d) Batch size = 50000 (full training set)

Figure 5.5: Results for the CNN network on CIFAR-10 for various batch sizes with the proposed method having no prior (very large τ). The same notes as Figure 5.4 apply.

also of note; understanding why this occurs could help make improvements to the method.

The behaviour in Figure 5.4b is also of note. It is not clear what it is about the 1024 batch size that leads to GGC plateauing earlier than for 512 and 2048.

No Prior, CNN

Figure 5.5 presents the same experiments on the CIFAR-10 CNN experiment described in Section 3.2.1. For smaller the batch sizes (512 and 1024 in Figure 5.5a and 5.5b) GGC performs worse on average than both gradient descent and Adam. For a batch size of 2048 it is able to match the other methods. Only for the full training set batch size is it able to beat gradient descent.

Introducing the Prior

Figure 5.7 shows that using a prior is able to reduce the erratic behaviour observed for the proposed algorithm in Figure 5.4d. However this comes at the expense of decreasing initial performance, and the GGC no longer beats Adam in the earlier epochs.

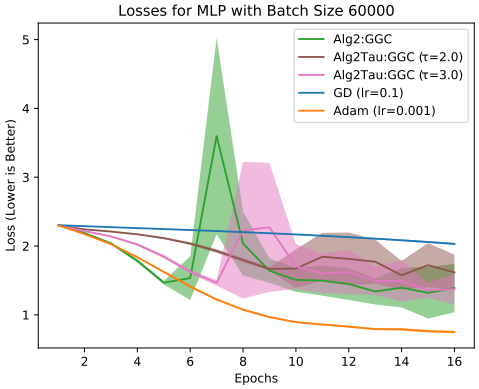


Figure 5.6: MLP, Batch size $= 60000$

Figure 5.7: Results with a small Gaussian prior. The values of τ are specified for plots of GGC that used it. The same notes as Figure 5.4 apply.

5.7 Summary

In this section we motivated and introduced an optimisation method based on a generative probabilistic model where the global optimum is a latent variable that generates the observed points, their gradients, and their loss values. We made simplifying assumptions about the model to produce an algorithm and tested it. The results showed poor performance on difficult numerical optimisation tasks, promising performance on one neural network, and mixed performance on another neural network.

Chapter 6

Discussion

6.1 Optimiser Comparison

The motivation for the second approach (GGC) described in Chapter 5 introduces the idea of optimisation as an unsupervised learning problem with a data set consisting of all previously observed points, their gradients, and their loss function values. In this section we compare the optimisers from this point of view.

The first approach taken in this project and described in Chapter 4 is able to show some positive attributes but ultimately is unlikely to be practically useful. The main reason for this is that it puts too much emphasis on exploration. While it is able to reach potentially better areas of the search space, it does nothing to exploit an area once it is there. In essence, it is ignoring the loss function values available in the data to learn from. If this algorithm were to be developed further, this would be the most important aspect to work on. This could be done by developing a way to incorporate the objective function values.

Another drawback of the first approach is the relatively weak motivation. While it fits with the overall goal of the project of reaching a consensus from multiple points, it does so in a way that does not have a clear model behind it.

Also note that the testing of the first approach in Section 4.4 is an unfair comparison. We present a single step of gradient descent against a single step of the first proposed method, when a single step of the first proposed method actually contains n individual gradient descent steps. A fair comparison would be to run gradient descent for N iterations and the first proposed method for $\frac{N}{n}$ iterations. If this were done the method would likely look to perform even worse.

The major drawbacks of the first approach informs part of the design of the second approach (GGC) described in Chapter 5. This includes both making use of the objective function values and having a stronger motivation to the underlying model. It is likely part of the reason GGC was able to show some success on neural network problems is due to the first lesson of making use of the objective function values. However it is the second aspect of having a stronger motivation for the model that is arguably more important.

GGC was formulated by considering the data available, what we want to infer from the data, and how we can build a model that describes that relationship. Then we select the particular components for the model, and the algorithm follows from that. Improvements to the optimisation algorithm can be made by reconsidering aspects of the model, and then testing the algorithm that results from the new model. Contrast this with approaches to improving on gradient descent, which take the algorithm and then try to make changes that counteract undesirable observed behaviour, such as introducing momentum to dampen oscillations (Section 2.2).

While GGC does not show outstanding performance, there are enough promising prop-

erties in the results to warrant trying to improve the method further, especially the fact that it is able to outperform Adam briefly. The way the method is designed accommodates making improvements motivation behind them. Section 7.3 outlines some of the possibilities for such extensions.

6.1.1 Moving to the Most Likely Location

GGC uses the location that is most likely to be the global optimum as the next point to visit. Gradient descent and its derivatives arguably do the same thing by only using local information or simplistic encodings of past information. This approach seems destined to halt prematurely as there is too much of a focus on exploiting local information. A better approach would perhaps be to keep track of previous promising points, but then aim to visit areas the model doesn't know much about. This could be done using an acquisition function like in Bayesian optimisation.

6.1.2 Inconsistent Units

A common feature of iterative gradient-based numerical optimisation methods is directly adding or subtracting $\boldsymbol{x}$ and $\nabla \boldsymbol{x}$ with only a constant multiplier. GGC does this in formulating the likelihood of the gradients (Equation 5.17). While the dimensionality lines up, the two quantities are in different spaces; one is the domain of the objective function, and one is the domain of the gradients of the objective function. Improvements could be made by addressing this issue.

6.2 Using the Rank

One of the notable features of GGC is its use of the relevancy α, defined as the inverse rank of the points seen according to their objective function values (Equation 5.14). There are several aspects of this worth discussing further.

In Section 5.3.3, the assumption is made that the likelihood of the relevancy $P(\alpha|\boldsymbol{x}, \hat{\boldsymbol{x}})$ is uniformly distributed, implying that knowing x and $\hat{\boldsymbol{x}}$ does not inform us at all about the relative ranking of those points. This makes sense in the context of the naive Bayes inspired assumption that all data tuples are conditionally independent given $\hat{\boldsymbol{x}}$, but it seems wrong to use a ranking at all given that assumption because it doesn't seem possible to use compute the rank and therefore relevancy in the first place. This is an area that could be further explored.

The relevancy instead ends up entering the equation for the posterior via the log likelihood of the gradients given by Equation 5.18. This is allowed by the model because the relevancy is used as a substitute measure to provide similar information to the loss, and there is a dependency relationship from the loss to the gradient. However it does bring about further questions about the validity of the model, and any developments to this method should address that.

Another unusual feature is the way the mode for the posterior (Equation 5.21) ends up with both relevancy and squared relevancy terms. It would be interesting to explore further what is actually happening in steps of the algorithm and how these terms contribute.

6.3 Prior Acting as a Regularisation

Using a zero mean prior with small variance had some success in controlling the erratic behaviour of GGC, as illustrated in Figure 5.7. This is effectively performing regularisation on

the weights of the network by encouraging smaller weights. Regularisation by encouraging smaller weights is a common way to try and improve the performance when training neural networks [15], so it is encouraging to see the potential for this behaviour in the proposed method.

However this model has the potential to be even more powerful, as the prior is not just limited to the one we set. There considerable flexibility in how the prior can be defined, and with this comes flexibility in imposing desired results from the optimiser.

Chapter 7

Conclusions and Future Work

7.1 Report Summary

This report introduces and motivates the goal of the project as aiming to outperform gradient descent by combining the information of multiple samples of the search space. It provides a background context for how this problem fits into machine learning and what informs the methods developed in the project. It describes the testing framework used to compare optimisation algorithms to produce the results that are presented. Two optimisation methods are motivated, described, implemented, and tested. The algorithms are discussed, with a focus on the second method, which shows promise to be further developed.

7.2 Conclusion

The main finding of this work is providing a new way to think about optimisation problems and developing an extendable method based on this. Optimisation is effectively an unsupervised learning task where we want to learn the location of the global optimum of a function given observed points of that function and their gradients and loss values. This lends itself to being framed as a generative problem where the the optimum generates this data. We provide the algorithm Generative Gradient Consensus (GGC) that learns this relationship. The method shows some promising results on one neural network including outperforming Adam in one situation. The algorithm is able to be improved by building upon the underlying model and then implementing the algorithm that results from the new model. As such, we propose possibilities for future work based on GGC.

7.3 Future Work

There is a great deal of potential for future work in investigating and extending the second approach, GGC. The form presented in this report shows reasonable performance in some situations, but there are many opportunities to change and extend the model and study the effects.

Further insight could be gained by understanding why the formula for the mode of the posterior in the form given here (Equation 5.21) is able to optimise effectively in some situations. While it bears some similarity to an averaged gradient descent with weighting based on the best ranked points, the particular form it takes with both rank and squared rank terms is not clear geometrically.

The model was based on many simplifying assumptions. Changing or removing these assumptions and testing the resulting model is likely to lead to different performance. This

includes trying different structures for the probabilistic graphical model in Figure 5.2, or using a less severe assumption than the one similar to naive Bayes that we make.

There is also potential in developing different forms of the mean and covariance for while keeping a multivariate normal representing the likelihood of the gradients. Developing several examples for each of these and systematically testing them could be an entire project by itself. Possibilities for mean functions include those that make use of loss directly, and trying to overcome the unit issue discussed in Section 6.1.2. The covariance could be chosen to vary for each dimensions, and could be adapted as learning proceeds to achieve some of the benefits of adaptive learning rate methods but formulated in a way that is more theoretically grounded.

The way the method learns could be improved, such as changing the current behaviour of going straight to the point calculated to be the most likely. An avenue for such improvement is attempting to implement a learning process that takes advantage of the fact that the prior and likelihood of the gradients are conjugate distributions; that is they both come from the same family and stay in that family when combined. A full Bayesian inference process could then be followed, where the prior and likelihood are combined to form a posterior, which is then used as the prior for the next step.

Another possible extension is using different distribution families altogether.

The thread linking all of the above suggestions for future work together is that they involve making changes to the underlying theoretical model we have developed. The optimisation algorithm that any changed model suggests would then follow from that model, by contrast to other approaches such as adaptive learning rate methods that attempt to change the algorithm behaviour directly. This provides a convenient way to reason about the problem, and then produce an algorithm that is based on that reasoning.

Bibliography

[1] ALLAIRE, G., KABER, S. M., TRABELSI, K., AND ALLAIRE, G. *Numerical linear algebra*, vol. 55. Springer, 2008.

[2] AMARI, S.-I. Natural gradient works efficiently in learning. *Neural computation 10*, 2 (1998), 251–276.

[3] BOLSTAD, W. M., AND CURRAN, J. M. *Introduction to Bayesian statistics*. John Wiley & Sons, 2016.

[4] BONNANS, J.-F., GILBERT, J. C., LEMARÉCHAL, C., AND SAGASTIZÁBAL, C. A. *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media, 2006.

[5] BOYD, S., BOYD, S. P., AND VANDENBERGHE, L. *Convex optimization*. Cambridge university press, 2004.

[6] CAUCHY, A., ET AL. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris 25*, 1847 (1847), 536–538.

[7] CHEN, X., LIU, X., AND JIA, Y. A soft target method of learning posterior pseudo-probabilities based classifiers with its application to handwritten digit recognition. In *2008 11th International Conference on Frontiers in Handwriting Recognition* (2008).

[8] CHEN, X., LIU, X., AND JIA, Y. Combining evolution strategy and gradient descent method for discriminative learning of bayesian classifiers. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (2009), pp. 507–514.

[9] CHOFFIN, B., AND UEDA, N. Scaling bayesian optimization up to higher dimensions: a review and comparison of recent algorithms. In *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)* (2018), IEEE, pp. 1–6.

[10] DE JONG, K. Evolutionary computation: a unified approach. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2017), pp. 373–388.

[11] DEFAZIO, A., BACH, F., AND LACOSTE-JULIEN, S. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in neural information processing systems 27* (2014).

[12] DORIGO, M., BIRATTARI, M., AND STUTZLE, T. Ant colony optimization. *IEEE computational intelligence magazine 1*, 4 (2006), 28–39.

[13] DUCHI, J., HAZAN, E., AND SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research 12*, 7 (2011).

[14] GIVENS, G. H., AND HOETING, J. A. *Computational statistics*, vol. 703. John Wiley & Sons, 2012.

[15] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep learning*. MIT press, 2016.

[16] GORBUNOV, E., HANZELY, F., AND RICHTÁRIK, P. A unified theory of sgd: Variance reduction, sampling, quantization and coordinate descent. In *International Conference on Artificial Intelligence and Statistics* (2020), PMLR, pp. 680–690.

[17] HANSEN, N., AND OSTERMEIER, A. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation* (1996), IEEE, pp. 312–317.

[18] HIGHAM, N. J. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[19] JOHNSON, R., AND ZHANG, T. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems 26* (2013).

[20] KENNEDY, J., AND EBERHART, R. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (1995), vol. 4, IEEE, pp. 1942–1948.

[21] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[22] KIRKPATRICK, S., GELATT JR, C. D., AND VECCHI, M. P. Optimization by simulated annealing. *science 220*, 4598 (1983), 671–680.

[23] KRIZHEVSKY, A. Learning multiple layers of features from tiny images. Tech. rep., CIFAR, 2009.

[24] MARTENS, J., ET AL. Deep learning via hessian-free optimization. In *ICML* (2010), vol. 27, pp. 735–742.

[25] MARTINS, J. R., AND NING, A. *Engineering design optimization*. Cambridge University Press, 2021.

[26] MOCKUS, J., TIESIS, V., AND ZILINSKAS, A. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization 2*, 117-129 (1978), 2.

[27] MOLGA, M., AND SMUTNICKI, C. Test functions for optimization needs. *Test functions for optimization needs 101* (2005), 48.

[28] MURPHY, K. P. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[29] NG, A., AND JORDAN, M. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems 14* (2001).

[30] NOCEDAL, J., AND WRIGHT, S. J. *Numerical optimization*. Springer, 1999.

[31] PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L., DESMAISON, A., KOPF, A., YANG, E., DEVITO, Z., RAISON, M., TEJANI, A., CHILAMKURTHY, S., STEINER, B., FANG, L., BAI, J., AND CHINTALA, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[32] QIAN, N. On the momentum term in gradient descent learning algorithms. *Neural networks 12*, 1 (1999), 145–151.

[33] RANA, S., LI, C., GUPTA, S., NGUYEN, V., AND VENKATESH, S. High dimensional bayesian optimization with elastic gaussian process. In *International conference on machine learning* (2017), PMLR, pp. 2883–2891.

[34] ROUX, N., SCHMIDT, M., AND BACH, F. A stochastic gradient method with an exponential convergence _rate for finite training sets. *Advances in neural information processing systems 25* (2012).

[35] RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[36] SHAHRIARI, B., SWERSKY, K., WANG, Z., ADAMS, R. P., AND DE FREITAS, N. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE 104*, 1 (2015), 148–175.

[37] SHEWCHUK, J. R., ET AL. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[38] STEWART, J. *Essential calculus: Early transcendentals*. Cengage Learning, 2012.

[39] SUN, S., CAO, Z., ZHU, H., AND ZHAO, J. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics 50*, 8 (2019), 3668–3681.

[40] SUTTON, R. Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986* (1986), pp. 823–832.

[41] TIELEMAN, T., HINTON, G., ET AL. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning 4*, 2 (2012), 26–31.

[42] TONG, Y. L. *The multivariate normal distribution*. Springer Science & Business Media, 2012.

[43] TRIPP, A., DAXBERGER, E., AND HERNÁNDEZ-LOBATO, J. M. Sample-efficient optimization in the latent space of deep generative models via weighted retraining. *Advances in Neural Information Processing Systems 33* (2020), 11259–11272.

[44] WANG, Z., ZOGHI, M., HUTTER, F., MATHESON, D., DE FREITAS, N., ET AL. Bayesian optimization in high dimensions via random embeddings. In *IJCAI* (2013), Citeseer, pp. 1778–1784.

[45] WILSON, A. C., ROELOFS, R., STERN, M., SREBRO, N., AND RECHT, B. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems 30* (2017).

[46] WRIGHT, S. J. Coordinate descent algorithms. *Mathematical Programming 151*, 1 (2015), 3–34.

[47] WU, J., POLOCZEK, M., WILSON, A. G., AND FRAZIER, P. Bayesian optimization with gradients. *Advances in neural information processing systems 30* (2017).

[48] XIAO, H., RASUL, K., AND VOLLGRAF, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[49] ZHANG, H. The optimality of naive bayes. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference* (2004), pp. 562–567.