Neural Zeroth-Order Optimization: Empirical Study and Insights

Akhil Nadigatla

April 2022

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science.

Keywords: Optimization, Black-Box, Bandits, Neural Networks.

To my parents.

Abstract

In zeroth-order optimization, we are concerned with optimizing some objective function f when we only have access to it via black-box queries *i.e.,* we are limited to function evaluations alone. Zeroth-order optimization is prevalent in several settings, including computer science, medicine, material science, and chemistry. In machine learning, for instance, problems such as hyperparameter tuning and neural architecture search can be posed as zeroth-order optimization problems.

Over the years, numerous algorithms have been proposed for performing zeroth-order optimization. However, existing techniques exhibit one or more of the following drawbacks: (1) they make restrictive structural assumptions about f which rarely hold in practice, (2) they are computationally expensive and do not scale well to high-dimensional problems, and/or (3) they make too many queries to the zeroth-order oracle.

Neural networks are beneficial in zeroth-order optimization as they can arbitrarily approximate any continuous function and be extended to a variety of domains with ease. Towards this end, we propose a simple yet principled greedy algorithm leveraging the power of neural networks (which fits a neural network to the observed data and uses the learned network as the acquisition function). Our goal with this algorithm is to simulate Bayesian optimization techniques like Gaussian processes with Thompson sampling while avoiding (1) the high complexity issues of non-parametric methods, and (2) the task of defining a suitable kernel. We study our technique empirically and find that it outperforms Gaussian processes with expected improvement in high dimensional settings (with respect to simple regret). Finally, we provide theoretical insights as to why our algorithm works so well and outline future directions of interest.

Acknowledgments

I would like to thank Prof. Pradeep Ravikumar of the Machine Learning Department for serving as my research advisor for this senior thesis project. I would also like to extend my gratitude to Dr. Arun Sai Suggala at Google Research and Biswajit Paria of the Machine Learning Department for greatly helping me over the duration of this investigation. This was a great learning experience that would not have been possible without the continued help and guidance of these individuals.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Let us consider the following problem: suppose we want to minimize (or maximize) some function within the bounds of a provided domain. A reasonable initial thought could be to use gradient information of this objective, employing 'first-order' techniques like gradient descent or its many variants. An alternative could be to utilize optimization algorithms like Newton-Raphson's method that relies on the Hessian *i.e.,* 'second-order' details about the underlying function.

However, what if we do not have access to either piece of information? What if our function of interest is a black-box that we can only interact with via point queries? This is the paradigm within which zeroth-order optimization techniques operate.

Zeroth-order optimization is an important problem with applications in a number of settings including computer science, material science [1], medicine [2], chemistry [3], and biology [4]. For instance, in machine learning, zeroth-order optimization techniques are often used for hyperparameter tuning [5] and neural architecture search [6].

Zeroth-order optimization problems also appear in robust machine learning – the setting could be such that an adversary attempts to introduce imperceptible changes (or perturbations) to the inputs of a neural network with the goal of making the network misclassify them [7, 8]. Defending against such adversarial attacks often requires the use of zeroth-order optimization techniques which can efficiently identify the worst possible perturbation for any given example. These worst perturbations are subsequently used to train an adversarially-robust neural network [9].

Zeroth-order optimization has also been proven to be helpful for model interpretability. What makes this particularly interesting is that assuming a black-box setting affords a model-agnostic method for providing explanations. In a structured data regime, it has been shown that class probabilities for a desired input can alone be used to provide contrastive explanations: ones which not only convey which features are minimally sufficient to justify the predicted class of a particular input, but also which features should be minimally necessarily absent (as per some determined threshold) to maintain the original classification. Hence, techniques like MACEM only rely on a classification model providing class probabilities for the desired input [10].

Zeroth-order optimization problems appear in other engineering disciplines as well. The tech-

niques used to solve them could be used in engineering design to help expedite the search for promising designs [11]. They also have applications in differential privacy, where we may intentionally want to hide first-order information while still trying to minimize some regularized empirical risk function [12, 13]. More recently, zeroth-order optimization has even been used to design better culinary recipes [14]!

Due to its widespread prevalence, zeroth-order optimization has been a constant subject of study in various fields. Over the years, numerous techniques have been developed for solving this problem. At a high level, these algorithms can be classified into two broad categories: model-based and model-free techniques. Model-based techniques, as the name suggests, aim to construct a good surrogate model for the unknown function and rely on this model in order to compute a global optimum. On the other hand, model-free techniques do not rely on such approximations – rather, they perform random walks through the search space. Popular model-free zeroth-order optimization techniques include simulated annealing [15], genetic and evolutionary algorithms [16, 17], consensus-based optimization [18], tabu search methods [19, 20], and particle swarm techniques [21].

In this work, our focus is on model-based techniques as they typically make fewer queries to the black-box oracle relative to model-free techniques. Algorithms that make fewer queries are especially useful in scenarios where calls to the zeroth-order oracle are very expensive – such as exploring automotive designs based on simulations that may take from hours to days each before completion [22].

Within the class of model-based techniques, there are two major sub-categories. One set of approaches impose structural assumptions on the unknown objective function, with the hope of making the zeroth-order optimization more manageable in this way. For example, one could assume that the black-box function is linear [23, 24]. Some works assume that the function is convex [25, 26], some presume strong convexity [27], while others assume that the function is both convex and smooth [28, 29].

The other group of approaches do not make any assumptions about the structure of the underlying function, or minimal ones like smoothness in that the payoff function satisfies Lipschitz continuity [30, 31]. Some rely on Bayesian optimization mechanisms like Gaussian processes [32] – an approach that has been extensively scrutinized in the past few years.

Both of the aforementioned categories of zeroth-order optimization techniques have their pros and cons. The structural assumptions made by the former class of techniques do not often hold in practice, while the latter approaches have high sample complexity. Neural networks can get the best of both worlds. Here, the assumption is that the unknown function can be modeled using a neural network. While this may seem like a structural assumption at first glance, it is a reasonable one to make given that neural networks are 'almost' non-parametric and have been shown to have the power to arbitrarily approximate any continuous function [33]. At the same time, they do not face the high sample complexity issues of non-parametric methods like Gaussian processes [34, 35]. In addition, using neural networks avails two other advantages:

- Techniques based on neural networks can easily be extended to domains such as computer vision, natural language processing, and computational biology, where we have neural ar-

chitectures which can encode priors as per the domain (for instance, convolutional neural networks encode priors that are based on the functioning of the human visual cortex). More broadly, it is easy to incorporate inductive biases into neural networks. Moreover, neural networks provide us the flexibility to incorporate a wide variety of constraints, which may serve as data augmentation techniques [36]. Consequently, neural network-based algorithms can have lower sample complexity compared to classical non-parametric techniques like Gaussian processes.

- Deep learning frameworks like PyTorch and Tensorflow allow for efficient implementation of neural networks and can overcome the computational barrier otherwise associated with the implementation of Gaussian processes.

A number of recent works have attempted to develop neural network-based algorithms for zeroth-order optimization. In general, they try to extend the classic Upper Confidence Bound and Thompson sampling approaches to neural networks (see chapter 3 for more details on these particular algorithms). The downsides of these techniques are that they either require posterior sampling or the establishment of confidence bands for predictions made by neural networks - which are often non-trivial and computationally expensive to perform. As a result, existing approaches resort to heuristics which are not well studied [37].

In this work, we consider the problem of zeroth-order optimization in both the noiseless (where we have access to exact function evaluations) and noisy (where results from oracle queries are assumed to contain some Gaussian noise) settings. Our main contribution here is to develop a greedy algorithm and show that it provides good empirical performance when compared to baselines such as Gaussian processes (which are often touted as the 'gold standard' for zeroth-order optimization). We also present some theoretical insights towards explaining why our seemingly simple technique appears to perform as well as it does. Our empirical and theoretical results suggest that complex algorithms relying on Upper Confidence Bound or Thompson sampling are unnecessary for general purposes. Moreover, they imply that there is a need for more adaptive measures of complexity for zeroth-order optimization, as there appears to be a mismatch between what is expected based on theoretical principles versus what is witnessed in practice.

The thesis is structured as follows: in chapter 2, we formally introduce the problem at hand. In chapter 3, we present necessary background on the landscape of zeroth order optimization techniques. In chapter 4, we describe our algorithm and, in chapter 5, present empirical evidence demonstrating the performance of our algorithm. In chapter 6, we try to understand the workings of our algorithm from a theoretical perspective and present other interesting phenomena witnessed during our investigation. Finally, in chapter 7, we conclude the discussion with some future directions of interest.

Chapter 2

Background

At a high level, zeroth-order optimization can be viewed as a more challenging version of the multi-armed bandits problem [38]. In its simplest formulation, the bandit problem consists of a set of k 'arms' $\mathcal{A} = \{a_1, \ldots, a_k\}$ over some domain $\mathcal{D}$. Metaphorically pulling each arm yields some real result, the crucial detail here being that the value associated with each arm is initially unknown. At each time step $t = 1, 2, \ldots$, the player chooses one of the k available arms and receives some reward. The goal of the player is two-fold: (1) identifying the arm that provides the greatest reward, and (2) maximizing the cumulative rewards over the duration of this game. Oftentimes, the problem is defined such that a budget of T is set over which to accumulate as much reward as possible.

Zeroth-order optimization relates to the multi-armed bandits problem in that there is now an infinite number of arms, each representative of a point in d-dimensional space [39]. Let us express our goal in zeroth-order optimization more formally: we want to minimize some function f over a prescribed set $\mathcal{X} \subseteq \mathbb{R}^d$

$$f^* = \min_{x \in \mathcal{X}} f(x),$$

when we are limited in the ways in which we can interact with the function. In particular, we only have access to f via point evaluations *i.e.,* by querying a black-box (or zeroth-order) oracle. This, in turn, outputs a potentially noisy estimate of the function when queried at any $x \in \mathcal{X}$

$$y = f(x) + \xi,$$

where ξ is assumed to be a mean-zero random variable. Our goal is to find an approximate minimizer of f while making as few oracle queries as possible. In other words, a zeroth-order optimization algorithm performs a sequence of queries $x_1, x_2, \ldots, x_T$ and uses the acquired information to output some point $\hat{x}_T$ as a potential minimizer of f. Once again, notice the analogs to the multi-armed bandits problem, with the main difference being that the arms there are finite and discrete.

In line with the aforementioned goals in a bandits problem, the performance of a zeroth-order optimization technique is typically measured using one of the following optimality criteria

$$f(\hat{x}_T) - f^* \quad \text{(Simple Regret)},$$
$$\frac{1}{T}\sum_{t=1}^{T}(f(x_t) - f^*) \quad \text{(Cumulative Regret)}.$$

The difference in these criteria is highlighted by a concern common in most online decision-making problems: whether to make the best decision given current information (*exploitation*), or gather more information (*exploration*). In our case, this dilemma manifests in the following way: do we make our next query at the best point seen thus far or do we search for some previously unseen point to visit? This is a trade off that every zeroth-order optimization technique needs to balance – determined by whether the focus is on short-term or long-term rewards, and the levels of acceptable risk.

Chapter 3

Prior Work

Zeroth-order optimization is not a new field of study. Owing to its importance, the subject has received a lot of attention from various research communities and, over the years, numerous techniques have been proposed towards solving this problem. Let us briefly examine some popular algorithms used. As previously mentioned, bandit optimization is closely related to zeroth-order optimization, so we discuss here algorithms traditionally developed for the multi-armed bandits problem that have served as the basis for associated zeroth-order optimization algorithms – which boils down to adapting these algorithms to an infinite number of arms.

3.1 Bandit Optimization Techniques

Upper Confidence Bound (UCB): The mechanics of the Upper Confidence Bound-based algorithm for bandit optimization are simple: at each time step, we choose the arm that yielded the highest empirical reward up to that point, *plus* an additional term that is inversely proportional to the number of times that specific arm was pulled [40]. Hence, the UCB algorithm aims to vary the exploration-exploitation balance based on knowledge it gathers at each time step of operation.

For arms $\{a_1, \ldots, a_k\}$ during $t = 1, \ldots, T$, let

- $Q_t(a)$ be the estimated value associated with arm a at time step t.
- $N_t(a)$ be the number of times arm a has been pulled prior to time step t.

The first term in the optimization objective of the UCB algorithm controls its exploitation incentive. It biases the algorithm towards arms that have yielded the highest estimated reward up to that point (usually, this is the average reward across all $N_t(a)$ times that a was previously selected).

Intuitively, the additional term in the objective helps to avoid pulling the same arm without exploring the other available arms first. Therefore, the hyperparameter c is a confidence value that controls the level of exploration performed in this algorithm. The expected cumulative regret of this simple UCB algorithm is logarithmic in T, the total budget [41].

Algorithm 1: Upper Confidence Bound

Input: Hyperparameter c.

for $t = 0, 1, 2, \ldots, T$ **do**

  Execute

$$a_t = \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

  and observe the reward r_t.

  Update $Q_{t+1}(a_t)$ using r_t.

end

Extending this finite-arm UCB algorithm to the infinite-arm setting (which is equivalent to a zeroth-order optimization problem) may simply involve, for each time step, randomly sampling the domain $(\mathcal{X})$ k times and running the algorithm as shown [42]. Hence, the only difference is that the decision set under consideration changes for every iteration. It is important to note that this only describes one way to adapt the bandit version of this algorithm to a continuous, infinite setting: there are many possible approaches to do the same [43].

Thompson Sampling: Also referred to as the Bayesian bandits algorithm, Thompson sampling is another popular multi-armed bandits algorithm [44]. The main idea here is to compute a posterior distribution of each arm being optimal given the information accumulated up to the current time step, and then sampling an arm from this distribution to pull. More generally, what this means is that we are considering distributions π over the space of parameters that completely define a problem instance $\theta \in \Theta$. In the multi-armed bandits case, θ could encode the raw reward values for a set of arms, or even their distributions (as is the case for stochastic bandits).

Algorithm 2: Thompson Sampling

Input: Prior distribution over models, $\pi_0 : \theta \in \Theta \rightarrow [0, 1]$.

for $t = 0, 1, 2, \ldots, T$ **do**

  Sample model $\theta_t \sim \pi_t$.

  Compute $a_t = \operatorname{argmax}_a \mathbb{E}_{\theta_t}[r_t | a_t = a]$.

  Select arm a_t and observe reward r_t.

  Update posterior distribution π_{t+1} using (a_t, r_t).

end

Of course, the tricky part in the case of multi-armed bandits (and accordingly, zeroth-order optimization) is that we lack access to the *actual* posterior distribution π_t given the observed data as of time t. The assumption made in constructing an *approximate* posterior (as done in the algorithm above) is that, if the approximate is close to the actual, then Thompson sampling would yield near-optimal cumulative regret. In particular, it has been shown that Thompson sampling can provide logarithmic regret with respect to the number of rounds, T [45].

However, it is worth noting that the forenamed assumption may not necessarily hold for some problems. For example, it could be the case that exploring a sub-optimal arm could provide useful insights about the nature of the other arms [46]. A downside of Thompson sampling is that it would never select such actions [47]. Moreover, Thompson sampling does not also account for the budget of the algorithm T in balancing exploration versus exploitation unless specific changes are made [48].

One way of extending this Thompson sampling technique from bandit optimization to zeroth-order optimization is identical to the prescription outlined for the UCB-style algorithm above *i.e.,* randomly sampling the domain $\mathcal{X}$ to establish a new set of arms in each round. Note again that this is but one, very simple way of performing this extension.

Gradient Descent-based Techniques: Works in this category hope to leverage (stochastic) gradient descent by computing an approximation of the gradient using point evaluations from the black box. While reasonable, the downside of such approaches is that, in the stochastic setting, they assume that the expected payoffs are a linear or convex function of the arm chosen – something necessary for gradient descent to provide good performance – which can be restrictive. Assuming linearity, it has been shown that we can obtain a regret bound of $O(\sqrt{T})$ [49] while convexity yields us $O(T^{5/6})$ [50].

3.2 Model-based Techniques

Let us shift our attention back to the zeroth-order optimization paradigm, and model-based algorithms in particular. As mentioned in chapter 1, these techniques can be subdivided into ones that make structural assumptions about the unknown black-box function, and ones that do not.

Given a problem of dimensionality d with T rounds of operation, we can summarize the upper bounds on regret achieved by the techniques making structural assumptions as follows:

Assumption	Regret Bound	
Linear	$O\left(\sqrt{\frac{d}{T}}\right)$ or $O\left(\sqrt{\frac{d^2}{T}}\right)$	[51, 52, 53]
Quadratic	$O\left(\sqrt{\frac{d^2}{T}}\right)$	[27]
Strongly Convex + Smooth	$O\left(\sqrt{\frac{d^2}{T}}\right)$	[27]
Strongly Convex	$O\left(\min\left(\sqrt[4]{\frac{d^2}{T}}, \sqrt{\frac{d^{34}}{T}}\right)\right)$	[28, 29]
Convex	$O\left(\min\left(\sqrt[3]{\frac{d^2}{T}}, \sqrt{\frac{d^{34}}{T}}\right)\right)$	[25]

Table 3.1: A summary of regret bounds provided by techniques imposing certain structures to the underlying function f.

On the flip side, techniques that make no structural assumptions about f other than Lipschitz

continuity have been proven to provide regret bounds in $O(T^{-\gamma})$ for $\gamma = \frac{d+1}{d+2}$ [30]. Bayesian optimization techniques like Gaussian processes provide regret bounds in $O(d \log T)$ (assuming linear kernels) [32].

3.3 Neural Network-based Techniques

Neural networks are able to model complex functions in high-dimensional spaces, hence can be used as surrogates for the unknown function f. There exist a variety of neural-network based techniques available to tackle the zeroth-order optimization problem. The ones of interest to our work are discussed in more detail below.

One particular work empirically compared the various Thompson sampling-based approaches using deep Bayesian neural networks [54]. They compare a variety of models including linear (again, where the assumption is that the expected value of the stochastic rewards is an unknown linear function of arm choice), neural linear (neural networks with a linear Bayesian layer at the bottom), variational inference for parameter posterior estimation, dropout, Markov chain Monte Carlo sampling using stochastic gradient descent (for posterior parameter sampling), bootstrap, parameter noise injection, and Gaussian processes. Overall, they found that linear and neural linear approaches performed the best under this bandits setting due to their ability to compute informative uncertainty measures. Another interesting observation noted in this work is that a number of approaches require careful tuning of associated hyperparameters in order to yield the best performance. This is a concern we will revisit in this work as well.

A simple approximation of Thompson sampling is ensemble sampling [55]; here, a collection of neural network models is maintained and, during each iteration, a network is sampled uniformly at random from this set and its minimizer chosen as the next point of query. The obtained information is then used to update all models in said collection. However, this is could become a very expensive endeavor in high-dimensional settings, more so if a large variety of networks are to be considered as part of the ensemble [56].

The connection between wide neural networks and Gaussian processes [57, 58] has also inspired a number of neural network-focused approaches relying on the idea of a neural tangent kernel (NTK) [59]. At a high level, the NTK describes the evolution of deep neural network during training by gradient descent. A key result derived from its application is that wide enough networks converge to a global minimum when trained to minimize empirical risk [60]. As a result, these approaches for zeroth-order optimization are principled and provide theoretical bounds on their regret.

NeuralUCB is a UCB-based approach that relies on infinitely wide neural networks [61]. It can also be extended to a batched setting, where a number of points are queried during a single round before updating the decision policy [62]. NeuralTS is, analogously, a Thompson sampling-based approach that utilizes infinitely wide networks [63]. What differentiates NeuralTS from standard Thompson sampling algorithms is that the posterior is with respect to reward outputs from the neural network rather than the weight parameters themselves. Another UCB-based approach using neural networks computes confidence bounds only for the final *linear* layer rather than for

the whole parameter space [64]. BORE is another recent Bayesian optimization strategy that reformulates the expected improvement acquisition function as density ratio estimation and uses neural networks to solve that problem [65].

Regardless of the wealth of techniques evidently available for solving zeroth-order optimization problems, most algorithms exhibit one or more of the following downsides:

1. They make restrictive structural assumptions about the objective function f such as linearity or convexity, which often do not hold in practice.
2. They are computationally expensive and do not scale well to high-dimensional problems.
3. They make too many queries to the zeroth-order oracle, which can prove to be prohibitive in applications where each function evaluation is expensive.

Our focus in this work is on neural network-based techniques because neural networks have been proven to be good at representation learning and can easily be applied to a variety of problems, from neural architecture search [66] to protein structure discovery [67]. In contrast, Bayesian techniques like Gaussian process-based UCB cannot be readily adapted to such domains as they involve the specification of an appropriate kernel function, which typically requires substantial domain knowledge [68].

Chapter 4

Methodology

We begin our discussion by considering the noiseless setting *i.e.,* querying the black-box oracle returns a deterministic result that can be relied upon to be accurate of the underlying function, and querying the black-box at the same point twice yields the same value. We also assume that T is known beforehand. In tackling the zeroth-order optimization problem, we propose a simple yet powerful algorithm that yields strong empirical results and provides some worthwhile theoretical insights.

4.1 The Neural Greedy Algorithm

4.1.1 Notation and Setup

We begin by introducing notation to describe the neural network that is trained at each iteration of our algorithm. Let d represent the input dimensions for examples, $\mathcal{D}_t \subseteq \mathbb{R}^d \times \mathbb{R}$ denote the training set for the neural network of size n at time step t, with $\mathcal{X}_t = \{x_i : (x_i, y_i) \in \mathcal{D}_t,\ \forall i \in \{1, \ldots, n\}\}$ and $\mathcal{Y}_t = \{y_i : (x_i, y_i) \in \mathcal{D}_t,\ \forall i \in \{1, \ldots, n\}\}$ denoting the inputs and labels. For our problem, $\mathcal{X}_t$ represents the points at which the zeroth-order oracle has been queried up and until time t, and $\mathcal{Y}_t$ the associated return values.

Consider a fully-connected feed-forward neural network with L hidden layers. m_l for all $l \in \{0, \ldots, L+1\}$ represents the width of each layer, with $m_0 = d$ for the input layer and $m_{L+1} = 1$ for the output layer (recall that our unknown objective function is of the form $f : \mathbb{R}^d \to \mathbb{R}$). For each $x \in \mathcal{X}_t$, we use $h_t^l(x)$ and $\alpha_t^l(x)$ to represent the pre- and post-activation functions at layer l for input x.

We can fully define our feed-forward neural network during round t of the zeroth-order optimization algorithm with the following recurrence relation

$$\begin{aligned} &\alpha_t^0 = x, \\ &h_t^l = W_t^l \alpha^{l-1} + b_t^l, \\ &\alpha_t^l = \phi(h_t^l) \end{aligned} \quad \text{and} \quad \begin{cases} W_{t\|i,j}^l &= \frac{\sigma}{\sqrt{m_{l-1}}} \omega_{t\|i,j}^l, \\ b_{t\|i,j}^l &= \beta_{t\|j}^l \end{cases} \qquad \forall l \in \{1, \ldots, L\}$$

for some activation function ϕ, $W_t^l \in \mathbb{R}^{m_{l-1} \times m_l}$ and $b_t^l \in \mathbb{R}^{m_l}$ are the weights and biases, $\omega_{t\|i,j}^l$ and $\beta_{t\|j}^l$ are trainable variables that are drawn i.i.d. from a standard Gaussian distribution $\omega_{t\|i,j}^l, \beta_{t\|j}^l \sim \mathcal{N}(0,1)$ at initialization, and σ^2 is the weight variance.

Note that σ^2 is not a learnable parameter, but rather a hyperparameter used to configure our algorithm – we will revisit the weight variance parameter and understand its significance in the context of our problem. Overall, this parametrization is referred to as the NTK parametrization [69]; what differentiates it from standard parametrization is that, not only does it normalize the forward dynamics of the network, it also normalizes the backward dynamics as well.

We define $\theta_t^l \equiv \text{vec}(\{W_t^l, b_t^l\})$, a vector of all parameters for layer l in the network. Therefore, $\theta_t = \text{vec}\left(\cup_{l=1}^{L+1} \theta_t^l\right)$ indicates all network parameters, with $\theta_{t,t'}$ indicating network parameters during step t' of training the neural network. The final output of the neural network is $f(x;\theta_t) = W_t^{L+1}\alpha_t^L + b_t^{L+1}$. During training, we are interested in learning a θ_t that minimizes square loss $\mathcal{L}(\theta_t, \mathcal{D}_t) = \sum_{i=1}^n \left(f(x_i;\theta_t) - y_i\right)^2$.

4.1.2 Algorithm Definition

Our algorithm consists of two components: (1) an outer runner that controls the algorithm across all T rounds of operation, and (2) an inner procedure conducted for each time step $t \in \{1, \ldots, T\}$. Based on the value of T, the algorithm allocates the first t_e rounds for pure exploration *i.e.,* the points of query are sampled at random from the domain of the function using Latin hypercube sampling: a statistical method that generates a near-random set of samples from a multidimensional domain [70].

Algorithm 3: Neural Greedy

Input: Hyperparameter σ^2, budget T.

Initialize $\mathcal{D}_0 = \emptyset$.
Determine t_e using T.
for $t = 1, 2, \ldots t_e$ **do**
    Sample x_t randomly from the domain.
    Query f at x_t and obtain y_t.
    Update $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(x_t, y_t)\}$.
end
for $t = t_e, t_{e+1}, \ldots, T$ **do**
    Initialize $\theta_{t,0}$.
    Train neural network θ_t on $\mathcal{D}_{t-1}$.
    Find $x_t = \text{argmin}_x f(x;\theta_t)$.
    Query f at x_t to obtain y_t.
    Update $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(x_t, y_t)\}$.
end

Note that our algorithm as presented is in its more general form. There are a lot of hyperparameters and architectural decisions that need to be made once it comes down to real-world

application. Considerations common to all neural network-based algorithms apply here as well: the number of hidden layers, the width of the layers, the learning rate, the optimization technique to use during training, the number of epochs, amongst others [1].

4.2 Motivation

Let us try to motivate why and how we landed on the algorithm defined above. In doing so, let us begin by understanding a popular class of optimization techniques used to solve zeroth-order optimization problems, Bayesian optimization [71]. Any Bayesian optimization algorithm consists of two main components: a Bayesian statistical model for modeling the black-box function f, and an acquisition function for deciding where to query the oracle next. The statistical model at play here is a Gaussian process: a non-parametric supervised learning method that can be used to a learn a posterior distribution over all possible *functions* that fit the data.

Algorithm 4: Generic Bayesian Optimization

Place a Gaussian process prior on f.
Observe f at t_e points according to an initial exploration experimental design.
Set $t = t_e$.
while $t_e \leq T$ **do**
    Update the posterior probability distribution of f using all available data.
    Compute the acquisition function using the current posterior distribution.
    Let x_t be a minimizer of the acquisition function over x.
    Observe $y_t = f(x_t)$.
    Increment t.
end
Output: Either the point evaluated with the largest $f(x)$, or the point with the largest posterior mean.

Our goal with algorithm 3 was to simulate posterior sampling as is the case in Gaussian processes, but now using neural networks. The reason for attempting this is straightforward: as is the case for any Bayesian technique, we want to make the best query using data collected from prior rounds and update our beliefs using data yielded in the current round.

In order to understand the correlation, let us understand how Thompson sampling works for a Gaussian process. Let us consider the basic setting of optimizing a one-dimensional function under the zeroth-order regime, as shown in Figure 4.1.

We begin by querying the black-box oracle at random points within the prescribed domain to gain a rough idea of the function's 'shape', creating a data set similar to that depicted in Figure 4.2.

A Gaussian process can then be fit to the data collected. The uncertainty associated with the resulting approximation manifests itself as a confidence band spread over the mean fit. This is represented by the posterior distribution seen in Figure 4.3.

[1] The specific configurations we used during in our implementation are discussed in Appendix A.

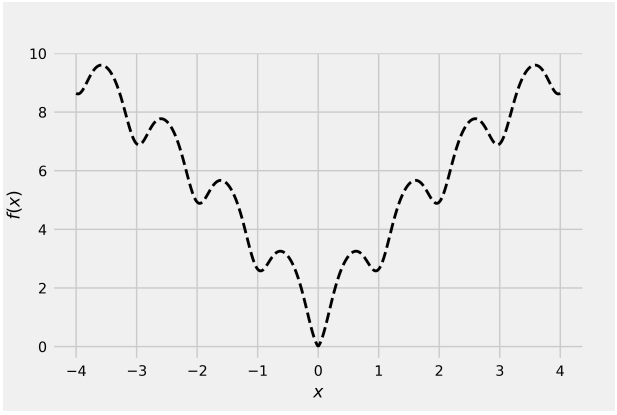


Figure 4.1: A plot of the Ackley function cut to one dimension.

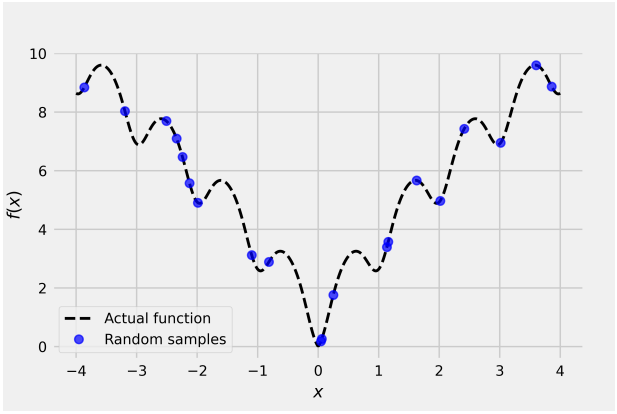


Figure 4.2: A plot of random queries to an oracle with the Ackley function as its objective.

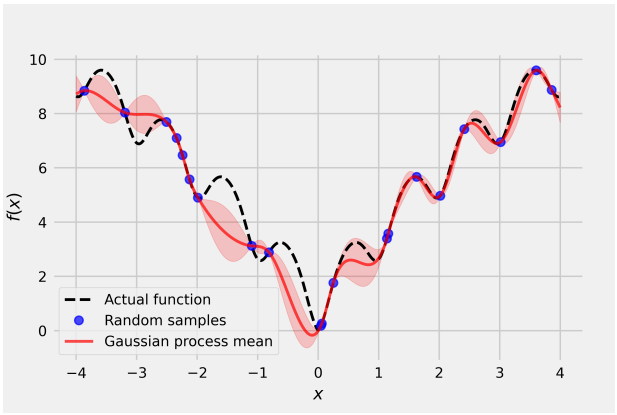


Figure 4.3: A plot showing the Gaussian process fit to the observations seen.

We can draw samples from this posterior, which are essentially candidate approximations to f. These are plotted in Figure 4.2.

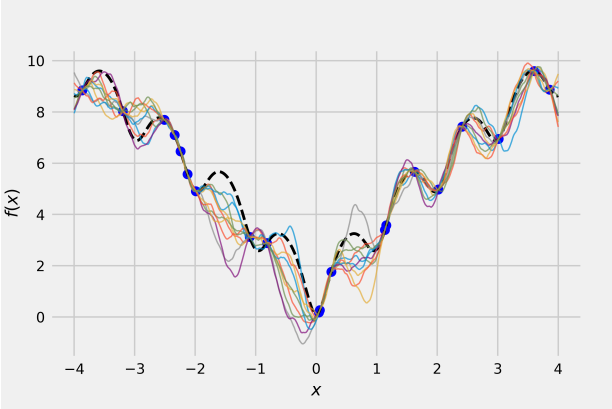


Figure 4.4: A plot showing samples drawn from the posterior distribution of the Gaussian process model.

This is where Thompson sampling comes in. What it does is draw a random sample (a function) from the posterior distribution, which is then greedily minimized in order to determine the next point of query. UCB with Gaussian processes is similar, with the only difference being how the acquisition function is computed from the prior and how the next point of query is selected [72].

Notice the similarities between this technique and our method: instead of posterior sampling from a Gaussian process fit, what we are doing to emulating the same by fitting a neural network. They are multiple possible neural networks that can fit the data – similar to how there are multiple possible samples that can be drawn from the posterior – and the one that ultimately is yielded from training is analogous to the function sample that is finally drawn from that posterior distribution. The neural network analog for a one dimensional problem is shown in Figure 4.5.

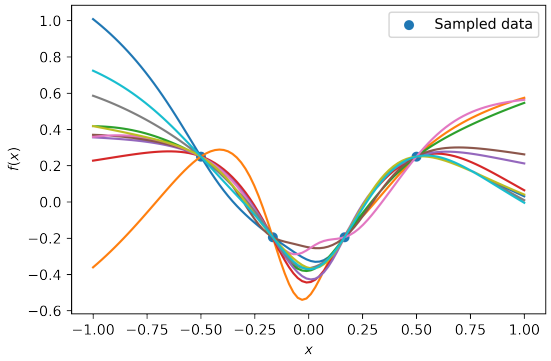


Figure 4.5: Multiple neural networks fit to queries made to a black-box oracle for a one-dimensional function.

By leveraging neural networks, we hope to reap two main benefits: (1) we avoid the task of defining a prior, as is required for Bayesian optimization, which often demands extensive domain knowledge for good performance, and (2) Gaussian processes are known to perform poorly in high dimensional settings [73].

4.3 Tuning σ^2

A hyperparameter required of the Neural Greedy algorithm is σ^2, which denotes the initialization variance for the weight parameters of the neural network. An interesting observation is that σ^2 has a bearing on the 'smoothness' of the neural network trained. We observe that, in our implementation, increasing the variance lead to less smooth solutions. We discuss possible explanations for why this is the case in chapter 6.

The value of σ^2, consequently, biases the algorithm towards a certain set of solutions within the class of all possible neural network fits. In turn, this affects the quality of the approximation of f made by the neural network. Therefore, tuning σ^2 can be an consequential part of running algorithm 3. In our experiments, we selected the initialization variance by simple grid search over a reasonable range and granularity of values.

Chapter 5

Empirical Results

In this chapter, we focus on evaluating the empirical performance of the Neural Greedy algorithm. The baseline algorithm we use for comparison here is Gaussian processes with the expected improvement criterion for optimizing the acquisition function, GP-EI [74]. This was chosen as our baseline because (1) it has been shown to be more well-behaved than alternatives like Gaussian processes with the probability of improvement criterion, and (2) it does not require any tuning parameters, unlike Gaussian processes with UCB [5, 72].

We run the two algorithms on a set of synthetic functions that are commonly used in black-box optimization bench marking and competitions [75, 76]. The functions to test on were chosen such that they vary in dimensionality, modality, smoothness, and structure [1]. The budget T for each function was set – for the sake of experimentation – based on its complexity: black-box functions of higher dimensions and complexity were allocated more rounds.

Table 5.1 shows the results of our investigation assuming noiseless black-box queries, and the graphs in Figure 5.1 depict the rates of convergence of each algorithm across the different synthetic functions. In both cases, the data was gathered and averaged across ten independent runs [2].

Table 5.2 and Figure 5.2 contain the same information as previously mentioned, but now with a noisy black-box oracle for each of the synthetic functions. Noise levels can widely vary in practice: some problems may have very low noise (such as mechanical structure design problems utilizing accurate simulators), while others can exhibit very high amounts of noise (for example, A/B testing new features in internet services). With these experiments, we target the former setting and, correspondingly, limit the standard deviation of the Gaussian noise added to be not more than 10% of f's range.

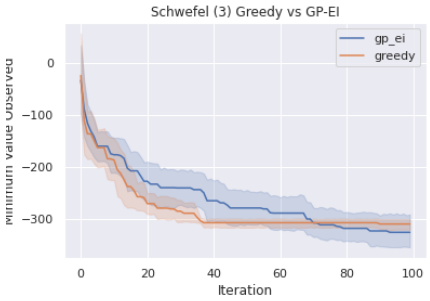
[1] More details regarding the synthetic functions used can be found in Appendix B.

[2] The wall-clock times were determined while running on an Intel® Xeon® Silver 4216 CPU clocked at 2.10 GHz.

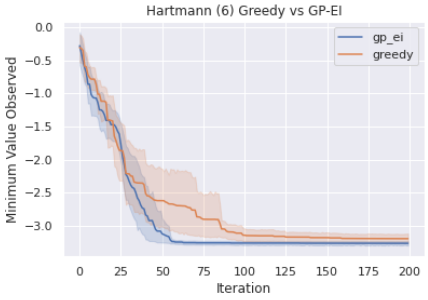
f	d	T	Simple Regret		Cumulative Regret		Wall-clock Time	
			GP-EI	Neural Greedy	GP-EI	Neural Greedy	GP-EI	Neural Greedy
Branin-Hoo	2	50	**0.052**	0.130	**13.567**	17.116	**84**	955
Schwefel	3	100	**93.949**	109.971	379.568	**281.508**	**415**	2769
Hartmann	6	200	**0.051**	0.121	2.276	**1.762**	**2017**	6152
Styblinski-Tang	10	200	94.919	**74.224**	**263.186**	358.300	**2132**	6336
Levy	15	200	9.945	**4.111**	**73.444**	101.974	**2251**	6503
Ackley	20	200	16.741	**13.840**	18.349	**17.399**	**2317**	6936
Rosenbrock	40	500	295095.330	**143254.722**	**1320021.921**	1630700.099	**15698**	29298
Rastrigin	100	1000	**1362.198**	1588.264	**1551.339**	1998.529	**87037**	103067

Table 5.1: A summary of the results achieved by our algorithm against baselines across a variety of synthetic functions assuming a noiseless oracle.

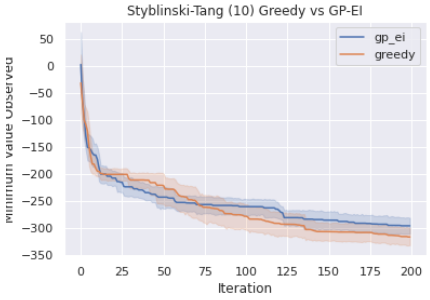
((a)) Branin-Hoo $(d = 2)$



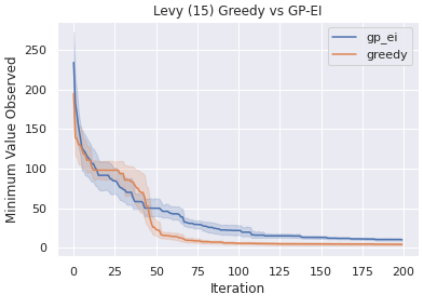
((b)) Schwefel $(d = 3)$



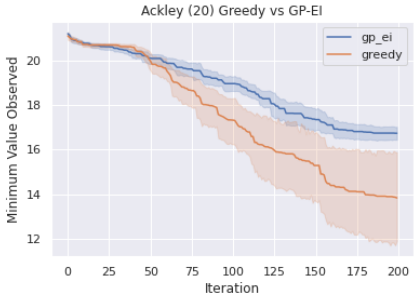
((c)) Hartmann $(d = 6)$



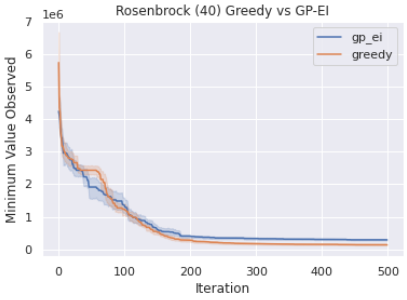
((d)) Styblinski-Tang $(d = 10)$



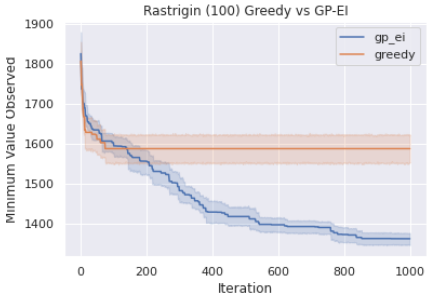
((e)) Levy $(d = 15)$



((f)) Ackley $(d = 20)$



((g)) Rosenbrock $(d = 40)$

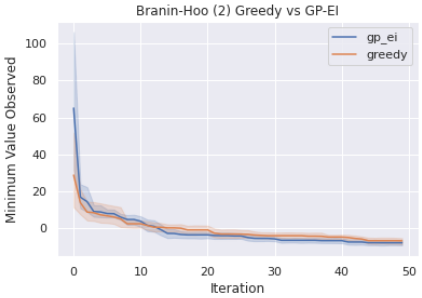


((h)) Rastrigin $(d = 100)$

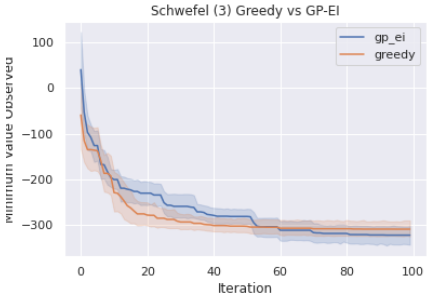
Figure 5.1: Plots indicating the minimum values achieved by the two algorithms across the duration of operation on the synthetic functions (with noiseless oracles), as an indication of the speed of convergence.

f	d	T	**Simple Regret**		**Cumulative Regret**		**Wall-clock Time**	
			GP-EI	Neural Greedy	GP-EI	Neural Greedy	GP-EI	Neural Greedy
Branin-Hoo	2	50	**11.944**	13.005	**15.962**	17.766	**92**	977
Schwefel	3	100	**97.419**	110.451	345.710	**266.438**	**427**	2874
Hartmann	6	200	**0.303**	0.458	**2.460**	2.579	**2162**	6536
Styblinski-Tang	10	200	97.436	**78.119**	**278.407**	378.668	**2262**	6647
Levy	15	200	9.991	**0.270**	**80.117**	98.374	**2357**	6812
Ackley	20	200	**17.268**	17.742	**19.575**	19.910	**2449**	7342
Rosenbrock	40	500	306080.870	**138696.423**	**1312985.224**	1731204.671	**16816**	32290
Rastrigin	100	1000	**1349.849**	1611.106	**1554.041**	1997.395	**88038**	101938

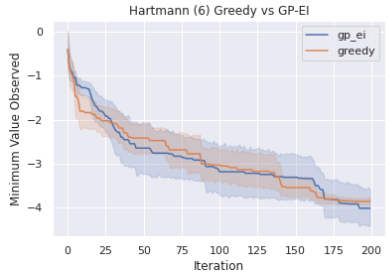
Table 5.2: A summary of the results achieved by our algorithm against baselines across a variety of synthetic functions assuming a noisy oracle.



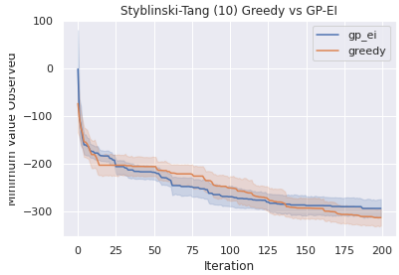
((a)) Branin-Hoo $(d = 2)$



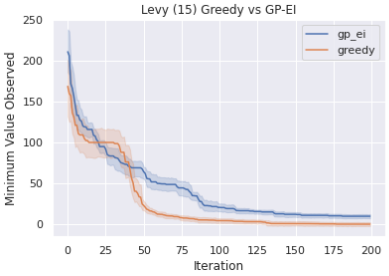
((b)) Schwefel $(d = 3)$



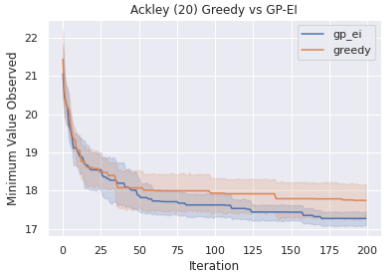
((c)) Hartmann $(d = 6)$



((d)) Styblinski-Tang $(d = 10)$



((e)) Levy $(d = 15)$



((f)) Ackley $(d = 20)$

((g)) Rosenbrock $(d = 40)$

((h)) Rastrigin $(d = 100)$

Figure 5.2: Plots indicating the minimum values achieved by the two algorithms across the duration of operation on the synthetic functions (with noisy oracles), as an indication of the speed of convergence.

Based on the results shown, we make the following observations. In general, as the dimensionality of the problem increases, Neural Greedy tends to dominate GP-EI in terms of simple regret - a phenomenon that is more clearly witnessed in the noiseless setting. An evident shortcoming that appears to be limiting the performance of our technique is that it bears the risk of falling into a local minima and staying there. This is especially apparent in the plots for the Rastrigin function with both noiseless and noisy black-box oracles; while the function does have high modality, GP-EI is still able to continue pursuing a better minima over the course of the experiment.

One possible solution to investigate here could be to add noise to the gradients during neural network training, in order to increase the likelihood of the algorithm escaping such local minima and providing a new network fit that encourages more exploration. Techniques like Stochastic Gradient Langevin Dynamics (SGLD) may prove to be useful in this regard [77]. It could also well be that the choice of σ^2 used for these particular experiments was sub-optimal (please refer to chapter 6 where we elaborate on the significance of this hyperparameter).

When it comes to cumulative regret, the experiment we ran are inconclusive: while Neural Greedy does show some potential in the noiseless case, GP-EI's statistics appear better overall.

With respect to wall-clock time, GP-EI trumps Neural Greedy in every single instance. We reckon that the primary cause for this severe discrepancy is that algorithm 3 requires the training of a new neural network at every single time step. Possible solutions to alleviate this bottleneck could be to provide a warm-start for the training process [78], and/or only training the neural network completely at regular intervals within the overall budget (for instance, once every ten rounds).

Nevertheless, our seemingly simple algorithm provides better-than-expected results. It is an indicator of the power of the neural networks in the context of zeroth-order optimization, and our experiments suggest that - with the right adjustments - even a greedy technique such as ours can provide great performance for practical purposes.

Chapter 6

Theoretical Insights

In this chapter, we attempt to provide an explanation for what our algorithm is doing and why it performs the way that it does. In particular, we provide insights as to how our algorithm selects a function fit and how that is closely related to the hyperparameter σ^2, the initialization variance for the weights of the neural network.

A reasonable question to ask based on our presentation of algorithm 3 is the following: if all we are doing in each iteration of the neural training process is fitting to the data, why do we not converge to a single, smooth function fit with zero training loss? What causes our algorithm to randomly explore over a number of possible function approximations across t rounds of the zeroth-order optimization problem?

The workings of our algorithm are closely related to the concept of implicit regularization. Generally, this describes the phenomenon whereby an optimization algorithm prefers to converge to a certain solution (or class of solutions) in favor of the others in parameter space. When training neural networks, there are several modeling choices made that could implicitly impose a regularization effect on the final solution [79]. Our belief is that σ^2 is biasing our algorithm to a certain solution class.

Let us consider the neural network training process, with $\theta_{t,0}$ denoting the initialized weight parameters during some arbitrary round t of our greedy algorithm. In the overparameterized setting, it has been shown that $\theta_{t,t'}$ (the weight parameters of the neural network after t' training loops) are still quite close to $\theta_{t,0}$ in the parameter space [80]. Consider the overparametrized, single hidden layer networks we operate over in our implementation of Neural Greedy. When overparameterized, it has been shown that the gradients of the *individual* weights and biases tend to 0 [81]. Training still occurs, in that the network's weights 'collectively' modify the solution, with the individual parameters remaining relatively unchanged.

As σ^2 explicitly has a bearing on the initialization parameters (as seen in chapter 4), this is why the choice of its value leads to different neural network fits. Referring back to the question posed earlier, it is σ^2 that allows for this exploration across candidate neural network fits to occur across rounds. In other words, it is the application of the σ^2 hyperparameter that allow us to simulate

Thompson sampling in Gaussian processes, as explained in chapter 4.

Let us now try to understand why exactly σ^2 is controlling the nature of the neural network fit to the training data (in our case, the point queries and results collected up to a certain point in time). To explain this, we rely on neural network training theory for infinite width networks *i.e.,* networks whose widths tend to ∞ and theories surrounding NTKs [59]. The reason for assuming infinitely wide networks is that the characteristics of finite width neural networks are hard to study [82]. Hence, suppose we have an infinitely wide network: it has been shown that gradient of the network with respect to an example x, $\nabla_\theta f_\theta(x)$, remains almost constant over the training process. This has an impact on the training dynamics on the neural network and can be used to characterize $f_{\theta_{t,t'}}(x)$ after t' training steps. In this setting, we can compute the NTK associated with the network as $\Theta(x, x') = \nabla_\theta f_\theta(x)^\top \nabla_\theta f_\theta(x)$.

Another phenomenon of interest here is the following: suppose we randomly initialize the weights of an infinitely wide neural network with no training performed after. It has been proven that this network is a sample from a Gaussian process prior [57], and we can calculate the kernel associated with this fit: $\mathcal{K}(x, x')$. This is referred to as the neural network Gaussian process (NNGP) kernel.

We return to our problem of understanding the variance in neural network fits during a round of the Neural Greedy algorithm. It has been proven that the distribution over all possible solutions can be characterized as a Gaussian process with [1]

$$\begin{aligned}
&\text{mean } \mu(\mathcal{X}_{\text{test}}) = \Theta(\mathcal{X}_{\text{test}}, \mathcal{X}_t)\Theta^{-1}(I - e^{-\eta\Theta t})\mathcal{Y}_t, \text{ and} \\
&\text{covariance } \Sigma(\mathcal{X}_{\text{test}}, \mathcal{X}_{\text{test}}) = \mathcal{K}(\mathcal{X}_{\text{test}}, \mathcal{X}_{\text{test}}) + \Theta(\mathcal{X}_{\text{test}}, \mathcal{X}_t)\Theta^{-1}(I - e^{-\eta\Theta t})\Theta^{-1}\Theta(\mathcal{X}_t, \mathcal{X}_{\text{test}}) \\
&\qquad\qquad - \left(\Theta(\mathcal{X}_{\text{test}}, \mathcal{X}_t)\Theta^{-1}(I - e^{-\eta\Theta t})\mathcal{K}(\mathcal{X}_t, \mathcal{X}_{\text{test}}) + h.c.\right),
\end{aligned}$$

where $\mathcal{X}_{\text{test}}$ refers to the test set, $\mathcal{X}_t$ is our training examples at time t, and η is the learning rate of the training process [69]. This assumes that we are working in an overparametrized setting, which our implementation does. What this means is that the variance of the *implicit* Gaussian process is dependent on the NNGP kernel which, in turn, depends on the initialization variance of the weights of the network, σ^2. Therefore, this explains why σ^2 has an impact on the variance of our solution.

In conclusion, we observe that (1) imposing an initialization variance σ^2 is what causes the algorithm to explore a variety of neural network fits, (2) the value of σ^2 decides the variance of solutions (neural network function fits) possible during an iteration of our algorithm, and (3) the fact that gradient descent in the overparametrized setting leads to solutions that are close to the vicinity of the initial parameters $\theta_{t,0}$ implies that the final, post-training neural network inherits the smoothness of the initialization, which is determined by σ^2.

[1] where '$h.c.$' refers to the Hermitian conjugate.

Chapter 7

Future Work

There are a lot of interesting directions for research that stem from the work described in this thesis. A first step would be to establish explicit regret bounds for the Neural Greedy algorithm. This would allow ease of comparison with other baselines but, based on the empirical performance portrayed by our technique, our inclination is that we need to develop more adaptive measures of complexity in zeroth-order optimization. As it stands, existing complexity measures (like Eluder dimension [83]) used to bound the regret of zeroth-order optimization may not fully explain the behavior of our algorithm.

In terms of empirical investigations, we are yet to understand how our algorithm behaves when training using a multiple hidden layer neural network, as our implementation was restricted to a single hidden layer. Deeper networks may be harder to explain the behavior of, but for a large class of piece-wise smooth functions, the number of units needed by a shallow network to approximate that function to a certain degree is exponentially larger than the corresponding number of units for a deeper network [84].

Furthermore, while the synthetic functions we used provide a general sense of performance, it is essential that we apply our algorithm to higher dimensional problems and even real-world experiments. That would allow us to understand how generalizable our algorithm is.

Another important problem to tackle in the future would be to propose a method for automatically tuning σ^2, the initialization variance of the neural network parameters. We saw in chapter 6 that this hyperparameter has a great impact on the nature of the neural network fit and how well it serves as a surrogate for the black-box function. In our implementation, σ^2 was picked using a basic grid search, which can be slow and inefficient once we move to more complicated zeroth-order optimization problems. A simple, heuristic-based approach could be to find a candidate σ^2 using cross-validation, then incrementing it by some constant - as mentioned in chapter 4, higher initialization variance led to less smooth function fits. This could increase the amount of exploration performed by our algorithm.

Another natural consideration for the question of tuning σ^2 could be to employ some online learning algorithms. For instance, we could fit multiple neural networks with different initializa-

tion variances, each drawn from some intervals of values, and then run an Exp3-style algorithm on top of this to determine the best σ^2 overall: a meta-bandits problem is solved before our technique is run in earnest. Techniques like these have been studied and provide local guarantees that may be worth exploiting [85]. Another alternative could be to adapt estimations commonly utilized in Gaussian process regression that provide adequate theoretical guarantees [86].

Appendix A

Implementation Details

As mentioned in chapter 4, there are several implementation decisions to be made in running the proposed algorithm. For our particular implementation that was used for the experimental results shown above, the set of configurations is as follows [1]. The number of hidden layers was set at 1, as we saw no added benefit of a multi layer network in terms of regret, and single layer networks proved to be expressive enough for our purposes [87].

The width of the network was set at 5000 – a value much larger than the number of observations n used to fit the neural network in all our experiments. It has been shown that shallow, overparametrized neural networks have the capacity to fit the training data with zero error [88]; a neural network is deemed 'overparametrized' if the number of observations being used to perform training are fewer than the number of parameters in the model. Consequently, no regularization term was placed as overfitting was not a concern.

In terms of training the neural network, the number of epochs was set at 5000 – a relatively high number to account for high dimensional problems that may require more compute in order to obtain better fits. The optimizer used in the training process was Adam [89], a popular technique that adaptively adjusts the learning rate of training and employs momentum. Stochastic gradient descent was also explored as an alternative, but no significant difference were witnessed in the context of our experiment set. The learning rate was initially set to 0.001.

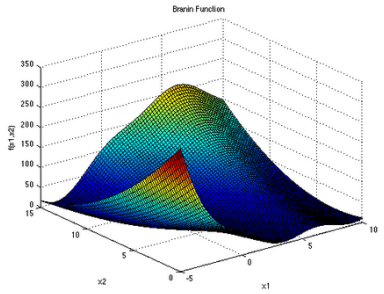
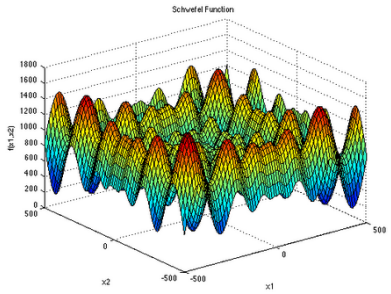
Another important choice to make is regarding to how to minimize the trained neural network in order to determine the next point of query for the black-box oracle (optimizing the acquisition function). A simple method could be to be randomly sample a number of points in the domain of the function, use the neural network to obtain corresponding estimates, and pick the point whose query yielded the minimum value. While this may be efficient, it is certainly not principled. Alternatives to this include COBYLA [90], L-BFGS-B [91], and consensus-based optimization [92]. The optimization technique we settled on particle-based gradient descent due to its global convergence guarantees and relative efficiency [93].

[1] The code can be found here: `https://github.com/biswajitsc/neural-bandits`.

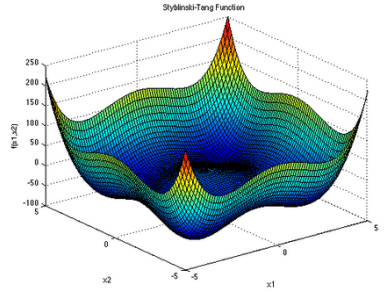
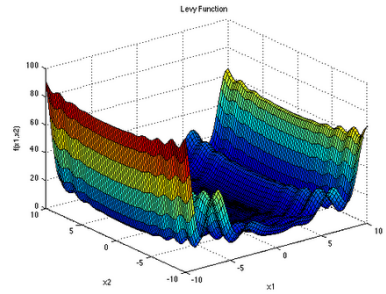
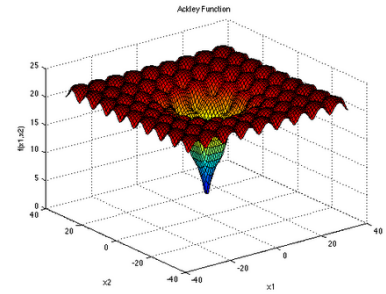
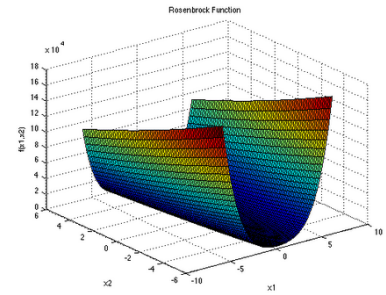
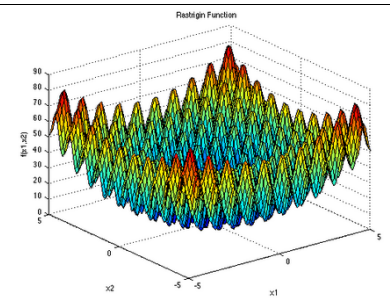
Appendix B

Synthetic Functions

The synthetic functions used to evaluate our algorithm are commonly used to benchmark black-box optimization techniques [94].

Function	Dimensions	Modes	Domain	Visualization
Branin-Hoo	2	3	$[-5, 10] \times [0, 15]$	
Schwefel	3	Many	$[-500, 500]^3$	
Hartmann	6	6	$(0, 1)^6$	$f(\mathbf{x}) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right)$, where $\alpha = (1.0, 1.2, 3.0, 3.2)^T$ $\mathbf{A} = \begin{pmatrix} 10 & 3 & 17 & 3.50 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}$ $\mathbf{P} = 10^{-4} \begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix}$

(Continued on the next page)

Function	Dimensions	Modes	Domain	Visualization
Styblinski-Tang	10	1	$[-5, 5]^5$	
Levy	15	1	$[-10, 10]^{15}$	
Ackley	20	1	$[-32.8, 32.8]^{20}$	
Rosenbrock	40	1	$[-5, 10]^{40}$	
Rastrigin	100	Many	$[-5.12, 5.12]^{100}$	

Bibliography

[1] Alison L Marsden, Jeffrey A Feinstein, and Charles A Taylor. A computational framework for derivative-free optimization of cardiovascular geometries. *Computer methods in applied mechanics and engineering*, 197(21-24):1890–1905, 2008. 1

[2] Alison L Marsden, Meng Wang, JE Dennis, and Parviz Moin. Trailing-edge noise reduction using derivative-free optimization and large-eddy simulation. *Journal of Fluid Mechanics*, 572:13–36, 2007. 1

[3] Stanley N Deming, Lloyd R Parker Jr, and M Bonner Denton. A review of simplex optimization in analytical chemistry. *CRC Critical Reviews in Analytical Chemistry*, 7(3):187–202, 1978. 1

[4] Genetha Anne Gray, Tamara G Kolda, Ken Sale, and Malin M Young. Optimizing an empirical scoring function for transmembrane protein structure determination. *INFORMS Journal on Computing*, 16(4):406–418, 2004. 1

[5] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012. 1, 5

[6] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31, 2018. 1

[7] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 154–169, 2018. 1

[8] Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred O Hero III, and Pramod K Varshney. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 37(5):43–54, 2020. 1

[9] Pin-Yu Chen and Sijia Liu. Recent progress in zeroth order optimization and its applications to adversarial robustness in data mining and machine learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3233–3234, 2019. 1

[10] Amit Dhurandhar, Tejaswini Pedapati, Avinash Balakrishnan, Pin-Yu Chen, Karthikeyan Shanmugam, and Ruchir Puri. Model agnostic contrastive explanations for structured data. *arXiv preprint arXiv:1906.00117*, 2019. 1

[11] András Sobester, Alexander Forrester, and Andy Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008. 1

[12] Cristiano Gratton, Naveen KD Venkategowda, Reza Arablouei, and Stefan Werner. Privacy-preserving distributed zeroth-order optimization. *arXiv preprint arXiv:2008.13468*, 2020. 1

[13] H Brendan McMahan, Galen Andrew, Ulfar Erlingsson, Steve Chien, Ilya Mironov, Nicolas Papernot, and Peter Kairouz. A general approach to adding differential privacy to iterative training procedures. *arXiv preprint arXiv:1812.06210*, 2018. 1

[14] Benjamin Solnik, Daniel Golovin, Greg Kochanski, John Elliot Karro, Subhodeep Moitra, and D Sculley. Bayesian optimization for a better dessert. 2017. 1

[15] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. 1

[16] Jinyin Chen, Mengmeng Su, Shijing Shen, Hui Xiong, and Haibin Zheng. Poba-ga: Perturbation optimized black-box adversarial attacks via genetic algorithm. *Computers & Security*, 85:89–106, 2019. 1

[17] Andranik S Akopov, Levon A Beklaryan, Manoj Thakur, and Bhisham Dev Verma. Parallel multi-agent real-coded genetic algorithm for large-scale black-box single-objective optimisation. *Knowledge-Based Systems*, 174:103–122, 2019. 1

[18] Claudia Totzeck. Trends in consensus-based optimization. *arXiv preprint arXiv:2104.01383*, 2021. 1

[19] Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989. 1

[20] Fred Glover. Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32, 1990. 1

[21] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995. 1

[22] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998. 1

[23] Sarah Filippi, Olivier Cappe, Aurélien Garivier, and Csaba Szepesvári. Parametric bandits: The generalized linear case. *Advances in Neural Information Processing Systems*, 23, 2010. 1

[24] Branislav Kveton, Manzil Zaheer, Csaba Szepesvari, Lihong Li, Mohammad Ghavamzadeh, and Craig Boutilier. Randomized exploration in generalized linear bandits. In *International Conference on Artificial Intelligence and Statistics*, pages 2066–2076. PMLR, 2020. 1

[25] Alekh Agarwal, Dean P Foster, Daniel J Hsu, Sham M Kakade, and Alexander Rakhlin. Stochastic convex optimization with bandit feedback. *Advances in Neural Information Processing Systems*, 24, 2011. 1, **??**

[26] Alexandre Belloni, Tengyuan Liang, Hariharan Narayanan, and Alexander Rakhlin. Escaping the local minima via simulated annealing: Optimization of approximately convex functions. In *Conference on Learning Theory*, pages 240–265. PMLR, 2015. 1

[27] Ohad Shamir. On the complexity of bandit and derivative-free stochastic convex optimization. In *Conference on Learning Theory*, pages 3–24. PMLR, 2013. 1, **??**, **??**

[28] Francis Bach and Vianney Perchet. Highly-smooth zero-th order online optimization. In *Conference on Learning Theory*, pages 257–283. PMLR, 2016. 1, **??**

[29] Krishnakumar Balasubramanian and Saeed Ghadimi. Zeroth-order nonconvex stochastic optimization: Handling constraints, high dimensionality, and saddle points. *Foundations of Computational Mathematics*, pages 1–42, 2021. 1, **??**

[30] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 681–690, 2008. 1, 3.2

[31] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvari. X-armed bandits. *arXiv preprint arXiv:1001.4475*, 2010. 1

[32] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias W Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE transactions on information theory*, 58(5):3250–3265, 2012. 1, 3.2

[33] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017. 1

[34] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015. 1

[35] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. *Advances in neural information processing systems*, 29, 2016. 1

[36] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1529–1537, 2015. 1

[37] Carola Doerr. Complexity theory for discrete black-box optimization heuristics. In *Theory of evolutionary computation*, pages 133–212. Springer, 2020. 1

[38] Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014. 2

[39] Rajeev Agrawal. The continuum-armed bandit problem. *SIAM journal on control and optimization*, 33(6):1926–1951, 1995. 2

[40] Tze Leung Lai, Herbert Robbins, et al. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985. 3.1

[41] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020. 4

[42] Yizao Wang, Jean-Yves Audibert, and Rémi Munos. Algorithms for infinitely many-armed bandits. *Advances in Neural Information Processing Systems*, 21, 2008. 4

[43] Dylan Foster, Alekh Agarwal, Miroslav Dudik, Haipeng Luo, and Robert Schapire. Practical contextual bandits with regression oracles. In *International Conference on Machine Learning*, pages 1539–1548. PMLR, 2018. 4

[44] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933. 4

[45] Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*, pages 39–1. JMLR Workshop and Conference Proceedings, 2012. 6

[46] Samarth Gupta, Shreyas Chaudhari, Gauri Joshi, and Osman Yagan. Multi-armed bandits with correlated arms. *IEEE Transactions on Information Theory*. 6

[47] Daniel Russo and Benjamin Van Roy. Learning to optimize via information-directed sampling. *Advances in Neural Information Processing Systems*, 27, 2014. 6

[48] Daniel Russo, David Tse, and Benjamin Van Roy. Time-sensitive bandit learning and satisficing thompson sampling. *arXiv preprint arXiv:1704.09028*, 2017. 6

[49] Qin Ding, Cho-Jui Hsieh, and James Sharpnack. An efficient algorithm for generalized linear bandit: Online stochastic gradient descent and thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pages 1585–1593. PMLR, 2021. 6

[50] Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. *arXiv preprint cs/0408007*, 2004. 6

[51] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *International conference on Algorithmic learning theory*, pages 23–37. Springer, 2009. **??**

[52] Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. *Advances in neural information processing systems*, 24, 2011. **??**

[53] Jean-Yves Audibert, Sébastien Bubeck, and Gábor Lugosi. Minimax policies for combinatorial prediction games. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 107–132. JMLR Workshop and Conference Proceedings, 2011. **??**

[54] Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018. 3.3

[55] Xiuyuan Lu and Benjamin Van Roy. Ensemble sampling. *Advances in neural information processing systems*, 30, 2017. 3.3

[56] T Poggio and Q Liao. Theory i: Deep networks and the curse of dimensionality. *Bulletin of the Polish Academy of Sciences. Technical Sciences*, 66(6), 2018. 3.3

[57] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017. 3.3, 6

[58] Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019. 3.3

[59] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018. 3.3, 6

[60] Sanjeev Arora, Simon S Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. *arXiv preprint arXiv:1910.01663*, 2019. 3.3

[61] Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural contextual bandits with ucb-based exploration. In *International Conference on Machine Learning*, pages 11492–11502. PMLR, 2020. 3.3

[62] Quanquan Gu, Amin Karbasi, Khashayar Khosravi, Vahab Mirrokni, and Dongruo Zhou. Batched neural bandits. *arXiv preprint arXiv:2102.13028*, 2021. 3.3

[63] Weitong Zhang, Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural thompson sampling. *arXiv preprint arXiv:2010.00827*, 2020. 3.3

[64] Pan Xu, Zheng Wen, Handong Zhao, and Quanquan Gu. Neural contextual bandits with deep representation and shallow exploration. *arXiv preprint arXiv:2012.01780*, 2020. 3.3

[65] Louis C Tiao, Aaron Klein, Matthias W Seeger, Edwin V Bonilla, Cedric Archambeau, and Fabio Ramos. Bore: Bayesian optimization by density-ratio estimation. In *International Conference on Machine Learning*, pages 10289–10300. PMLR, 2021. 3.3

[66] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *Advances in neural information processing systems*, 31, 2018. 3.3

[67] Theodore Papalexopoulos, Christian Tjandraatmadja, Ross Anderson, Juan Pablo Vielma, and David Belanger. Constrained discrete black-box optimization using mixed-integer programming. *arXiv preprint arXiv:2110.09569*, 2021. 3.3

[68] Tom Briggs and Tim Oates. Discovering domain-specific composite kernels. *UMBC Computer Science and Electrical Engineering Department Collection*, 2005. 3.3

[69] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019. 4.1.1, 6

[70] Wei-Liem Loh. On latin hypercube sampling. *The annals of statistics*, 24(5):2058–2080, 1996. 4.1.2

[71] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018. 4.2

[72] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009. 4.2, 5

[73] Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31(11):4405–4423, 2020. 4.2

[74] Adam D Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12(10), 2011. 5

[75] I Loshchilov and T Glasmachers. Black-box optimization competition (bbcomp), 2015. 5

[76] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021. 5

[77] Chunyuan Li, Changyou Chen, David Carlson, and Lawrence Carin. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. 5

[78] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in Neural Information Processing Systems*, 33:3884–3894, 2020. 5

[79] Behnam Neyshabur. Implicit regularization in deep learning. *arXiv preprint arXiv:1709.01953*, 2017. 6

[80] Siddhartha Satpathi and R Srikant. The dynamics of gradient descent for overparametrized neural networks. In *Learning for Dynamics and Control*, pages 373–384. PMLR, 2021. 6

[81] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. PMLR, 2019. 6

[82] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *Advances in Neural Information Processing Systems*, 32, 2019. 6

[83] Daniel Russo and Benjamin Van Roy. Eluder dimension and the sample complexity of optimistic exploration. *Advances in Neural Information Processing Systems*, 26, 2013. 7

[84] Shiyu Liang and Rayadurgam Srikant. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016. 7

[85] Zhou Lu, Wenhan Xia, Sanjeev Arora, and Elad Hazan. Adaptive gradient methods with local guarantees. *arXiv preprint arXiv:2203.01400*, 2022. 7

[86] Aretha L Teckentrup. Convergence of gaussian process regression with estimated hyperparameters and applications in bayesian inverse problems. *SIAM/ASA Journal on Uncertainty Quantification*, 8(4):1310–1337, 2020. 7

[87] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *international conference on machine learning*, pages 2847–2854. PMLR, 2017. A

[88] Mahdi Soltanolkotabi, Adel Javanmard, and Jason D Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 65(2):742–769, 2018. A

[89] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. A

[90] Michael JD Powell. A view of algorithms for optimization without derivatives. *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications*, 43(5):170–174, 2007. A

[91] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995. A

[92] René Pinnau, Claudia Totzeck, Oliver Tse, and Stephan Martin. A consensus-based model for global optimization and its mean-field limit. *Mathematical Models and Methods in Applied Sciences*, 27(01):183–204, 2017. A

[93] Lenaic Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. *Advances in neural information processing systems*, 31, 2018. A

[94] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved April 20, 2022, from `http://www.sfu.ca/~ssurjano`. B