DEAKIN UNIVERSITY

DOCTORAL THESIS

Thesis Title

Author:
Dat PHAN TRONG

Supervisor:
Prof. Sunil GUPTA

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

January 1, 2025

Declaration of Authorship

I, Dat PHAN TRONG, declare that this thesis titled, "Thesis Title" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

List of Abbreviations

BO Black-box Optimization

DNN Deep Neural Network

EI Expected Improvement

ES Entropy Search

PES Predictive Entropy Search

MES Max-value Entropy Search

KG Knowledge Gradient

GP Gaussian Process

LCB Lower Confidence Bound

NTK Neural Tangent Kernel

RKHS Reproducing Kernel Hilbert Space

PDE Partial Differential Equation

PINN Physics-Informed Neural Network

TS Thompson Sampling

UCB Upper Confidence Bound

RF Random Forest

BNN Bayesian Neural Network

Deep GP Deep Gaussian Process

MIG Maximum Information Gain

ReLU Rectified Linear Unit

PI Probability of Improvement

CDF Cumulative Distribution Function

PDF Probability Density Function

MAB Multi-armed Bandit

GD Gradient Descent

NAG Nesterov Accelerated Gradient

Adam Adaptive Moment Estimation

RMSProp Root Mean Square Propagation

CG Conjugate Gradient

DFO Derivative-Free Optimization

GA Genetic Algorithm

SA Simulated Annealing

CMA-ES Covariance Matrix Adaptation Evolution Strategy

List of Symbols

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

For/Dedicated to/To my…

Abstract

Optimization has become an essential task across numerous fields, as many systems and processes require fine-tuning to achieve maximum efficiency. In many cases, these systems are "black-box", meaning that the underlying mechanisms are unknown or too complex to model explicitly; we can only provide inputs and observe the resulting outputs. The challenge with optimizing black-box systems is that evaluating them can often be costly, both in terms of time and resources. This creates a demand for optimization methods that are not only effective but also sample-efficient, minimizing the number of evaluations needed to find optimal solutions.

Bayesian optimization or Black-box Optimization (BO) has emerged as a leading framework for optimizing expensive and unknown functions in various fields, including materials science, biomedical research, and machine learning. BO typically relies on two core components: (1) a surrogate model that approximates the underlying function based on available observations, and (2) an acquisition function that balances the trade-off between exploration (searching unexplored regions with high uncertainty) and exploitation (refining known promising regions). Traditionally, Gaussian Process (GP) has been the preferred choice for the surrogate model due to its ability to quantify uncertainty and its analytic expression, hence being convenient to provide theoretical guarantees. However, GPs suffer from cubic computational complexity in relation to the number of observations, making them impractical for large-scale applications.

First of all, investigating alternative surrogate models to enhance the performance of standard BO, especially in scenarios where GP are computationally prohibitive, is focused. The ability of Deep Neural Networks (DNNs) to scale linearly with the number of data points and capture complex patterns makes them more suitable for tasks involving structural data (like images and text). We introduce **Neural-BO**, a DNN-based optimization algorithm that leverages recent advances in neural network theory to estimate uncertainty and uses Thompson Sampling (TS) for next point selection. This method maintains the flexibility of DNNs while achieving convergence guarantees through the Neural Tangent Kernel (NTK)-based theory, showing improved sample efficiency and faster convergence.

However, optimizing real-world functions often goes beyond simple maximization or minimization, requiring the incorporation of constraints. These constraints ensure that solutions not only optimize the objective but also meet critical feasibility criteria, which is especially important in domains like engineering and physics. Hence, we extend the DNN-based approach to handle optimization problems with expensive, unknown constraints through **Neural-CBO**. In this method, both the objective function and the constraints are modeled using DNNs, and the acquisition function is designed to balance exploration and exploitation while ensuring constraint satisfaction. By using Lower Confidence Bound (LCB) conditions to guarantee feasibility and the Expected Improvement (EI) acquisition function to guide the search, Neural-CBO efficiently explores the feasible region, even when it is significantly smaller than the overall search space. Our method provides upper bounds

on both regret and constraint violations, offering a theoretically sound and scalable approach for constrained black-box optimization tasks.

Moreover, in fields like physics, engineering, and biology, there is often domain-specific knowledge available, typically in the form of physical laws or models, which can be expressed as Partial Differential Equations (PDEs). Incorporating physical knowledge into the optimization process enhances function approximation by providing more accurate guidance, improving surrogate model predictions, and reducing uncertainty in regions that follow known physical laws. Therefore, we propose PINN-BO, an approach that integrates Physics-Informed Neural Network (PINN) into the BO framework. By incorporating these physical constraints into the optimization process, PINN-BO significantly improves sample efficiency. The method employs PINN to model the objective function, utilizing known physical principles by embedding the PDE into the network's training process. This ensures that the optimization process not only converges quickly but also remains aligned with the underlying physical system. Experimental results on both synthetic and real-world tasks demonstrate the superior performance of PINN-BO in domains where physical knowledge is crucial.

In summary, this thesis proposed three novel methods, each leveraging DNNs-based surrogate models to enhance computation complexity and sample efficiency in BO as well as its extended settings.

Chapter 1

Introduction

Science has made a great impact on human prosperity. From healthcare to food production to modern communication, scientific breakthroughs have transformed nearly every facet of life. Many of these breakthroughs arise from experiments conducted on complex, unknown systems, with the aim of discovering which inputs yield the most favorable outputs. Some well-known examples are the synthesis of short polymer fiber materials, alloy design, 3D bio-printing, molecule design, etc, (Greenhill et al., 2020; Shahriari et al., 2015). In many cases, experts rely on informed guesses or prior knowledge to decide which inputs to test. This trial-and-error process typically requires numerous experiments to identify the optimal input, a task that can be both time-consuming and expensive. The high costs, in terms of both financial resources and labor, restrict the extent to which these systems can be optimized, thereby slowing progress in research and industry. Therefore, strategies that help find optimal solutions with fewer trials could significantly speed up innovation.

Bayesian Optimization is one of the most effective methods for optimizing expensive, black-box systems (Mockus, Tiesis, and Zilinskas, 1978; Streltsov and Vakili, 1999). It works by using all previous inputs and their corresponding outputs, along with any known information about the system, to build a statistical model of the function. Typically, a probabilistic model such as a GP is trained on available observations. This model helps generate an acquisition function that balances exploration (trying new inputs) and exploitation (refining known good inputs). The acquisition function is optimized to suggest the next input to evaluate, and the resulting output is used to update the model. By iterating this process, Bayesian Optimization can identify optimal inputs with far fewer experiments than traditional methods, leading to significant cost savings in both time and resources.

Mathematically, we can formalize this task as a global optimization problem to optimize $f(\mathbf{x})$ subject to $\mathbf{x} \in \mathcal{D} \subset \mathbb{R}^d$, where d is the number of dimensions, and f is an expensive black-box system that can only be evaluated point-wise. Without any loss in generality, we assume $\mathcal{D}$ to be a d-dimensional space. Further, due to different aspects (e.g., measurement noise), we often only have access to noisy evaluations of f in the form $y = f(\mathbf{x}) + \epsilon$, where ϵ is the noise.

Bayesian Optimization has found widespread use across various disciplines, including materials science, biomedical research, and even other areas of computer science. In materials science, it has contributed to innovations such as the creation of new polymer fibers (Li et al., 2017), the analysis of metal oxide grain boundary structures (Kikuchi et al., 2018), and the enhancement of thermal conductivity in nanostructures (Ju et al., 2017). Within biomedical research, it has been applied to a range of tasks, such as aiding in COVID-19 diagnosis (Nour, Cömert, and Polat, 2020), examining the impact of aging on time perception (Turgeon, Lustig, and Meck, 2016), and designing synthetic genes (Gonzalez et al., 2015). In computer science,

Bayesian optimization is frequently utilized to improve robot control (Berkenkamp, Krause, and Schoellig, 2023) and fine-tune hyperparameters in machine learning models (Snoek, Larochelle, and Adams, 2012; Bergstra and Bengio, 2012).

One of the most fundamental components in Bayesian Optimization is the *surrogate model*. As an approximation of the true objective function, the surrogate model provides both predictions of the function's value and estimates of uncertainty for each potential input. This dual output is crucial for designing the acquisition function, which determines the next point to sample in the optimization process. The acquisition function uses the surrogate model's predictions to balance exploration and exploitation. Exploration encourages sampling in regions with high uncertainty, where the model is less confident, in the hope of discovering better solutions. Exploitation, on the other hand, focuses on regions where the surrogate predicts high performance, aiming to refine promising areas. By leveraging the surrogate model's predictions and uncertainty estimates, the acquisition function can suggest new inputs that strategically advance the search for the optimal solution, often reducing the number of expensive evaluations required.

Due to their flexibility, well-calibrated uncertainty estimates, and favorable analytical properties, GP has long been a popular choice for modeling distributions over functions in Bayesian optimization (Osborne, Garnett, and Roberts, 2009). GP provide a probabilistic framework that captures not only the predictions of the objective function but also the uncertainty associated with those predictions, allowing Bayesian optimization to balance exploration and exploitation effectively. Additionally, the ability of GP to compute closed-form posterior distributions and acquisition functions, such as EI or Upper Confidence Bound (UCB), further enhances their appeal for efficient decision-making in optimization tasks. The combination of Bayesian optimization with GP has led to strong theoretical results, particularly in terms of convergence guarantees. For instance, Srinivas et al. (2009) presented rigorous theoretical bounds showing that Bayesian Optimization with GP models can achieve sublinear regret, which means that the performance of the algorithm improves significantly over time as more data is gathered. Moreover, this framework has been successfully extended to more complex settings, including multi-task and multi-objective optimization (Swersky, Snoek, and Adams, 2013), where GP can jointly model several related tasks or objectives, leveraging shared information to improve optimization efficiency across different tasks.

Despite these advantages, GP comes with inherent limitations that have motivated further research into alternative models. One major challenge is the computational complexity associated with GP, particularly the cubic time complexity for inference, which becomes prohibitive as the dataset grows. This computational burden is especially problematic when handling large-scale problems or high-dimensional input spaces, where GP struggle due to the curse of dimensionality. As the number of dimensions increases, the predictive performance of GP worsens, and the inference becomes slower, limiting their scalability for real-world applications involving large, high-dimensional datasets.

While GP remains the dominant choice for modeling in Bayesian optimization, in response to these challenges, researchers are actively exploring more alternative models to GP, such as Random Forest (RF), Bayesian Neural Network (BNN), and Deep Gaussian Process (Deep GP). These models aim to retain some of the desirable properties of GP, such as uncertainty quantification and flexibility while improving scalability and performance in high-dimensional spaces. For instance, Random Forests (RFs) and DNNs have been employed to estimate black-box functions in this context. RFs tend to perform well at making accurate predictions in the vicinity of

the training data. However, they struggle with extrapolation, meaning their performance can significantly deteriorate when making predictions outside the range of observed data points (Shahriari et al., 2015). Additionally, hybrid approaches that combine GP with other models or approximate inference techniques, such as variational methods (Tran, Ranganath, and Blei, 2016) and sparse approximations (Snelson and Ghahramani, 2007), are being developed to tackle the computational bottlenecks and extend the practical applicability of GP and Bayesian Optimization to more complex and large-scale problems. The first significant application of DNN in Bayesian Optimization was introduced by Snoek et al. (2015), who employed a DNN to transform high-dimensional inputs into lower-dimensional feature vectors. These feature vectors were then fitted to a Bayesian linear regression surrogate model, allowing for more efficient optimization. Another promising approach was presented in Springenberg et al. (2016), where the objective function was modeled directly using a Bayesian Neural Network (BNN). This method aimed to capture the complex relationships within the data while providing uncertainty estimates for the predictions. Despite the empirical success of these neural network approaches in Bayesian optimization tasks, a significant gap remains in the theoretical understanding of their convergence properties. Unlike GPs, which have well-established theoretical frameworks demonstrating their convergence guarantees, the algorithms leveraging neural networks lack similar assurances. This absence of theoretical backing raises questions about the reliability and robustness of these methods in practice, particularly in high-dimensional applications. Moreover, the use of Bayesian linear regression or BNNs introduces substantial computational challenges. Both approaches require maintaining an ensemble of models to derive the posterior distribution over the parameters, leading to increased complexity and potentially prohibitive computational costs. As the dimensionality of the input space or the size of the dataset grows, the demand for computational resources can become a critical limitation. Consequently, while alternative models like random forests and neural networks present exciting avenues for exploration in Bayesian optimization, further research is *necessary to develop* methods that not only demonstrate empirical efficacy but also provide the theoretical guarantees and computational efficiency needed for practical application.

Real-world optimization problems often come with *unknown constraints*. In this setting, black-box optimization with black-box constraints becomes crucial, as both the objective function and constraints need to be optimized simultaneously while ensuring that evaluations are made within feasible regions. Traditional BO methods, which focus solely on optimizing the objective function, fail to capture the interplay between the unknown constraints and the objective, potentially leading to infeasible or suboptimal solutions. Hence, considering only standard BO is insufficient to address such tasks effectively. Many attempts have been made to tackle this challenge by incorporating constraint handling into the BO framework, resulting in methods specifically designed for constrained BO(Gelbart, Snoek, and Adams, 2014; Ariafar et al., 2019; Nguyen et al., 2023). However, despite these advancements, all of these methods still rely on GPs as the surrogate model, which introduces drawbacks similar to those seen in standard GP-based BO, such as poor scalability and increasing computational costs as the number of observations grows. This limitation motivates further *exploration* into more scalable surrogate models, such as DNNs, to better handle the complexities of black-box optimization with black-box constraints.

Additionally, these constraints could represent physical laws, safety conditions,

or other complex requirements that must be satisfied during the optimization process. In many scientific and engineering domains, the objective function and its constraints are governed by well-established physical principles, such as conservation laws, thermodynamics, or fluid dynamics. These principles can often be expressed in the form of PDEs or other mathematical models that describe the underlying system behavior. *Integrating this prior knowledge into the BO framework can significantly improve both the efficiency and accuracy of the optimization process*. By leveraging physical knowledge, we not only reduce the need for expensive function evaluations but also ensure that the optimization process remains within physically feasible regions, thereby avoiding solutions that violate fundamental laws. Following these motivations, we introduce our main aims and approaches

1.1 Aims and Approaches

The main objective of this thesis is to improve the performance and scalability of BO, especially on structural data. More specifically, we focus on these targets:

1. As the number of optimization iterations increases, GPs-based black-box optimization faces computational costs, leading to poor scalability. To address this, we propose replacing the traditional GP surrogate model with a more scalable alternative – DNN.

2. Applying the use of DNN surrogate models into black-box optimization with black-box, expensive constraints.

3. Improving the performance of black-box optimization by integrating useful physical information in the form of PDEs.

To realize these aims, we developed our methods around the idea of using DNN as the surrogate model. Our works can be summarized as:

- To achieve Aim 1, we replace the conventional GP model by a DNN surrogate model. We apply a Thompson Sampling based strategy for choosing the next evaluation. We named this method as Neural-BO.

- To achieve Aim 2, we apply the Expected Improvement (EI) acquisition function to select the next samples within a feasible region, determined by Lower Confidence Bound (LCB) conditions for all constraints. The LCB-based approach guarantees constraint feasibility, while EI efficiently balances exploration and exploitation, especially when the feasible regions are much smaller than the overall search space. We called this method as Neural-CBO.

- To achieve Aim 3, we model the objective function using a Physics-Informed Neural Network (PINN). By incorporating physical knowledge, expressed through PDEs, into the PINN training process, we enhance the sample efficiency of the black-box optimization. We called this method PINN-BO.

1.2 Significant Contributions

The work presented in this thesis is important as it addresses the scalability limitations found in GPs-based Black-box Optimization (BO). As BO is designed for high-cost problems, our approach significantly lowers the computational cost across a wide range of practical tasks. Specifically:

- As GP scale poorly with the number of data points, GPs-based BO have to deal with the increasing computational cost when the number of optimization iterations increases. DNNs-based approaches, such as methods, that rely on BNNs to manage predictive uncertainty, continue to face computationally efficient challenges. Our Neural-BO algorithm, in contrast, incorporated recent advances in neural network theory through the use of NTK to estimate a confidence interval around the neural network model and employ Thompson Sampling technique to determine the next evaluation point. Our theoretical analysis of the algorithm's regret bound demonstrates that it is both convergent and more sample-efficient than existing methods. Furthermore, we show the potential of our Neural-BO algorithm in optimizing complex structured data, such as images and text, while maintaining computational scalability by growing linearly with the number of data points. The experimental results on synthetic benchmarks and real-world optimization tasks highlight that our algorithm outperforms the current state-of-the-art in BO.

- In real-world life, BO tasks often come along with unknown, expensive constraints. Our Neural-CBO has extended the DNN-based idea to this unknown, expensive constraints BO problem. By inheriting the good scalability of DNNs, we modeled both the objective function and constraints using DNNs, and EI is used as the acquisition function. The feasible region is determined using LCB conditions, ensuring constraint satisfaction while balancing exploration and exploitation. Our theoretical analysis shows that cumulative regret and constraint violations have upper bounds comparable to Gaussian Process-based methods. Importantly, the convergence of our model only requires network width to scale linearly with the number of observations. Benchmarking experiments on synthetic and real-world tasks demonstrate that Neural-CBO performs competitively with state-of-the-art techniques.

- Many natural phenomena are governed by physical laws, which can be viewed as constraints when modeling these phenomena as underlying black-box functions. Our contribution focuses on the BO problem, where the black-box function is subject to physical constraints expressed through PDEs. To address this, we introduce the PINN-BO algorithm, which offers several advantages: enhanced sample efficiency in optimization, scalable computation that grows linearly with the number of function evaluations, and the ability to handle a wide variety of PDEs. We demonstrate that PINN-BO outperforms existing methods that lack physical knowledge across both synthetic and real-world problems.

1.3 Structure of this Thesis

- Chapter 2 provides the foundational concepts for our works. We begin by introducing the optimization problem and its variants, from gradient-based approaches to derivative-free strategies under black-box assumptions on the objective functions. Then we describe black-box optimization and its core components, including surrogate models and acquisition functions. The surrogate models relied on key mathematical tools, such as GP and the associated kernels, along with DNN, particularly in over-parameterized settings. We then link these models through the lens of RKHS. Next, we introduce the trade-off between exploration-exploitation through acquisition functions, with a focus

on utility-based acquisition functions (e.g., TS, EI, and UCB). Additionally, we discuss other types of acquisition functions, such as those based on information theory (Entropy Search (ES), Predictive Entropy Search (PES), Max-value Entropy Search (MES)). Finally, we discuss Bandit-based Optimization and establish a connection with Black-box Optimization.

- Chapter 3 describes our DNN-based black-box optimization method named Neural-BO. We begin by discussing the motivations behind the approach and explain how to mathematically integrate a DNN model with TS. Next, we present proof of regret-bound convergence for our proposed method under canonical assumptions. Finally, we provide experimental results on both real-world and synthetic problems, demonstrating that our method outperforms standard approaches, especially in high-dimensional structural data.

- Chapter 4 introduces Neural-CBO method. In this chapter, we utilized the benefit of DNN-based surrogate model into black-box optimization under unknown, black-box constraints. We detail how to use DNN to model, control the uncertainty and balance the exploration-exploitation of both objective function and constraints. Finally, we provide the theoretical guarantee not only for the cumulative regret of the objective function but also for the constraint violation. Similar to Chapter 3, we conduct a wide range of experiments, from synthetic to real-world benchmarks, to demonstrate the effectiveness of our method.

- Chapter 5 focuses on the PINN-BO method, which addresses optimization problems where the objective function is governed by physical knowledge, expressed as physics-based constraints. We begin by outlining the physical information in the form of PDEs and then describe how to incorporate this information into the optimization process. The chapter also provides a theoretical analysis of the convergence of this framework, leveraging the properties of this PINN-BO framework, leveraging the properties of PINN under NTK perspective.

- Chapter 6 concludes this thesis with a summary of our research and a discussion of potential future works.

Chapter 2

Background

This chapter provides a structured exploration of black-box optimization, beginning with the basic mathematical concepts necessary for understanding the subject. Section 2.1 introduces the fundamental mathematical tools and principles that support optimization theory, ensuring a solid foundation for later discussions. Section 2.2 provides a general overview of optimization, distinguishing between local and global approaches and highlighting their respective roles and applications.

In Section 2.3, we shift our focus to black-box optimization, examining the challenges posed by functions that are expensive to evaluate or lack explicit mathematical representation. Section 2.4 extends this discussion by addressing problems with unknown constraints, emphasizing strategies to effectively explore feasible regions in such complex scenarios. Section 2.5 explores how incorporating physics-based knowledge into optimization frameworks can improve both solution accuracy and computational efficiency. Finally, in Section 2.6, we conclude the chapter by outlining potential future research directions, focusing on unresolved challenges and emerging opportunities in black-box optimization.

2.1 Essential Mathematics Backgrounds

2.1.1 Gaussian and Sub-Gaussian Random Variables

2.1.1.1 Gaussian Random Variables

Gaussian random variables, also known as normal random variables, are fundamental in probability theory and statistics. A Gaussian random variable X is fully characterized by its mean μ and variance σ^2, and its probability density function (PDF) is given by:

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad x \in \mathbb{R}.$$

The tails of a Gaussian distribution decay exponentially, meaning the probability of large deviations from the mean diminishes rapidly. Specifically, for $X \sim \mathcal{N}(\mu, \sigma^2)$, the tail probability can be bounded as:

$$\mathbb{P}(|X-\mu| > t) \leq 2\exp\left(-\frac{t^2}{2\sigma^2}\right), \quad t > 0.$$

This exponential decay of tail probabilities is a key property that makes Gaussian distributions central to statistical theory and concentration inequalities. However, many random variables in practice exhibit similar tail behaviors without necessarily

following the exact Gaussian distribution. This motivates the broader class of *sub-Gaussian random variables*.

2.1.1.2 Sub-Gaussian Random Variables

A random variable X is sub-Gaussian if its tail probabilities decay at least as fast as those of a Gaussian random variable. Intuitively, sub-Gaussian random variables are those whose deviations from their mean are tightly controlled, similar to or better than the Gaussian case.

Formally, X is sub-Gaussian if there exists a constant $\sigma > 0$ such that for all $t > 0$:

$$\mathbb{P}(|X - \mathbb{E}[X]| > t) \leq 2\exp\left(-\frac{t^2}{2\sigma^2}\right).$$

Here, σ^2 is called the *sub-Gaussian variance parameter*, and it provides a measure of the "spread" of the random variable, analogous to the variance in a Gaussian distribution.

An equivalent definition involves the moment generating function (MGF). A random variable X is sub-Gaussian if there exists a constant $\sigma > 0$ such that:

$$\mathbb{E}[\exp(\lambda(X - \mathbb{E}[X]))] \leq \exp\left(\frac{\lambda^2\sigma^2}{2}\right), \quad \forall\lambda \in \mathbb{R}.$$

Many common random variables exhibit sub-Gaussian behavior, including Gaussian, bounded, and discrete random variables. Below are some illustrative examples:

- **Gaussian Random Variables:** Any Gaussian random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ is inherently sub-Gaussian, with σ^2 as its variance.
- **Bounded Random Variables:** If X is a bounded random variable, i.e., $|X| \leq B$ almost surely, then X is sub-Gaussian with $\sigma \leq B$.
- **Rademacher Random Variables:** A Rademacher random variable, which takes values ± 1 with equal probability, is sub-Gaussian with $\sigma = 1$.

Sub-Gaussian random variables exhibit several important properties that make them useful in probabilistic analyses, particularly when handling extreme deviations and ensuring concentration of measure. These properties include:

- **Tight Tail Bounds:** Sub-Gaussian random variables satisfy exponential tail bounds, making them useful in scenarios where extreme deviations must be controlled.
- **Closed under Affine Transformations:** If X is sub-Gaussian, then $aX + b$ is also sub-Gaussian, with the sub-Gaussian parameter scaled by $|a|$.
- **Sum of Independent Sub-Gaussians:** If $X_1, X_2, \ldots, X_n$ are independent sub-Gaussian random variables, their sum $S_n = \sum_{i=1}^{n} X_i$ is also sub-Gaussian, with the variance parameter satisfying $\sigma_{S_n}^2 = \sum_{i=1}^{n} \sigma_i^2$.

While sub-Gaussian random variables provide valuable tools for modeling uncertainties in single values, many real-world problems cannot be modeled with single values, as these problems often deal with the uncertainty over entire functions. Therefore, it becomes necessary to capture the dependencies between the values of

the function at different locations or times, which implies a need for working with joint distributions. The key challenge arises when dealing with functions, which can take values at infinitely many points, thus requiring a notion of probability distributions over functions. In the next section, we introduce Gaussian Process (GP), which is the natural extension of the normal distribution from the case of individual random variables to function spaces.

2.1.2 Gaussian Process

A Gaussian Process (GP) is a finite collection of normally distributed random variables with each having a multivariate normal distribution. GP offers a prior distribution over smooth functions and are completely specified by a mean function, $\mu(\mathbf{x})$ and covariance function, $k(\mathbf{x}, \mathbf{x}')$. We have

$$f(\mathbf{x}) \sim \mathrm{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{2.1}$$

where the mean and variance of a normal distribution is returned as the value of $f(\mathbf{x})$ for an arbitrary $\mathbf{x}$. Without loss in generality, $f(\mathbf{x})$ values can be assumed as a realization of random variables with prior mean as zero thus making the Gaussian process fully specified by the covariance matrix (Rasmussen, Williams, et al., 2006). There are many popular covariance functions including Matérn kernel, Linear kernel, Squared Exponential kernel. These kernels will be briefly introduced in the next section.

We assume that function values $\mathbf{y}_{1:t} = f(\mathbf{x}_{1:t}) + \epsilon_{1:t}$ corresponding to the initial points $\mathbf{x}_{1:t}$ are sampled from the prior Gaussian process. Collectively we denote the observations as $\mathcal{D}_{1:t} = \{\mathbf{x}_{1:t}, \mathbf{y}_{1:t}\}$. The function values $f(\mathbf{x}_{1:t})$ follow a multivariate Gaussian distribution $\mathcal{N}(0, \mathbf{K}_t)$, where

$$\mathbf{K}_t = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_t, \mathbf{x}_1) & \dots & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix}$$

Given a new point $\mathbf{x}_{t+1}$, then $\mathbf{x}_{t+1}, y_{1:t}$ and $f(\mathbf{x}_{t+1})$ are jointly Gaussian. By the properties of Gaussian process we can write

$$\begin{pmatrix} \mathbf{y}_{1:t} \\ f(\mathbf{x}_{t+1}) \end{pmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_t & \mathbf{k}_{1:t} \\ \mathbf{k}_{1:t}^\top & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix}\right),$$

where $\mathbf{k}_{1:t} = \begin{bmatrix} k(\mathbf{x}_{t+1}, \mathbf{x}_1) & k(\mathbf{x}_{t+1}, \mathbf{x}_2) & \dots & k(\mathbf{x}_{t+1}, \mathbf{x}_t) \end{bmatrix}$.

Then using Sherman-Morrison-Woodbury formula in Rasmussen, Williams, et al. (2006), the predictive distribution is:

$$\mathbb{P}(y_{t+1} | \mathcal{D}_{1:t}, \mathbf{x}_{t+1}) = \mathcal{N}(\mu_t(\mathbf{x}_{t+1}), \sigma_t^2(\mathbf{x}_{t+1})),$$

where the predictive mean is $\mu_t(\mathbf{x}_{t+1}) = \mathbf{k}_{1:t}^\top \mathbf{K}_t^{-1} \mathbf{y}_{1:t}$ and variance $\sigma_t^2(\mathbf{x}_{t+1}) = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}_{1:t}^\top \mathbf{K}_t^{-1} \mathbf{k}_{1:t}$.

2.1.3 Kernels

In the context of Gaussian Processes (GPs), kernels (also known as covariance functions) play a crucial role in defining the prior belief about the function being modeled. A kernel, denoted as $k(\mathbf{x}, \mathbf{x}')$, quantifies the similarity or correlation between function values at two input locations, $\mathbf{x}$ and $\mathbf{x}'$. It is symmetric, positive semi-definite, and it defines the properties of the functions sampled from the GP prior. These properties ensure that the resulting covariance matrix constructed using this kernel will be valid (positive semi-definite), which is necessary for the Gaussian Process to be well-defined.

The choice of kernel is a critical aspect of GP modeling, as it directly impacts the smoothness, complexity, and overall behavior of the functions that the model is capable of representing. Different kernels encode different assumptions about the underlying function we are attempting to model, and it is crucial to carefully consider these assumptions when choosing a kernel for a specific task. From the perspective of Bayesian inference, the kernel can be seen as the main source of prior information about the properties of the functions we are modeling. Therefore, it is essential to choose a kernel that is appropriate for the structure and characteristics that one might expect from the function. We now consider some common types of kernels, each with its own assumptions and characteristics:

Linear Kernel: The linear kernel is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

where $\mathbf{x}$ and $\mathbf{x}'$ are vectors in the input space. This kernel models the assumption that the function values are linearly related to the inputs. The resulting functions are linear combinations of the input variables. This kernel is useful when there is a reason to assume that the underlying data has a linear structure, or when working with high-dimensional data where the kernel has less sensitivity to changes in the input. However, this kernel is often too restrictive for complex functions due to its lack of flexibility and can't capture non-linear patterns.

Squared Exponential (RBF) Kernel The squared exponential kernel, also known as the Radial Basis Function (RBF) kernel, is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right),$$

where ℓ is the length-scale parameter. The RBF kernel assumes that the function values are highly correlated when the inputs are close to each other. The parameter ℓ controls the "reach" of the correlation; a larger ℓ implies that the function values are correlated even when the inputs are farther apart. This kernel is associated with infinitely smooth functions, which implies that the functions represented with this kernel do not have sharp changes or rapid variations. The RBF kernel is widely used for its flexibility and its ability to model non-linear functions, and also because its parameter controls the range of smoothness of the represented functions. However, its isotropic nature might make it not ideal in scenarios where the different dimensions have different degrees of correlation.

Matérn Kernel The Matérn kernel is a more general kernel and is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|\mathbf{x} - \mathbf{x}'\|}{\ell} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}\|\mathbf{x} - \mathbf{x}'\|}{\ell} \right)$$

where $\nu > 0$ is the smoothness parameter, ℓ is the length-scale parameter, and K_{ν} is the modified Bessel function of the second kind. The Matérn kernel is a generalization of the RBF kernel, which is obtained in the limit of large ν. This kernel allows for controlling the smoothness of the underlying function, providing a continuous range of possibilities. The smoothness of functions associated with the Matérn kernel is determined by the parameter ν. A low ν implies less smooth functions, whereas, as ν increases, the functions become increasingly smoother. This parameter also has an impact on the convergence of Bayesian Optimization algorithms. Matérn kernels with different smoothness parameters have also been found to have a crucial effect in the generalization capacity of Gaussian Process models, as it reflects the trade-off between modeling data and having a smooth prior (Rasmussen, Williams, et al., 2006). The Matérn kernel allows the flexibility of the RBF kernel, but includes an additional parameter that enables one to control the smoothness of the modeled function. This parameter is not present in the case of RBF kernels.

The choice of the kernel is not merely a matter of convenience; it directly influences the properties of the function space over which the Gaussian process operates. Selecting an appropriate kernel demands a careful understanding of the underlying assumptions of each kernel type and considering the characteristics of the function to be modeled. Kernels enable us to encode our prior beliefs about the function, and hence, proper selection is crucial for the efficiency and generalization capability of our Gaussian Process model. We have presented three common kernels that are often used in practice, and their differences highlight the importance of selecting the appropriate kernel for the task. The theoretical properties of these kernels can be found in the literature. Choosing between them requires an understanding of the properties of the data and the functions to be modeled.

2.1.4 Maximum Information Gain

In the context of machine learning and information theory, Maximum Information Gain (MIG) is a principle used to select features, queries, or actions that yield the highest amount of information about an unknown variable of interest. The concept of information gain is rooted in Shannon's entropy theory and measures the reduction in uncertainty about a random variable after observing another variable.

2.1.4.1 Entropy

Entropy is a measure of the uncertainty or unpredictability in a random variable. For a discrete random variable X with possible outcomes $\mathbf{x} \in \mathcal{X}$ and probability distribution $p(X)$, the entropy $H(X)$ is defined as:

$$H(X) = -\sum_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}) \log p(\mathbf{x}).$$

This entropy value represents the average number of bits required to encode information about the variable X. When we consider a second random variable Y, we can examine how much observing Y reduces uncertainty about X, which is the basis for defining information gain.

The *conditional entropy* of X given Y, denoted $H(X|Y)$, measures the remaining uncertainty about X after observing Y. It is defined as:

$$H(X|Y) = -\sum_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}) \sum_{\mathbf{x}\in\mathcal{X}} p(\mathbf{x}|\mathbf{y}) \log p(\mathbf{x}|\mathbf{y}).$$

The conditional entropy quantifies the uncertainty in X given that we know Y. If observing Y provides information about X, we expect $H(X|Y) < H(X)$.

2.1.4.2 Mutual Information

The *mutual information* $IG(X;Y)$ between two random variables X and Y is defined as the reduction in entropy of X after observing Y, and is given by:

$$IG(X;Y) = H(X) - H(X|Y).$$

This quantity represents the average reduction in uncertainty about X provided by knowledge of Y. Alternatively, it can also be expressed in terms of the mutual information $I(X;Y)$ between X and Y:

$$IG(X;Y) = I(X;Y),$$

where mutual information is defined as:

$$I(X;Y) = \sum_{\mathbf{x}\in\mathcal{X}} \sum_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{x},\mathbf{y}) \log \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{x})p(\mathbf{y})}.$$

Mutual information is symmetric and non-negative, i.e., $I(X;Y) = I(Y;X) \geq 0$, and measures the amount of shared information between X and Y.

While the concept of Mutual Information generally quantifies the reduction in uncertainty about one random variable when observing another, its application becomes specialized within the context of GPs. Specifically, when using a GP to learn an unknown function f, the information gain is redefined as the mutual information between the noisy observations (the function values at chosen points) and the true, underlying function values f. This specialization allows us to analyze the effectiveness of our GP learning process based on the information provided by our observed data.

2.1.4.3 Maximum Information Gain in GP regression

Assume we have a GP prior over f with mean function $\mu(\mathbf{x})$ and covariance $k(\mathbf{x},\mathbf{x}')$. After t steps, the model receives an input sequence $\mathcal{X}_t = (\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_t)$ and observes noisy rewards $\mathbf{y}_t = (y_1, y_2, \ldots, y_t)$. The *information gain* at step t, quantifies the reduction in uncertainty about f after observing $\mathbf{y}_t$, defined as the mutual information between $\mathbf{y}_t$ and f:

$$I(\mathbf{y}_t; f) := H(\mathbf{y}_t) - H(\mathbf{y}_t|f),$$

where H denotes the entropy. To obtain the closed-form expression of information gain, one needs to introduce a GP model where f is assumed to be a zero mean GP indexed on $\mathcal{X}$ with kernel k. Let $\mathbf{f}_t = [(f(\mathbf{x}_1), f(\mathbf{x}_1), \ldots, f(\mathbf{x}_t)])$ be the corresponding true function values. From Cover (1999), the mutual information between two

multivariate Gaussian random variables is:

$$I(\mathbf{y}_t; f) = I(\mathbf{y}_t; \mathbf{f}_t) = \frac{1}{2} \log \det\Big(\mathbf{I}_t + \lambda^{-1}\mathbf{K}_t\Big),$$

where $\lambda > 0$ is a regularization parameter, $\mathbf{K}_t$ is the covariance kernel matrix of the points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, and $\mathbf{I}_t$ is the identity matrix.

The Information Gain obtained is highly dependent on the specific locations where observations are made. To address this variability, the Maximum Information Gain (MIG) after observing t data points, represented as γ_t, is introduced as an upper bound. Instead of focusing on a particular point selection, MIG quantifies the maximum possible information that could be gained from any set of t points in the input space. MIG provides an input-independent and kernel-specific bound on the information that can be acquired:

$$\gamma_t := \max_{\mathcal{A} \subset \mathcal{X}, |\mathcal{A}| = t} I(\mathbf{y}_A; \mathbf{f}_A),$$

As mentioned above, $\mathcal{A}$ represents a set of t input points in the input space $\mathcal{X}$, $\mathbf{y}_\mathcal{A}$ denotes the noisy observations of the objective function f at these points, and $\mathbf{f}_\mathcal{A}$ represents the corresponding function values *without* noise. $I(\mathbf{y}_\mathcal{A}; \mathbf{f}_\mathcal{A})$ is the mutual information between the observed data $\mathbf{y}_\mathcal{A}$ and the underlying function values $\mathbf{f}_\mathcal{A}$. Basically, the MIG quantifies the maximum amount of information that *any* set of t points could possibly reveal about the unknown objective function f. Specifically, the MIG serves as an upper bound on the information obtained by any selection of t points. This means that even in the worst-case scenario (i.e., the scenario where we choose the least informative set of points within the optimization algorithm), the amount of information gained will never surpass the MIG. The work by Srinivas et al. (2009) provides crucial kernel-specific bounds on the MIG. Specifically, they analyzed three common kernels: Linear, RBF, and Matérn introduced in Section 2.1.3. The bounds on MIG for these kernels are:

- **Linear Kernel:** The MIG is bounded by

  $$\gamma_t = \mathcal{O}(d \log t),$$

  where d is the input dimension.

- **Squared Exponential (RBF) Kernel:** The MIG for RBF kernel is bounded by:

  $$\gamma_t = \mathcal{O}((\log t)^{d+1}).$$

- **Matérn Kernel:** For a specific smoothness parameter ν of Matérn kernel, the MIG has a bound:

  $$\gamma_t = \mathcal{O}\left(t^{\frac{d(d+1)}{2\nu + d(d+1)}} \log t\right).$$

2.1.5 Reproducing Kernel Hilbert Space (RKHS)

RKHSs provide a rigorous framework for dealing with functions in high-dimensional spaces using kernel methods, which are central in machine learning, functional analysis, and statistics. Formally, RKHSs are Hilbert spaces associated with a positive definite kernel, allowing efficient manipulation of functions via inner products and enabling regularization in infinite-dimensional settings.

2.1.5.1 Hilbert Spaces and Inner Products

A *Hilbert space* $\mathcal{H}$ is a complete vector space equipped with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ such that every Cauchy sequence in $\mathcal{H}$ converges in $\mathcal{H}$. Completeness is crucial for ensuring that limits of function sequences (e.g., solutions to optimization problems) lie within the space.

For an inner product space $\mathcal{H}$, the norm induced by the inner product is given by:

$$\|f\|_{\mathcal{H}} = \sqrt{\langle f, f \rangle_{\mathcal{H}}}.$$

2.1.5.2 Kernel Functions and Positive Definiteness

Let $\mathcal{X}$ be an input space, typically $\mathbb{R}^d$, and let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a *kernel function*. A function k is called *positive definite* if for any finite set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathcal{X}$ and any $\mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{R}^n$, we have:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0.$$

This property ensures that k can serve as a similarity measure and enables the construction of a unique RKHS associated with k.

2.1.5.3 Constructing RKHS: The Moore-Aronszajn Theorem

A central result in RKHS theory is the *Moore-Aronszajn Theorem*, which guarantees the existence of a unique RKHS for any positive definite kernel k. The theorem states:

Theorem 2.1.1 (Moore-Aronszajn)**.** *For any positive definite kernel k on $\mathcal{X} \times \mathcal{X}$, there exists a unique Hilbert space $\mathcal{H}$ of functions $f : \mathcal{X} \to \mathbb{R}$ such that:*

1. *$k(\mathbf{x}, \cdot) \in \mathcal{H}$ for all $\mathbf{x} \in \mathcal{X}$,*
2. *For every $f \in \mathcal{H}$ and $\mathbf{x} \in \mathcal{X}$, $f(\mathbf{x}) = \langle f, k(\mathbf{x}, \cdot) \rangle_{\mathcal{H}}$, called the reproducing property.*

This theorem allows us to construct $\mathcal{H}$ as the *completion* of the span of the functions $\{k(\mathbf{x}, \cdot) : \mathbf{x} \in \mathcal{X}\}$ with respect to the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$.

2.1.5.4 Inner Product and Norm in RKHS

Given two functions $f, g \in \mathcal{H}$ represented as finite linear combinations of kernel functions, i.e., $f = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot)$ and $g = \sum_{j=1}^{m} \beta_j k(\mathbf{y}_j, \cdot)$, the inner product in $\mathcal{H}$ can be computed as:

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{y}_j).$$

The corresponding norm is:

$$\|f\|_{\mathcal{H}} = \sqrt{\langle f, f \rangle_{\mathcal{H}}} = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)}.$$

This norm represents the "smoothness" or "complexity" of functions in $\mathcal{H}$, with higher values corresponding to more complex functions.

2.1.5.5 Reproducing Property and Point-wise Evaluation

A crucial property in RKHS is the *reproducing property*, which allows pointwise evaluation of functions $f \in \mathcal{H}$ using the inner product. For any $f \in \mathcal{H}$ and $\mathbf{x} \in \mathcal{X}$,

$$f(\mathbf{x}) = \langle f, k(\mathbf{x}, \cdot) \rangle_{\mathcal{H}}.$$

This property simplifies function evaluations to inner products, making it possible to evaluate functions at any point without explicitly knowing the form of f.

2.1.5.6 Representer Theorem

In many machine learning applications, we wish to minimize a regularized empirical risk functional of the form:

$$\min_{f \in \mathcal{H}} \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + \lambda \|f\|_{\mathcal{H}}^2,$$

where L is a loss function (e.g., squared loss for regression) and $\lambda > 0$ is a regularization parameter. The *Representer Theorem* asserts that the solution to this optimization problem can be represented as a linear combination of kernel evaluations at the training points:

$$f^*(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}),$$

where the coefficients α_i can be found by solving a finite-dimensional problem. This reduces the complexity of optimization from the infinite-dimensional space $\mathcal{H}$ to a finite-dimensional space of size n, greatly simplifying computation.

2.1.5.7 Regularization and Smoothness in RKHS

The RKHS norm $\|f\|_{\mathcal{H}}$ provides a measure of the smoothness of f. When we regularize by minimizing $\|f\|_{\mathcal{H}}^2$, we control the complexity of the function, effectively penalizing highly oscillatory or rough functions. This form of regularization is particularly relevant in *kernel ridge regression* and *support vector machines*, where the objective function is minimized subject to a smoothness constraint in $\mathcal{H}$.

2.1.5.8 Examples of Kernels and Corresponding RKHS

- **Linear Kernel** $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$: The RKHS corresponding to this kernel is the space of linear functions $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.
- **Polynomial Kernel** $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + c)^p$: This kernel represents polynomial functions of degree p, enabling polynomial feature spaces.
- **Gaussian (RBF) Kernel** $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)$: The RKHS for this kernel includes all smooth functions, making it particularly effective for non-linear, smooth function approximation.

2.2 Introduction to Optimization

Optimization is a fundamental mathematical process focused on identifying the best possible solution from a range of options for a given problem. It is a core technique

used in diverse fields, including engineering, economics, machine learning, and operations research. To address this broad spectrum of problems, a consistent way to represent them is necessary. Optimization problems are typically expressed using an objective function $f\colon \mathbb{R}^d \to \mathbb{R}$, which transforms an input vector $\mathbf{x}$ into a scalar output y. The goal is to determine the input $\mathbf{x}^*$ that results in the optimal output. Although some problems involve minimizing f, this can be equivalently framed as maximizing $-f$, allowing a uniform representation of all optimization problems as:

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

where $\mathcal{X}$ represents the set of feasible inputs. In certain cases, f might have a well-defined mathematical form, enabling direct algebraic solution. However, in practical, real-world scenarios, the function is often too complex. In such cases, the function's value must be estimated by evaluating it at various inputs. The key challenge is how to effectively select which inputs to evaluate. Randomly selecting, although straightforward, does not guarantee convergence to an optimal solution within a finite number of evaluations. Therefore, specialized algorithms are required to guide the selection of inputs, ensuring both efficient and effective optimization.

2.2.1 Gradient-Based Optimization

Gradient-based methods use the gradient (the vector of first-order partial derivatives) of the objective function to iteratively find optimal solutions. The gradient at any point indicates the direction of the steepest ascent of the objective function, so the negative gradient is used for minimization.

The core idea behind gradient-based optimization is to update the decision variable $\mathbf{w}$ iteratively using the following update rule:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k d_k, \tag{2.2}$$

where:

- $\mathbf{x}_k$ is the decision variable at iteration k.
- $\mathbf{x}_{k+1}$ is the updated decision variable for the next iteration.
- α_k is the learning rate or step size, a positive scalar.
- d_k is the direction vector which can be specific for each optimization algorithm.

We will now provide a detailed examination of the mathematical foundations of various gradient-based optimization methods, where the objective function is denoted as $f(\mathbf{x})$.

2.2.1.1 Gradient Descent

In its simplest form, Gradient Descent (GD) updates the solution by moving in the direction of the negative gradient. The negative gradient points towards the direction of the steepest descent on the error surface. This update pushes the solution in the direction that decreases the objective function the most.

$$d_k = -\nabla f(\mathbf{x}_k) \tag{2.3}$$

The update rule of Eqn. 2.2 thus becomes:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k), \tag{2.4}$$

where $\nabla f(\mathbf{x}_k) = \left(\frac{\partial f}{\partial \mathbf{x}_1}, \frac{\partial f}{\partial \mathbf{x}_2}, \ldots, \frac{\partial f}{\partial \mathbf{x}_d} \right)$ is the gradient vector of f at x_k.

2.2.1.2 Momentum

Building on the foundational principles of gradient-based optimization, we turn our attention to momentum-based variants of GD. These methods aim to improve the convergence rate and stability of the optimization process by incorporating a momentum term. Momentum introduces a velocity term that accumulates gradients, which helps to smooth the optimization trajectory and escape shallow local minima.

The update rule involves an intermediate variable called velocity $\mathbf{v}_k$.

$$\mathbf{v}_k = \beta \mathbf{v}_{k-1} - \alpha_k \nabla f(\mathbf{x}_k), \tag{2.5}$$

where β is the momentum parameter, typically a scalar in the range $[0, 1)$, controlling the influence of past gradients.Then the update of $\mathbf{x}_k$ becomes:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_k \tag{2.6}$$

The momentum term βv_{k-1} effectively remembers previous gradients, adding them to the current gradient, which acts as a smoothing filter. This helps the solution continue in a similar direction even if it encounters small hills or valleys on the error surface. Momentum-based methods, such as classical Momentum and Nesterov Accelerated Gradient (NAG), are widely used for their ability to overcome challenges like slow convergence on flat surfaces and oscillations in steep directions. These methods are particularly beneficial for optimizing non-convex functions, where standard GD often struggles.

2.2.1.3 Adaptive Learning Rate Algorithms

In addition to momentum-based methods, adaptive learning rate algorithms have emerged as powerful tools in optimization. These algorithms adjust the learning rate dynamically during the optimization process, allowing for more efficient convergence. By adapting the step size based on the magnitude of gradients or past updates, they aim to overcome challenges such as vanishing or exploding gradients and improve performance across diverse problem settings.

Adagrad One of the simplest forms of adaptive learning rate algorithms is Adagrad, which modifies the learning rate for each parameter based on the accumulated squared gradients:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\alpha}{\sqrt{G_k + \epsilon}} \odot \nabla f(\mathbf{x}_k),$$

where:

- $G_k = \sum_{i=1}^{k} \nabla f(\mathbf{x}_i) \odot \nabla f(\mathbf{x}_i)$ is the element-wise sum of squared gradients up to iteration k,
- ϵ is a small positive constant to avoid division by zero,
- $\odot$ denotes element-wise multiplication,

- α is the initial learning rate.

Adam Another popular algorithm, Adaptive Moment Estimation (Adam) (Adaptive Moment Estimation), introduced by Kingma and Ba (2014), builds upon Adagrad to address its limitations, particularly its tendency to excessively decay the learning rate in the presence of accumulated gradients. Adam combines momentum with adaptive learning rates and maintains both a first-moment estimate (momentum) and a second-moment estimate (variance) for more robust optimization. This dual approach helps stabilize parameter updates and adaptively scales learning rates for each parameter:

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1 - \beta_1) \nabla f(\mathbf{x}_k), \tag{2.7}$$

$$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1 - \beta_2) (\nabla f(\mathbf{x}_k))^2, \tag{2.8}$$

where $\mathbf{m}_k$ is the estimate for the first-order moment (mean of the gradients), and $\mathbf{v}_k$ is the estimate for the second-order moment (uncentered variance of the gradients). The hyperparameters $\beta_1 \in [0, 1)$ and $\beta_2 \in [0, 1)$ control the exponential decay rates of the moving averages for the first and second moments, respectively. Common default values are $\beta_1 = 0.9$ and $\beta_2 = 0.999$, as recommended in the original paper.

To address bias introduced by initializing $\mathbf{m}_k$ and $\mathbf{v}_k$ to zero at the beginning of training, corrected estimates are computed as:

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1 - \beta_1^k}, \tag{2.9}$$

$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1 - \beta_2^k}, \tag{2.10}$$

These bias-corrected estimates ensure that $\hat{\mathbf{m}}_k$ and $\hat{\mathbf{v}}_k$ are unbiased, particularly during the initial steps when k is small. The final parameter update step is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \frac{\hat{\mathbf{m}}_k}{\sqrt{\hat{\mathbf{v}}_k} + \epsilon}, \tag{2.11}$$

where α is the learning rate, typically set to a small value (e.g., 0.001 in many applications), and ϵ is a small constant added to prevent division by zero. A typical value for ϵ is 10^{-8}. The inclusion of ϵ improves numerical stability, particularly in cases where $\hat{\mathbf{v}}_k$ approaches zero.

RMSProp Root Mean Square Propagation (RMSProp), proposed by Tieleman (2012), is another popular optimization algorithm designed to overcome the drawbacks of Adagrad in non-convex optimization problems. While Adagrad's aggressive learning rate decay can hinder optimization over time, RMSProp mitigates this by using an exponentially decaying average of squared gradients to adapt learning rates. This mechanism ensures a more stable and efficient convergence, particularly in neural network training. The update rule for RMSProp is given by:

$$\mathbf{v}_k = \beta \mathbf{v}_{k-1} + (1 - \beta) (\nabla f(\mathbf{x}_k))^2, \tag{2.12}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \frac{\nabla f(\mathbf{x}_k)}{\sqrt{\mathbf{v}_k} + \epsilon}, \tag{2.13}$$

where $\mathbf{v}_k$ represents the moving average of the squared gradients, and $\beta \in [0, 1)$ is a hyperparameter controlling the decay rate of this average, typically set to 0.9. The term ϵ is a small positive constant, commonly 10^{-8}, added for numerical stability to avoid division by zero. The adaptive scaling of gradients via $\sqrt{\mathbf{v}_k}$ allows RMSProp to handle large gradients effectively while maintaining a consistent learning rate for smaller gradients. This balance makes RMSProp particularly well-suited for deep learning applications involving non-stationary loss functions.

2.2.1.4 Conjugate Gradient

The Conjugate Gradient (CG) method is a powerful optimization algorithm primarily used for solving large-scale, unconstrained quadratic minimization problems, particularly when the Hessian matrix is too large to store explicitly. Introduced by Hestenes, Stiefel, et al. (1952), CG iteratively updates the solution by combining gradient descent with a conjugacy condition, ensuring that successive search directions are mutually conjugate with respect to the quadratic form. This approach enables faster convergence compared to standard gradient descent. At each iteration, the solution is updated as:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{2.14}$$

$$\mathbf{p}_{k+1} = -\nabla f(\mathbf{x}_{k+1}) + \beta_k \mathbf{p}_k, \tag{2.15}$$

where $\mathbf{p}_k$ is the search direction, α_k is the step size determined by line search, and β_k is the conjugate gradient coefficient. The value of β_k can be computed using various formulas, including Fletcher-Reeves and Polak-Ribiere. The Fletcher-Reeves formula defines β_k as:

$$\beta_k^{\text{FR}} = \frac{\|\nabla f(\mathbf{x}_{k+1})\|^2}{\|\nabla f(\mathbf{x}_k)\|^2}, \tag{2.16}$$

while the Polak-Ribiere formula introduces a more flexible approach:

$$\beta_k^{\text{PR}} = \frac{\nabla f(\mathbf{x}_{k+1})^\top (\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\|^2}. \tag{2.17}$$

The Polak-Ribiere method can handle non-quadratic objective functions more effectively and may reset β_k to zero when its value becomes negative, reverting to steepest descent for better stability. These formulations ensure that successive directions remain conjugate, allowing CG to converge in at most n iterations for a quadratic objective with n variables. The CG method is particularly efficient for sparse systems and avoids the need to compute or store the full Hessian matrix, making it well-suited for high-dimensional problems. Extensions of CG, such as the nonlinear Conjugate Gradient method, adapt these principles to more general optimization scenarios.

2.2.1.5 Newton's Method

Newton's Method is a second-order optimization algorithm widely used for solving unconstrained optimization problems. It leverages both gradient and curvature information to achieve faster convergence, especially near the optimum. At each

iteration, Newton's Method approximates the objective function $f(\mathbf{x})$ as a second-order Taylor expansion around the current point $\mathbf{x}_k$ and updates the parameters by solving the resulting quadratic model. The update rule is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k^{-1} \nabla f(\mathbf{x}_k), \tag{2.18}$$

where $\nabla f(\mathbf{x}_k)$ is the gradient vector, and $\mathbf{H}_k$ is the Hessian matrix, representing the second-order partial derivatives of $f(\mathbf{x})$ with respect to the parameters. The inverse Hessian $\mathbf{H}_k^{-1}$ scales and modifies the gradient direction to account for the curvature of the objective function, allowing for more precise steps toward the optimum.

One of the key advantages of Newton's Method is its quadratic convergence rate when the initial guess is sufficiently close to the solution. However, the method also has notable challenges. Computing and inverting the Hessian matrix can be computationally expensive for high-dimensional problems, making it less practical for large-scale optimization. To address this, approximations like the quasi-Newton methods (e.g., BFGS) are often used to estimate the inverse Hessian iteratively without explicitly forming it.

Newton's Method is particularly effective for convex optimization problems where the Hessian is positive definite. In non-convex settings, the method may encounter difficulties such as saddle points or negative curvature, which can lead to divergence. To mitigate this, techniques such as adding a regularization term to the Hessian (e.g., trust-region methods) or using a modified update rule are commonly employed.

2.2.1.6 Quasi-Newton Methods (BFGS, L-BFGS)

Quasi-Newton methods are iterative optimization algorithms that approximate Newton's Method while avoiding the explicit computation and inversion of the Hessian matrix, making them more computationally efficient for high-dimensional problems. Among these, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is one of the most widely used. At each iteration, the search direction $\mathbf{p}_k$ is computed as:

$$\mathbf{p}_k = -\mathbf{H}_k^{-1} \nabla f(\mathbf{x}_k), \tag{2.19}$$

where $\mathbf{H}_k^{-1}$ is the approximation of the inverse Hessian matrix. The parameter update is then performed as:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \tag{2.20}$$

with α_k representing the step size, typically determined through a line search. In the BFGS algorithm, the approximation of the Hessian matrix $\mathbf{H}_k$ itself is updated iteratively using the formula:

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{y}_k \mathbf{y}_k^\top}{\mathbf{y}_k^\top \mathbf{s}_k} - \frac{\mathbf{H}_k \mathbf{s}_k \mathbf{s}_k^\top \mathbf{H}_k}{\mathbf{s}_k^\top \mathbf{H}_k \mathbf{s}_k}, \tag{2.21}$$

where $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$. The first term ensures the positive definiteness of $\mathbf{H}_{k+1}$, while the second term adjusts the curvature information. To maintain computational feasibility, $\mathbf{H}_k$ is often initialized as a scaled identity matrix, $\mathbf{H}_0 = \gamma \mathbf{I}$, where γ is a positive scalar.

For large-scale problems where storing or computing $\mathbf{H}_k$ or $\mathbf{H}_k^{-1}$ explicitly is impractical, the Limited-memory BFGS (L-BFGS) algorithm is employed. Instead of

maintaining the full Hessian or its inverse, L-BFGS uses a limited number of recent $\mathbf{s}_k$ and $\mathbf{y}_k$ vectors to implicitly compute the search direction $\mathbf{p}_k$. This reduces memory and computational overhead, making L-BFGS well-suited for problems with millions of variables.

Quasi-Newton methods such as BFGS and L-BFGS offer superlinear convergence rates and efficiently balance the rapid convergence of Newton's Method with the simplicity of gradient descent. Their robustness and efficiency make them standard tools for large-scale optimization tasks, particularly in machine learning and numerical optimization.

2.2.2 Derivative-Free Optimization

The optimization techniques discussed so far, including Gradient Descent, Newton's Method, and their variants, rely fundamentally on gradient information to navigate the search space and converge to an optimum. These methods are well-suited for problems where derivatives are readily available, accurate, and computationally inexpensive. However, many real-world optimization problems present challenges such as noisy evaluations, non-differentiable objective functions, or black-box functions where derivatives are unavailable or impractical to compute.

In such scenarios, Derivative-Free Optimization (DFO) methods provide a viable alternative. Rather than relying on gradient information, DFO algorithms use only function evaluations to iteratively explore the search space and identify optima. These methods are particularly effective in settings where the objective function is expensive to evaluate, non-convex, or highly constrained. The following sections introduce various DFO techniques, focusing on their underlying principles, advantages, and applicability to modern optimization challenges.

2.2.2.1 Nelder-Mead Method

One of the most well-known Derivative-Free Optimization methods is the Nelder-Mead method, introduced by Nelder and Mead (1965). This simplex-based algorithm is designed for unconstrained optimization of nonlinear functions and operates by iteratively refining a geometric simplex, which is a set of $n + 1$ points in an n-dimensional space.

At each iteration, the algorithm evaluates the objective function at the vertices of the simplex and replaces the worst-performing vertex with a better candidate point. The algorithm employs four main operations to modify the simplex: reflection, expansion, contraction, and shrinkage. These operations are defined as follows:

- **Reflection**: Reflect the worst vertex $\mathbf{x}_h$ through the centroid $\mathbf{c}$ of the remaining vertices:
$$\mathbf{x}_r = \mathbf{c} + \alpha(\mathbf{c} - \mathbf{x}_h), \tag{2.22}$$
where $\alpha > 0$ is the reflection coefficient, typically set to 1.

- **Expansion**: If the reflected point $\mathbf{x}_r$ is better than all other vertices, an expansion is attempted:
$$\mathbf{x}_e = \mathbf{c} + \gamma(\mathbf{x}_r - \mathbf{c}), \tag{2.23}$$
where $\gamma > 1$ is the expansion coefficient.

- **Contraction**: If the reflected point $\mathbf{x}_r$ is not better than the current best, a contraction is performed:
$$\mathbf{x}_c = \mathbf{c} + \rho(\mathbf{x}_h - \mathbf{c}), \tag{2.24}$$

where $0 < \rho < 1$ is the contraction coefficient.

- **Shrinkage**: If no improvement is observed, the entire simplex is shrunk towards the best vertex:
$$\mathbf{x}_i = \mathbf{x}_b + \sigma(\mathbf{x}_i - \mathbf{x}_b), \tag{2.25}$$
for all vertices i, where $0 < \sigma < 1$ is the shrinkage coefficient, and $\mathbf{x}_b$ is the best vertex.

The Nelder-Mead method does not require gradient information and is particularly suited for problems where the objective function is noisy, discontinuous, or non-differentiable. However, the algorithm may converge to non-stationary points in certain cases, especially in high-dimensional spaces or poorly scaled functions. Despite these limitations, its simplicity and effectiveness have made it a widely used tool in derivative-free optimization.

2.2.2.2 Genetic Algorithm (GA)

Another prominent approach in DFO is the Genetic Algorithm (GA), inspired by the principles of natural selection and evolutionary biology. GAs are population-based meta-heuristic methods that iteratively evolve a population of candidate solutions to optimize an objective function. These algorithms are particularly effective for global optimization in complex, multimodal, or discrete search spaces.

A typical GA involves the following key steps:

- **Initialization**: A population of candidate solutions, often represented as binary strings (or other encodings), is randomly generated.
- **Evaluation**: The objective function is evaluated for each candidate solution, assigning a fitness value that measures its quality.
- **Selection**: Candidate solutions are selected for reproduction based on their fitness. Common selection strategies include roulette wheel selection, tournament selection, and rank-based selection.
- **Crossover (Recombination)**: Pairs of selected candidates undergo crossover to produce offspring by exchanging segments of their representations. For example, in single-point crossover, a crossover point is chosen, and the segments are swapped:
$$\text{Offspring 1} = \text{Parent 1}_{[1:c]} + \text{Parent 2}_{[c+1:end]}, \tag{2.26}$$
$$\text{Offspring 2} = \text{Parent 2}_{[1:c]} + \text{Parent 1}_{[c+1:end]}, \tag{2.27}$$
where c is the crossover point.
- **Mutation**: To maintain diversity in the population and avoid premature convergence, mutation is applied to the offspring by randomly altering one or more bits or values in their representation:
$$\text{Mutated Offspring} = \text{Original Offspring} + \delta, \tag{2.28}$$
where δ represents a small random change.
- **Replacement**: The offspring replace some or all members of the current population, creating a new generation.

This process repeats over multiple generations, gradually improving the quality of solutions. The stopping criterion can be a fixed number of generations, a threshold for fitness improvement, or other problem-specific conditions. GAs are highly flexible and can be adapted to a wide range of optimization problems by tailoring the encoding scheme, fitness function, and genetic operators. While GAs do not guarantee convergence to a global optimum, their ability to explore large and complex search spaces makes them a popular choice for black-box and combinatorial optimization problems.

2.2.2.3 Simulated Annealing (SA)

Another widely used Derivative-Free Optimization method is Simulated Annealing (SA), inspired by the annealing process in metallurgy, where a material is heated and then slowly cooled to reduce defects and reach a more stable state. First proposed by Kirkpatrick, Gelatt Jr, and Vecchi (1983), SA is a probabilistic technique for finding a global optimum in a large search space, particularly for non-convex or combinatorial optimization problems.

SA operates by iteratively exploring the solution space, accepting new solutions based on their quality and a probabilistic acceptance criterion. The main steps are as follows:

- **Initialization**: Start with an initial solution $\mathbf{x}_0$ and an initial temperature T_0. Define a cooling schedule to gradually reduce the temperature.
- **Generate a Neighbor**: At each iteration k, generate a new candidate solution $\mathbf{x}_{k+1}$ by applying a small random perturbation to the current solution $\mathbf{x}_k$.
- **Acceptance Criterion**: Evaluate the change in objective function value, $\Delta f = f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)$. Accept the new solution with a probability:

$$P = \begin{cases} 1 & \text{if } \Delta f < 0, \\ \exp\left(-\frac{\Delta f}{T_k}\right) & \text{if } \Delta f \geq 0, \end{cases} \tag{2.29}$$

  where T_k is the temperature at iteration k. This allows the algorithm to escape local minima by occasionally accepting worse solutions.
- **Update the Temperature**: Gradually decrease the temperature according to a cooling schedule, such as $T_{k+1} = \alpha T_k$, where $0 < \alpha < 1$ is the cooling rate.

The algorithm terminates when the temperature reaches a predefined threshold or after a fixed number of iterations. By balancing exploration and exploitation through the temperature parameter, Simulated Annealing is capable of escaping local optima and exploring the global search space effectively.

Despite its simplicity, the performance of Simulated Annealing depends significantly on the cooling schedule and parameter settings. Its versatility and ability to handle complex optimization problems have made it a popular choice in fields ranging from logistics to engineering design.

2.2.2.4 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

One of the most advanced and widely used Derivative-Free Optimization methods is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), introduced by

Hansen and Ostermeier (2001). CMA-ES is a population-based algorithm that excels in solving complex, multimodal, and high-dimensional optimization problems. It is particularly effective for continuous domains and has been successfully applied in various scientific and engineering applications. CMA-ES operates by iteratively evolving a population of candidate solutions, using a multivariate normal distribution to sample new solutions and adaptively updating its parameters. The main components are as follows:

- **Initialization**: Start with an initial mean vector $\mathbf{m}_0$, an initial covariance matrix $\mathbf{C}_0 = \sigma_0^2\mathbf{I}$, and a step size σ_0. Define the population size λ and the weights w for weighted recombination.
- **Sampling**: At each iteration k, generate λ offspring solutions by sampling from the current multivariate normal distribution:

$$\mathbf{x}_i^{(k)} \sim \mathcal{N}(\mathbf{m}_k, \sigma_k^2\mathbf{C}_k), \quad i = 1, \ldots, \lambda. \tag{2.30}$$

- **Evaluation and Selection**: Evaluate the objective function for each offspring $\mathbf{x}_i^{(k)}$ and rank them based on their fitness. Select the top μ solutions ($\mu < \lambda$) for recombination.
- **Recombination**: Compute the new mean vector as a weighted sum of the selected solutions:

$$\mathbf{m}_{k+1} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}^{(k)}, \tag{2.31}$$

  where $\mathbf{x}_{i:\lambda}$ denotes the i-th best solution.
- **Covariance Matrix Adaptation**: The covariance matrix $\mathbf{C}_k$ is updated to reflect the distribution of successful steps, capturing correlations among variables:

$$\mathbf{C}_{k+1} = (1 + c_1\delta(h_\sigma) - c_1 - c_\mu \sum_{j=1}^{\mu} w_j)\mathbf{C}_k + c_1\mathbf{p}_c\mathbf{p}_c^\top + c_\mu \sum_{i=1}^{\lambda} w_i^{\mathrm{o}}\mathbf{y}_{i:\lambda}\mathbf{y}_{i:\lambda}^\top, \tag{2.32}$$

  where $\mathbf{p}_c$ is the evolution path for the covariance matrix, $\mathbf{y}_{i:\lambda} = \frac{\mathbf{x}_{i:\lambda} - \mathbf{m}_k}{\sigma_k}$, and c_1, c_μ are adaptation parameters. The term $\delta(h_\sigma)$ incorporates the Heaviside function:

$$h_\sigma = \begin{cases} 1, & \text{if } \frac{\|\mathbf{p}_\sigma\|}{\sqrt{1-(1-c_\sigma)^{2(g+1)}}} < \left(1.4 + \frac{2}{n+1}\right) \mathbb{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|, \\ 0, & \text{otherwise.} \end{cases} \tag{2.33}$$

  This condition prevents a too fast increase of axes of $\mathbf{C}_k$ when the step size is initially set too small or the objective function changes over time. The weights w_i^{o} are calculated as:

$$w_i^{\mathrm{o}} = w_i \times \begin{cases} 1, & \text{if } w_i \geq 0, \\ \frac{n}{\|\mathbf{C}^{-\frac{1}{2}}\mathbf{y}_{i:\lambda}\|^2}, & \text{otherwise.} \end{cases} \tag{2.34}$$

  Here, n is the problem dimension, and $\mathbf{C}^{-\frac{1}{2}}$ is the inverse square root of the covariance matrix. This adjustment ensures that the covariance matrix adaptation is robust and efficient, even in challenging optimization landscapes.

- **Evolution Paths**: CMA-ES uses evolution paths to track the progress of the search and guide the adaptation of the covariance matrix and step size:
  - The evolution path for the covariance matrix, $\mathbf{p}_c$, accumulates successive step directions:

$$\mathbf{p}_c = (1 - c_c)\mathbf{p}_c + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}}\frac{\mathbf{m}_{k+1} - \mathbf{m}_k}{\sigma_k}, \tag{2.35}$$

    where c_c is the learning rate, and μ_{eff} is the effective number of selected solutions.
  - The evolution path for step size, $\mathbf{p}_\sigma$, adapts the step size dynamically based on the normalized step directions:

$$\mathbf{p}_\sigma = (1 - c_\sigma)\mathbf{p}_\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}\mathbf{C}_k^{-\frac{1}{2}}\frac{\mathbf{m}_{k+1} - \mathbf{m}_k}{\sigma_k}, \tag{2.36}$$

    where c_σ is the learning rate for step size adaptation.
- **Step Size Adaptation**: Adjust the step size σ_k using the evolution path $\mathbf{p}_\sigma$:

$$\sigma_{k+1} = \sigma_k \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma\|}{\mathbb{E}[\|\mathcal{N}(0, \mathbf{I})\|]} - 1\right)\right), \tag{2.37}$$

  where d_σ is a damping factor.

CMA-ES balances exploration and exploitation through the adaptive covariance matrix and step size, making it highly robust and effective in diverse optimization tasks. While computationally intensive due to the covariance matrix updates, its ability to navigate complex landscapes and adapt to problem structure has made CMA-ES a benchmark algorithm in the field of black-box optimization.

2.3 Black-box Optimization

2.3.1 Introduction to Black-box Optimization

In previous sections, Gradient-Based Optimization and Derivative-Free Optimization are introduced. While gradient-based methods leverage explicit derivative information and derivative-free methods only rely on function evaluations, both assume some level of access to the structure or behavior of the objective function. Black-box Optimization (BO), in contrast, deals with scenarios where the objective function is a black-box, allowing evaluation only through expensive and potentially noisy queries. This requires suitable approaches that balance efficiency and accuracy while navigating the unknown landscape of the function. Mathematically, BO is the process of finding the optimal solution to a problem where the objective function $f(\mathbf{x})$ is unknown or analytically intractable. The function $f(\mathbf{x})$ is accessible only through point-wise evaluations, typically via simulations, experiments, or complex computations. This makes BO particularly challenging, as each evaluation can be computationally expensive or time-consuming.

Problem Definition The goal of black-box optimization is to solve the following problem:

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

where $\mathcal{X} \subseteq \mathbb{R}^d$ represents the feasible domain, which may include constraints. Unlike traditional optimization methods, BO assumes no explicit knowledge of $f(x)$, such as its gradients or function form. Black-box functions often exhibit the following characteristics:

- **Expensive Evaluations**: Each query to $f(x)$ incurs a high computational or physical cost.
- **High Dimensionality**: The input space $\mathcal{X}$ may be high-dimensional, increasing the difficulty of efficient exploration.
- **Noisy Observations**: Evaluations of $f(x)$ may be corrupted by noise, represented as:
$$y(x) = f(x) + \epsilon,$$
where ϵ is the noise and often assumed to follow a Gaussian distribution $\epsilon \sim \mathcal{N}(0, \sigma^2)$
- **Complex Constraints**: The search space $\mathcal{X}$ may include unknown or black-box constraints $g_i(x) \leq 0, i = 1, \ldots, m$

A key challenge in BO is balancing "exploration", searching unvisited areas to discover new optima, and "exploitation", refining evaluations near promising regions. This balance is often managed using acquisition functions $a(x)$, guiding the search by estimating the utility of sampling at x.

Let $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n$ represent the data collected after n evaluations, where $x_i \in \mathcal{X}$ are the sampled points and $y_i = f(x_i) + \epsilon$ are the observed responses. A surrogate model $\hat{f}(x|\mathcal{D}_n)$ is constructed to approximate $f(x)$ based on $\mathcal{D}_n$. The next evaluation point x_{n+1} is chosen to optimize the acquisition function:

$$x_{n+1} = \arg\min_{x \in \mathcal{X}} a(x|\mathcal{D}_n).$$

2.3.2 Surrogates Models

In black-box optimization, the objective function is often expensive to evaluate, high-dimensional, and analytically intractable. To address these challenges, surrogate models are employed as efficient approximations of the true objective function. These models, such as Gaussian Processes (GPs), Deep Neural Networks (DNNs), or Random Forests (RFs), act as computationally inexpensive proxies that capture the underlying structure of the objective function using a limited number of evaluations. By iteratively refining the surrogate model with newly acquired data, the optimization process can efficiently explore and exploit the search space. Surrogate models not only reduce the number of expensive function evaluations required but also provide probabilistic estimates of uncertainty, enabling the use of acquisition functions to guide the search toward promising regions. This approach is foundational in frameworks such as Bayesian optimization, where the balance between exploration and exploitation is critical.

2.3.2.1 Deep Neural Network and Neural Tangent Kernel

2.3.3 Utility-Based Acquisition Functions

In Bayesian optimization, the objective is to efficiently locate the global optimum of an expensive-to-evaluate function, a task particularly challenging in scenarios

where gradient information is unavailable or unreliable (Mockus, Tiesis, and Zilinskas, 1978; Jones, Schonlau, and Welch, 1998). Traditional optimization methods, often reliant on such gradient information, fall short in these contexts, necessitating alternative strategies. Bayesian optimization addresses this problem by employing a probabilistic surrogate model, often a Gaussian Process (GP) (Rasmussen, Williams, et al., 2006), to approximate the objective function. This surrogate model provides not only predictions of the objective function's values at unobserved points but also measures of uncertainty associated with these predictions. The selection of the next evaluation point is critical to the success of this methodology and is guided by *acquisition functions*. These functions quantify the utility or desirability of evaluating a given point based on the model's predictions and uncertainty, effectively navigating the trade-off between *exploration* of less-sampled regions and *exploitation* of regions that have yielded promising results [Citation1]. The choice of acquisition function plays a pivotal role in the optimization process, determining the efficiency and effectiveness of convergence to the global optimum [Citation2, Citation3]. The following subsections will detail several widely used utility-based acquisition functions, emphasizing their mathematical formulations, the underlying intuitions, and their relative strengths and weaknesses.

2.3.3.1 Probability of Improvement

The Probability of Improvement (PI) acquisition function, introduced by Kushner (1964), is a straightforward yet effective strategy for guiding Bayesian optimization, with the specific purpose of improving upon the current best-observed value. Unlike UCB and TS, PI does not explicitly incorporate the uncertainty of the prediction directly into its selection criteria; instead, it focuses only on the probability of improvement over the current best observation. The simplicity and computational efficiency of PI have made it a popular technique for many BO problems, especially when computational resources are limited, or quick convergence is preferred.

Given the best-observed objective function value $f(\mathbf{x}^+)$ up to the current iteration, and the predictive mean $\mu(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$ of a point $\mathbf{x}$, the PI function is defined as:

$$\alpha_{\text{PI}}(\mathbf{x}) = P(f(\mathbf{x}) > f(\mathbf{x}^+) + \xi) = \Phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})}\right) \tag{2.38}$$

where:

- $f(\mathbf{x}^+)$ is the best function value observed so far in the optimization process;
- $\mu(\mathbf{x})$ is the predicted mean of the objective function at the point $\mathbf{x}$;
- $\sigma(\mathbf{x})$ is the predicted standard deviation (or uncertainty) of the objective function at the point $\mathbf{x}$;
- $\Phi(\cdot)$ is the Cumulative Distribution Function (CDF) of the standard Normal distribution, which is used since it is assumed that the objective function $f(\mathbf{x})$ is distributed as a Gaussian;
- $\xi \geq 0$ is an optional hyperparameter that allows a trade-off between exploration and exploitation.

Equation (2.38) mathematically expresses the probability that the true objective function value at a location $\mathbf{x}$, denoted as $f(\mathbf{x})$, will be greater than the current best

value seen so far, $f(\mathbf{x}^+)$, with an optional margin ξ to encourage exploration. This probability is computed under the assumption that $f(\mathbf{x})$ follows a Gaussian distribution with the predicted mean $\mu(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$, as modeled by the surrogate function (e.g., GP). The term $\frac{\mu(\mathbf{x})-f(\mathbf{x}^+)-\xi}{\sigma(\mathbf{x})}$ represents the normalized distance from the predicted mean to the current best value, adjusted by the exploration parameter ξ and the uncertainty $\sigma(\mathbf{x})$.

The PI function prioritizes points that are likely to lead to an improvement over the current best-observed value. This intuition makes it useful for situations when the optimization goal is primarily about exploiting promising regions of the search space. However, this behavior contrasts with acquisition functions such as Expected Improvement (EI), which incorporate both the magnitude and probability of improvement, or Upper Confidence Bound (UCB), which balances exploration and exploitation more explicitly.

While PI is straightforward to understand and implement due to its simple formulation, it has limitations. A key benefit of PI is its computational efficiency, requiring only basic calculations to determine the next evaluation point. Additionally, it typically performs well in scenarios where the objective function is unimodal or when exploitation is prioritized. However, PI's greedy nature can result in overly local search behavior. By focusing exclusively on high-probability regions, it may neglect areas with higher uncertainty, leading to poor exploration. This can cause PI to converge prematurely to local optima, particularly in multi-modal or high-dimensional optimization problems where global exploration is crucial.

To mitigate this issue, the exploration parameter ξ can be introduced, increasing the focus on regions with higher uncertainty. Larger values of ξ encourage exploration, but tuning this parameter is non-trivial and problem-specific.

2.3.3.2 Expected Improvement

Expected Improvement (EI) is an acquisition function that extends the concept of PI by quantifying the expected magnitude of improvement, rather than simply the probability of improvement (Mockus, Tiesis, and Zilinskas, 1978; Jones, Schonlau, and Welch, 1998).

The EI is defined as the expected value of the maximum between zero and the difference of the current point and the best-observed objective function value. Formally:

$$\alpha_{\text{EI}}(\mathbf{x}) = \mathbb{E}[\max(0, f(\mathbf{x}^+) - f(\mathbf{x}))]. \tag{2.39}$$

Assuming that the values of the objective function $f(\mathbf{x})$ at a location $\mathbf{x}$ are normally distributed with mean $\mu(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$, the expected improvement in equation (2.39) can be rewritten as:

$$\alpha_{\text{EI}}(\mathbf{x}) = \sigma(\mathbf{x})[z\Phi(z) + \phi(z)], \tag{2.40}$$

where: $z = \frac{f(\mathbf{x}^+)-\mu(\mathbf{x})}{\sigma(\mathbf{x})}$; $f(\mathbf{x}^+)$ is the best function value observed so far in the optimization process; $\mu(\mathbf{x})$ is the predicted mean of the objective function at point $\mathbf{x}$; $\sigma(\mathbf{x})$ is the predicted standard deviation (or uncertainty) of the objective function at point $\mathbf{x}$; $\Phi(\cdot)$ is the CDF of the standard normal distribution; and $\phi(\cdot)$ is the Probability Density Function (PDF) of the standard Normal distribution. The next evaluation point is then selected by maximizing the EI function:

$$x_t = \underset{\mathbf{x}\in\mathcal{D}}{\operatorname{argmax}}\, \alpha_{\text{EI}}(\mathbf{x})$$

The formulation in equation (2.40) allows computation of the EI by combining the mean and standard deviation predicted by a GP. Intuitively, it represents a trade-off between the standard deviation $\sigma(\mathbf{x})$, which tends to promote exploration, and the z term which tends to promote exploitation of promising areas. This makes it a more comprehensive approach compared to other methods such as PI, which focuses mainly on the probability of improvement, rather than its magnitude as well.

The EI function is designed to provide a more balanced approach than the PI acquisition function, considering both the probability and magnitude of potential improvements. The EI values are higher for locations with high predicted mean values and/or high standard deviation, since the expected improvement is a combination of the probability of improvement and the size of the improvement itself. The EI selects the point that, on average, is expected to provide the greatest improvement in the objective function relative to the best-observed value. This balanced approach allows navigating the search space in a more effective way (Shahriari et al., 2015).

EI is well regarded for providing a more robust balance between exploration and exploitation compared to PI, which often results in a more efficient optimization process and faster convergence to optimal solutions (Frazier, 2018). It has shown good empirical performance across a range of different optimization problems and has been adopted as the main acquisition function in many different Bayesian Optimization libraries. However, similar to PI, EI might not be suitable when specific high levels of accurate and precise exploration or exploitation are needed, since it still might make greedy choices and get trapped in local optima. Furthermore, its performance can also be affected by the accuracy of the Gaussian Process prediction, as its formulation depends on the model prediction for a Gaussian distribution.

2.3.3.3 Upper Confidence Bound

The Upper Confidence Bound (UCB) acquisition function is a widely adopted strategy in Bayesian optimization, particularly when a balance between exploration and exploitation is crucial (Auer, 2002; Srinivas et al., 2009). UCB operates on the principle of selecting the point with the highest upper confidence bound on its predicted value, effectively encouraging both the evaluation of points with high predicted objective values and those with high uncertainty, thus enabling a comprehensive coverage of the search space. This balance is achieved by incorporating both the predictive mean and the uncertainty (typically standard deviation) associated with that prediction into a single criterion. The UCB strategy is appealing due to its relatively simple implementation and strong theoretical underpinnings. UCB is a variant of the so-called "optimism in the face of uncertainty" principle, a well-established technique in the reinforcement learning field.

In the context of BO, at the optimization iteration t, given a set of observations $\mathcal{D}_t$ and a probabilistic predictive model, such as a GP, which provides a mean prediction $\mu_t(\mathbf{x})$ and a standard deviation $\sigma_t(\mathbf{x})$ at a new point $\mathbf{x}$, the UCB function is defined as:

$$\alpha_{\text{UCB}}(\mathbf{x}) = \mu(\mathbf{x}) + \sqrt{\beta_t}\sigma(\mathbf{x}), \tag{2.41}$$

where: $\mu_t(\mathbf{x})$ represents the predicted mean of the objective function at point $\mathbf{x}$, obtained from the predictive model; $\sigma(\mathbf{x})$ denotes the predicted standard deviation (or uncertainty) of the objective function at point $\mathbf{x}$, also derived from the predictive model; and β_t is a hyperparameter, often referred to as the exploration-exploitation

trade-off parameter, that controls the relative importance of exploration and exploitation. A higher β_t encourages greater exploration, prioritizing the evaluation of points with high uncertainty, which typically correspond to less sampled regions.

UCB is relatively straightforward to implement and computationally efficient, requiring only simple arithmetic operations given the predictive mean and standard deviation. It has been shown to perform well in practice across a range of optimization problems, particularly in the early stages of optimization, because of its tendency to explore unknown regions efficiently. It is also accompanied by strong theoretical guarantees for convergence under certain conditions, which provides a formal justification for its effectiveness and is also a factor in its popularity among practitioners.

Following (Srinivas et al., 2009), assuming the chosen kernel has the smooth property for some constants $a, b, L > 0$:

$$\mathbb{P}\left\{\sup_{x\in\mathcal{X}}\left|\frac{\partial f}{\partial x_i}\right| > L\right\} \leq ae^{-(L/b)^2}, \quad i = 1\dots d,$$

then by choosing

$$\beta_t = 2\log\left(\frac{t^2\pi^2}{3\delta}\right) + 2d\log\left(t^2dbr\sqrt{\log\left(\frac{4da}{\delta}\right)}\right),$$

where d is the number of dimension, GP-UCB algorithm is proved to achieve sublinear regret bound with probability $1-\delta$. However, in practice, the performance of UCB is inherently sensitive to the hyperparameter β_t, which typically needs to be chosen by the user based on some empirical considerations and may require tuning and experimentation for each specific problem. The selection of β_t is not obvious and depends largely on the characteristics of the objective function. The simple linear combination structure used by UCB might struggle to effectively navigate complex and highly non-linear objective function landscapes, potentially leading to suboptimal results in high-dimensional search spaces or with highly multimodal objective functions.

2.3.3.4 Thompson Sampling

Thompson Sampling (TS), in contrast to UCB, is a probabilistic acquisition function that bases its decisions on samples drawn from the posterior distribution over the objective function (Thompson, 1933; Russo et al., 2018). This stochastic decision-making process provides a natural mechanism for both exploration and exploitation, in a different manner compared to the UCB acquisition function. TS has proven to be a powerful approach, and it is now widely adopted in reinforcement learning, Bayesian optimization, and many other fields, where it has been shown to perform very well in different settings (Agrawal et al., 2017; Chowdhury and Gopalan, 2017).

Rather than optimizing an explicit acquisition function based on a balance of mean and uncertainty like UCB, TS utilizes a posterior sampling approach.

Thompson Sampling (TS) does not have an explicit analytical formula for the acquisition function itself. Instead, it relies on the following iterative process, which is grounded in sampling the posterior function:

1. *Posterior Sampling*: At each step t, sample a function $\hat{f}_t$ from the posterior distribution given the observed data, i.e., sample $\hat{f}_t \sim P(f \mid \mathcal{D}_t)$. In the

case of a GP as the predictive model, this involves sampling a mean function from the posterior with the covariance also defined by the posterior. This is achieved by sampling from a multivariate Gaussian distribution. For a GP, the posterior distribution of the objective function $f(x)$ at a new point x is given by the formula in Section 2.1.2. Then TS with GP has a general form: $\hat{f}_t(\cdot) \sim \mathcal{GP}(\mu_t(\cdot), \nu^2\sigma_t(\cdot))$. Here, ν us the exploration parameter and can be time-dependent.

2. *Minimization*: Find the next location, $\mathbf{x}_{\text{new}}$, by minimizing the value of the sampled function $\hat{f}(\mathbf{x})$, i.e., find the $\mathbf{x}$ such that $\mathbf{x}_{\text{new}} = \arg\min_{\mathbf{x}} \hat{f}_t(\mathbf{x})$.

This process is repeated at each step to select the next evaluation point, thus making TS an iterative and adaptive procedure, rather than a method where one can easily calculate and evaluate an acquisition function. The power of TS lies in this step-by-step process, which balances both the exploitation and exploration capabilities.

TS's inherent stochastic nature is key to its success in exploration and exploitation. Locations with higher uncertainty are more likely to be sampled due to the increased variance of the posterior distribution in those regions, which inherently introduces exploration. However, locations where the best values have been observed so far are also more likely to be sampled, leading to focused optimization. In a very natural way, TS balances the need to explore new, unexplored regions and the need to exploit the information about previously seen promising ones. This behavior is what makes it a good candidate for Bayesian optimization, as well as many other settings.

However, the performance of TS depends heavily on the quality of the posterior approximation and the sampling method used. For GPs, the posterior is typically a multivariate Gaussian distribution, and sampling involves drawing from this distribution at each step. In practice, the computational overhead associated with repeatedly sampling from the posterior can be higher compared to methods that rely on closed-form acquisition functions such as UCB. For large-scale problems, this could lead to scalability issues, as sampling from the posterior involves evaluating the covariance matrix, which can become computationally expensive in high-dimensional spaces.

The use of sampling-based methods such as TS also introduces the possibility of overfitting, especially when the model is highly flexible or the amount of observed data is small. If the posterior distribution becomes overly confident in its predictions due to a limited number of samples, the algorithm may prematurely focus on regions that do not lead to further improvements. This can be mitigated through techniques like regularization or the use of more robust probabilistic models.

In certain cases, especially when the posterior distribution is complex, methods such as Markov Chain Monte Carlo (MCMC) are used to sample from the posterior more efficiently. While MCMC can offer more accurate posterior samples, it can also introduce additional computational complexity. The trade-off between accuracy and computational cost is an important consideration when implementing Thompson Sampling (TS).

The absence of a direct exploration-exploitation parameter, like $\sqrt{\beta_t}$ in UCB, allows Thompson Sampling (TS) to adapt more fluidly to the problem at hand. However, its success depends on the accuracy of the posterior approximation, and its performance may degrade if the model fails to capture the true structure of the objective function. Thus, while Thompson Sampling (TS) can achieve robust performance, its implementation and tuning require careful consideration of the underlying model and the sampling methods used.

2.3.4 Information-Theoretic Acquisition Functions

Information-theoretic acquisition functions offer a different perspective on the exploration-exploitation trade-off in Bayesian optimization. Instead of directly maximizing a utility function based on predicted means and uncertainties, they aim to maximize the information gained about the objective function, specifically the location of its global optimum. By focusing on information gain, these functions can be particularly effective in scenarios where the primary goal is to minimize the uncertainty associated with the location of the optimal solution. This approach often results in more directed exploration in specific regions. The following subsections detail several widely used information-theoretic acquisition functions, discussing their mathematical formulations, intuitive motivations, and practical implications.

2.3.4.1 Entropy Search

Entropy Search (ES) proposed by Hennig and Schuler (2012) is an information-theoretic acquisition function designed to identify the next evaluation point in BO by maximizing the expected reduction in entropy of the posterior distribution over the location of the global optimum. Unlike other methods that emphasize the function's value directly, ES concentrates on reducing the uncertainty regarding the location of the optimum, making it particularly useful in multi-modal or highly complex optimization landscapes.

Given a posterior distribution over the objective function f and its corresponding location of the optimum $\mathbf{x}^*$, ES aims to maximize the expected information gain about $\mathbf{x}^*$ obtained by observing the function value at a new point $\mathbf{x}$. This process naturally links the concepts of uncertainty reduction and optimization, as reducing uncertainty about $\mathbf{x}^*$ guides the optimization process more effectively.

As presented in Section 2.1.4.1, the entropy of a random variable quantifies the uncertainty associated with its probability distribution. Therefore, selecting the point with maximum information about $\mathbf{x}^*$ is equivalent to selecting the point that reduces the uncertainty about its location. Mathematically, the acquisition function for ES can be expressed as:

$$\alpha_{\text{ES}}(\mathbf{x}) = \mathbb{E}_{f(\mathbf{x})|\mathcal{D}}\left[H(p(\mathbf{x}^*|\mathcal{D})) - H(p(\mathbf{x}^*|\mathcal{D} \cup (\mathbf{x}, f(\mathbf{x}))))\right], \tag{2.42}$$

where $H(\cdot)$ represents the entropy of a probability distribution, $p(\mathbf{x}^*|\mathcal{D})$ is the posterior distribution over the location of the global optimum $\mathbf{x}^*$ given the observed data $\mathcal{D}$, and $p(\mathbf{x}^*|\mathcal{D} \cup (\mathbf{x}, f(\mathbf{x})))$ is the updated posterior distribution after evaluating the objective function at $\mathbf{x}$. The expectation $\mathbb{E}_{f(\mathbf{x})|\mathcal{D}}$ is taken with respect to the predictive distribution of the function value at $\mathbf{x}$, conditioned on the observed data $\mathcal{D}$.

The term $H(p(\mathbf{x}^*|\mathcal{D}))$ quantifies the posterior entropy of the optimum location before observing the new data point $\mathbf{x}$, while $H(p(\mathbf{x}^*|\mathcal{D} \cup (\mathbf{x}, f(\mathbf{x}))))$ represents the posterior entropy after the observation. The acquisition function seeks to maximize the reduction in entropy, prioritizing the evaluation of points that contribute the most to reducing uncertainty about $\mathbf{x}^*$.

This information-theoretic perspective naturally emphasizes exploration in regions where the model is uncertain while also focusing on exploitation near promising candidates for the global optimum. As a result, ES excels in scenarios where a balance between exploration and exploitation is crucial. For instance, in multi-modal problems where local optima might mislead the optimization process, ES ensures that the search considers broader regions of the parameter space.

However, while the theoretical formulation of ES is elegant and well-suited to complex optimization problems, it is computationally intensive. The evaluation of the entropy terms and the expectation over the posterior distributions often requires solving high-dimensional integrals, which can be challenging and time-consuming. Approximations and sampling methods are typically employed to make the computations tractable, but these can introduce additional complexity and computational overhead. Consequently, while ES offers significant advantages in terms of guiding the search effectively, its application is often limited to problems with manageable computational demands or where its potential benefits justify the additional cost.

2.3.4.2 Predictive Entropy Search

Predictive Entropy Search (PES) of Hernández-Lobato, Hoffman, and Ghahramani (2014) is an advanced information-theoretic acquisition function that builds upon the principles of ES. While ES focuses on reducing the entropy of the posterior distribution over the location of the global optimum, PES offers a computationally efficient reformulation by directly evaluating the expected reduction in entropy of the predictive distribution over the function values.

The fundamental idea behind PES is to quantify the expected information gain about the location of the global optimum $\mathbf{x}^*$ when observing the objective function at a new point $\mathbf{x}$. However, instead of working with the posterior distribution over $\mathbf{x}^*$, PES reformulates the acquisition function by focusing on the entropy of the predictive distribution of the function values. This reformulation simplifies the computation while preserving the core objective of reducing uncertainty about $\mathbf{x}^*$. The acquisition function for PES is defined as:

$$\alpha_{\text{PES}}(\mathbf{x}) = H(p(f(\mathbf{x})|\mathcal{D})) - \mathbb{E}_{\mathbf{x}^*|\mathcal{D}}\left[H(p(f(\mathbf{x})|\mathcal{D}, \mathbf{x}^*))\right], \tag{2.43}$$

where:

- $H(\cdot)$ denotes the entropy of a probability distribution.
- $p(f(\mathbf{x})|\mathcal{D})$ is the predictive distribution over the function value at $\mathbf{x}$ given the observed data $\mathcal{D}$.
- $p(f(\mathbf{x})|\mathcal{D}, \mathbf{x}^*)$ is the predictive distribution over the function value at $\mathbf{x}$, conditioned on the data $\mathcal{D}$ and the location of the optimum $\mathbf{x}^*$.
- $\mathbb{E}_{\mathbf{x}^*|\mathcal{D}}$ represents the expectation over the posterior distribution of the global optimum location $\mathbf{x}^*$ given the observed data $\mathcal{D}$.

The first term, $H(p(f(\mathbf{x})|\mathcal{D}))$, represents the entropy of the predictive distribution over the function value at $\mathbf{x}$ before incorporating any new information about the global optimum. The second term, $\mathbb{E}_{\mathbf{x}^*|\mathcal{D}}\left[H(p(f(\mathbf{x})|\mathcal{D}, \mathbf{x}^*))\right]$, represents the expected entropy of the predictive distribution after accounting for the global optimum's posterior distribution. By maximizing the difference between these terms, PES selects the evaluation point $\mathbf{x}$ that provides the most expected information about the global optimum.

The reformulation introduced by PES offers two significant advantages over traditional ES. First, the entropy terms in PES are evaluated directly on the predictive distributions of function values rather than the posterior over $\mathbf{x}^*$, which can simplify computation. Second, the sampling-based approach to evaluate $\mathbb{E}_{\mathbf{x}^*|\mathcal{D}}$ enables PES to scale better to higher-dimensional problems and more complex posterior distributions, where exact computation of the entropy terms in ES becomes intractable.

While PES shares many strengths with ES, such as its ability to balance exploration and exploitation effectively and its suitability for multi-modal optimization problems, it also addresses some of the computational challenges inherent in ES. However, PES still requires accurate sampling from the posterior over $\mathbf{x}^*$ and evaluations of the predictive distributions, which can introduce computational overhead and implementation complexity.

2.3.4.3 Max-value Entropy Search

Max-value Entropy Search (MES) (Wang and Jegelka, 2017) is another information-theoretic acquisition function designed to efficiently guide the search for the global optimum of a black-box function. Unlike methods such as ES and PES, which focus on the location of the optimum or the predictive entropy of function values, MES directly considers the uncertainty about the maximum function value itself. This focus provides a computationally efficient and intuitive way to prioritize evaluations.

The central idea behind MES is to quantify the expected reduction in entropy of the maximum value of the objective function, y^*, after observing a new function evaluation. Here, y^* represents the maximum value that the objective function can achieve over the entire search space. By reducing uncertainty about y^*, MES implicitly narrows the search toward regions likely to contain the global optimum.

The acquisition function for MES is defined as:

$$\alpha_{\text{MES}}(\mathbf{x}) = H(p(y^*|\mathcal{D})) - \mathbb{E}_{f(\mathbf{x})|\mathcal{D}}\left[H(p(y^*|\mathcal{D} \cup \{(\mathbf{x}, f(\mathbf{x}))\}))\right], \tag{2.44}$$

where:

- $H(\cdot)$ denotes the entropy of a probability distribution.
- $p(y^*|\mathcal{D})$ is the current posterior distribution over the maximum value y^*, given the observed data $\mathcal{D}$.
- $p(y^*|\mathcal{D} \cup \{(\mathbf{x}, f(\mathbf{x}))\})$ is the updated posterior distribution over y^* after observing the function value $f(\mathbf{x})$ at $\mathbf{x}$.
- $\mathbb{E}_{f(\mathbf{x})|\mathcal{D}}$ represents the expectation over the predictive distribution of $f(\mathbf{x})$ conditioned on the observed data $\mathcal{D}$.

The first term, $H(p(y^*|\mathcal{D}))$, captures the current uncertainty about the maximum value y^* across the search space, while the second term, $\mathbb{E}_{f(\mathbf{x})|\mathcal{D}}\left[H(p(y^*|\mathcal{D} \cup \{(\mathbf{x}, f(\mathbf{x}))\}))\right]$, represents the expected uncertainty after evaluating the function at $\mathbf{x}$. By maximizing the reduction in uncertainty about y^*, MES selects evaluation points that are most informative about the maximum value.

The computational efficiency of MES stems from its use of y^* as a one-dimensional summary of the optimization problem. Unlike ES, which requires sampling the posterior over the entire location of the optimum, MES simplifies the entropy computations by working directly with the scalar value y^*. Sampling-based approximations, such as Monte Carlo integration, are used to estimate the entropies, making MES practical even in higher-dimensional spaces.

MES is particularly effective in balancing exploration and exploitation. Points with high predictive uncertainty in regions where the function could exceed the current estimate of y^* are naturally prioritized, encouraging exploration of unexplored areas. Simultaneously, regions close to known high values are also favored, supporting exploitation of promising areas.

While MES addresses many computational challenges associated with other information-theoretic methods, it still requires accurate sampling from the posterior distribution of y^* and the predictive distribution of $f(\mathbf{x})$. These requirements can introduce computational overhead, especially in high-dimensional or complex posterior landscapes. However, its focused nature and simplicity often make MES more practical and scalable compared to methods like ES.

2.3.4.4 Knowledge Gradient

The Knowledge Gradient (KG) acquisition function in Frazier, Powell, and Dayanik (2008) and Wu and Frazier (2016) provides a principled way to select evaluation points by directly optimizing the improvement in the expected value of the solution to the optimization problem. Unlike entropy-based approaches such as ES or MES, which focus on reducing uncertainty about the global optimum or maximum value, KG measures the value of information gained from a single evaluation in terms of its impact on the expected objective value.

The key idea behind KG is to evaluate the utility of an experiment by estimating how much it improves the decision-making process. Specifically, KG selects the point $\mathbf{x}$ that maximizes the expected increase in the maximum posterior mean value of the objective function after observing the outcome of evaluating $f(\mathbf{x})$. Formally, the KG acquisition function is defined as:

$$\alpha_{\text{KG}}(\mathbf{x}) = \mathbb{E}_{f(\mathbf{x})|\mathcal{D}}\left[\min_{\mathbf{x}'\in\mathcal{X}} \mu_{t+1}(\mathbf{x}')\right] - \min_{\mathbf{x}'\in\mathcal{X}} \mu_t(\mathbf{x}'), \tag{2.45}$$

where:

- $\mu_t(\mathbf{x}')$ is the posterior mean of the objective function at $\mathbf{x}'$ given the current data $\mathcal{D}$.
- $\mu_{t+1}(\mathbf{x}')$ is the updated posterior mean after observing the value of $f(\mathbf{x})$ at $\mathbf{x}$.
- $\mathbb{E}_{f(\mathbf{x})|\mathcal{D}}$ denotes the expectation over the predictive distribution of $f(\mathbf{x})$ given the observed data $\mathcal{D}$.

The first term, $\mathbb{E}_{f(\mathbf{x})|\mathcal{D}}\left[\min_{\mathbf{x}'\in\mathcal{X}} \mu_{t+1}(\mathbf{x}')\right]$, represents the expected maximum posterior mean after evaluating $f(\mathbf{x})$. The second term, $\min_{\mathbf{x}'\in\mathcal{X}} \mu_t(\mathbf{x}')$, is the current minimum posterior mean. The KG acquisition function seeks to maximize the difference between these two quantities, guiding evaluations to points that are expected to improve the decision about the best location.

The KG approach has several appealing properties. By directly focusing on the impact of new information on decision-making, KG inherently balances exploration and exploitation. Points in regions with high uncertainty are likely to lead to significant updates in the posterior mean, encouraging exploration. Meanwhile, points near current estimates of the maximum mean are prioritized for exploitation.

KG is particularly effective in problems where evaluations are expensive, and it is crucial to optimize every experiment to maximize learning. Its ability to quantify the value of information gained from a single evaluation makes it a strong candidate for sequential optimization settings. However, implementing KG can be computationally demanding. The expectation over the predictive distribution and the maximization over the posterior mean both require accurate approximation techniques, such as Monte Carlo integration or surrogate modeling with Gaussian processes. In high-dimensional or highly multimodal problems, the computational cost can increase significantly, requiring careful optimization strategies.

2.3.5 Bandit-based Optimization

Bandit-based optimization is a foundational framework in sequential decision-making and optimization under uncertainty. The central objective is to maximize cumulative rewards over time while balancing exploration (gathering information about suboptimal choices) and exploitation (leveraging known high-reward choices). This paradigm is often used in various applications such as recommendation systems, clinical trials, and Bayesian optimization (Lattimore and Szepesvári, 2020; Bubeck, Cesa-Bianchi, et al., 2012).

2.3.5.1 Multi-armed Bandits

The Multi-armed Bandit (MAB) problem is a classic formulation in the bandit optimization framework. It derives its name from a metaphorical scenario involving a gambler faced with a row of slot machines (or "one-armed bandits"), each with an unknown probability distribution of payouts. The gambler must decide which machines to play, in what order, and how many times, to maximize their total reward over a sequence of trials (Robbins, 1952). Formally, the MAB problem can be described as follows:

- There are K arms (options), indexed by $i \in \{1, 2, \ldots, K\}$.
- Each arm i is associated with a reward distribution P_i, with an unknown mean reward $\mu_i = \mathbb{E}[r_i]$, where r_i is the reward obtained from pulling arm i.
- At each time step $t \in \{1, 2, \ldots, T\}$, the player selects an arm $a_t \in \{1, 2, \ldots, K\}$ and observes a stochastic reward $r_{a_t} \sim P_{a_t}$.
- The goal is to maximize the cumulative reward over T rounds:

$$R_T = \sum_{t=1}^{T} r_{a_t}. \tag{2.46}$$

The core challenge lies in the trade-off between *exploration* - pulling less-sampled arms to estimate their reward distributions and *exploitation* - pulling arms with known high rewards to accumulate gains. This trade-off is typically quantified using the concept of **regret**, which quantifies the cumulative loss incurred by not always selecting the optimal arm.

Regret Minimization Regret measures the difference between the cumulative reward of an optimal strategy and the actual cumulative reward obtained by a given algorithm. Formally, the *cumulative regret* $R(T)$ is defined as:

$$R(T) = T\mu^* - \mathbb{E}\left[\sum_{t=1}^{T} r_{a_t}\right], \tag{2.47}$$

where $\mu^* = \max_{i \in \{1,\ldots,K\}} \mu_i$ is the mean reward of the optimal arm. The objective of any bandit algorithm is to minimize $R(T)$ as T grows large.

2.3.5.2 Algorithms for MAB

Several algorithms have been developed to solve the MAB problem effectively. Key strategies include:

ϵ-Greedy Algorithm The ϵ-greedy algorithm balances exploration and exploitation by selecting the arm with the highest estimated mean reward with probability $1-\epsilon$, and exploring a random arm with probability ϵ. Formally, at each time step t, the algorithm selects:

$$a_t = \begin{cases} \arg\max_i \hat{\mu}_i & \text{with probability } 1-\epsilon, \\ \text{randomly select from } \{1,\ldots,K\} & \text{with probability } \epsilon, \end{cases} \tag{2.48}$$

where $\hat{\mu}_i$ is the estimated mean reward of arm i based on past observations.

Upper Confidence Bound Algorithm The UCB algorithm addresses the exploration-exploitation trade-off by assigning a confidence interval to each arm's estimated mean reward and selecting the arm with the highest upper confidence bound. At each step t, the arm a_t is chosen as:

$$a_t = \arg\max_i \left(\hat{\mu}_i + \sqrt{\frac{2\ln t}{n_i}} \right), \tag{2.49}$$

where n_i is the number of times arm i has been pulled, and $\ln t$ encourages exploration by weighting uncertainty higher in earlier stages. The UCB algorithm is particularly effective due to its logarithmic regret bounds (Auer, 2002).

Thompson Sampling (TS) Thompson Sampling (TS) is a Bayesian approach to MAB problems, where posterior distributions over the mean rewards of each arm are maintained and updated based on observed rewards. At each time step, an arm is sampled from its posterior distribution and played. Formally, the algorithm selects:

$$a_t = \arg\max_i \mu_i^{(t)}, \tag{2.50}$$

where $\mu_i^{(t)}$ is a sample from the posterior distribution of arm i. TS has been shown to achieve asymptotically optimal regret in many settings (Agrawal and Goyal, 2012).

2.3.5.3 Contextual Bandits

Contextual bandits extend the multi-armed bandit framework by introducing a context $\mathbf{c}_t \in \mathcal{C}$ at each round t. This context provides additional information to inform the decision-making process, allowing for more tailored arm selections. The challenge lies in effectively modeling the relationship between context and reward, as well as balancing exploration and exploitation in this extended setting. At each step, the algorithm observes a context vector $\mathbf{x}_t$ before selecting an arm. The objective is to learn a mapping from contexts to arm selection policies that maximize cumulative rewards. This framework has found widespread applications in areas such as personalized recommendations and adaptive experimentation (Li et al., 2010). In contrast to traditional multi-armed bandits, contextual bandits incorporate context. This integration allows for more adaptive and precise policy learning, which is crucial in dynamic and personalized environments.

Formally, contextual bandit problems involve a sequential decision-making process where, at each time step t, an agent observes a *context* $\mathbf{x}_t \in \mathcal{X}$, selects an *action* $a_t \in \mathcal{A}$, and receives a *reward* r_t. Unlike standard MAB problems, contextual bandits leverage the context $\mathbf{x}_t$ to inform the action selection process. The objective is

to maximize the cumulative reward over a sequence of interactions. Having established the problem setting of contextual bandits, we now present the general step of a standard Contextual Bandit Algorithm.

Steps in a Contextual Bandit Algorithm. A standard contextual bandit algorithm typically proceeds as follows:

1. **Observe context:** A new data point arrives, described by a context $\mathbf{c}_t$ (e.g., a user with specific attributes such as device type and location).
2. **Select action:** Based on the observed context and the exploration strategy (e.g., ϵ-greedy or Thompson Sampling), the algorithm selects an action a_t (e.g., a recommendation or treatment).
3. **Receive reward:** After the action is performed, the algorithm observes the outcome r_t (e.g., a user's response or purchase behavior).
4. **Update model:** The model is updated using the new context-action-reward tuple. To reduce noise, updates are often performed in batches rather than after every single sample.
5. **Repeat:** This process continues over multiple rounds to optimize the cumulative reward.

In contextual bandits, the context $\mathbf{c}_t$ acts as side information that is observed before selecting an arm. This involves a trade-off between exploring different actions to learn their potential rewards and exploiting actions that are known to produce good rewards given the current context. However, unlike classical multi-armed bandits, where exploration focuses solely on the arms, contextual bandits require exploration across both arms and the context space. This dual exploration-exploitation trade-off leads to questions such as:

- Should rare or less-sampled contexts be prioritized during exploration?
- How can the algorithm balance exploring unseen arms versus unseen contexts?

Furthermore, the success of contextual bandit algorithms heavily depends on the choice of the model used to estimate the expected reward. Hence, it introduces challenges such as modeling the reward function $r(\mathbf{c}_t, a)$, which maps context-arm pairs to rewards or managing the complexity of high-dimensional contexts while ensuring computational tractability. To address these challenges, it is crucial to select robust methods for modeling the reward function. A well-suited model not only needs to capture the underlying patterns and dependencies accurately but also maintain a balance between computational efficiency and predictive performance.

Modeling the Reward Function. $\mathbb{E}[r_t \mid \mathbf{c}_t, a]$:

- Linear models offer simplicity and computational efficiency but are limited in capturing non-linear relationships.
- Non-linear models, such as neural networks, provide flexibility but come at the cost of increased computational overhead and the need for larger datasets to generalize effectively.

To address these challenges, various algorithms have been proposed using both linear and non-linear models to estimate rewards. In the next subsections, we will explore some of these algorithms.

2.3.5.4 Linear Bandits

Linear models assume that the expected reward for a given context and action can be represented as a linear combination of some features. Specifically, we assume there exists an unknown parameter vector $\boldsymbol{\theta}$ such that the expected reward can be approximated by the inner product of this vector with a context-action feature vector:

$$\mathbb{E}[r_t|\mathbf{c}_t, a_t] = \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{c}_t, a_t) \tag{2.51}$$

where $\boldsymbol{\phi}(\mathbf{c}_t, a_t)$ is the feature mapping of context $\mathbf{c}_t$ and action a_t into a feature vector.

Linear Upper Confidence Bound (LinUCB) Algorithm LinUCB operates by maintaining an estimate of the parameter vector $\boldsymbol{\theta}$, and using confidence bounds to balance exploration and exploitation. It calculates an upper confidence bound on the expected reward using a known hyper-parameter α to balance exploration/exploitation trade-off. At each time step, the action that maximizes the upper confidence bound is selected. The algorithm works as follows:

1. Initialize $\mathbf{A}_0 = \mathbf{I}$ (identity matrix), $\mathbf{b}_0 = \mathbf{0}$, $\hat{\boldsymbol{\theta}}_0 = \mathbf{0}$.
2. At each time step t:
   (a) Observe context $\mathbf{x}_t$.
   (b) For each action $a \in \mathcal{A}$, compute:
   $$\hat{r}_t(a) = \hat{\boldsymbol{\theta}}_{t-1}^T \boldsymbol{\phi}(\mathbf{x}_t, a) + \alpha \sqrt{\boldsymbol{\phi}(\mathbf{x}_t, a)^T \mathbf{A}_{t-1}^{-1} \boldsymbol{\phi}(\mathbf{x}_t, a)}$$
   (c) Select action $a_t = \text{argmax}_{a \in \mathcal{A}} \hat{r}_t(a)$.
   (d) Observe reward r_t.
   (e) Update:
   $$\begin{aligned} \mathbf{A}_t &= \mathbf{A}_{t-1} + \boldsymbol{\phi}(\mathbf{x}_t, a_t) \boldsymbol{\phi}(\mathbf{x}_t, a_t)^T \\ \mathbf{b}_t &= \mathbf{b}_{t-1} + r_t \boldsymbol{\phi}(\mathbf{x}_t, a_t) \\ \hat{\boldsymbol{\theta}}_t &= \mathbf{A}_t^{-1} \mathbf{b}_t \end{aligned}$$

Thompson Sampling for Linear Models : Thompson Sampling (TS) for linear bandits also attempts to learn the weight vector $\boldsymbol{\theta}$ by maintaining a posterior distribution over these weights. At each step, the agent samples a weight vector from the posterior, and uses this weight vector to select the action that maximizes the reward. The algorithm works as follows:

1. Initialize the posterior using a prior distribution for $\boldsymbol{\theta}$ and a covariance matrix. Usually a gaussian distribution is used for the prior.
2. At each time step t:
   (a) Observe context $\mathbf{x}_t$.
   (b) Sample parameter $\hat{\boldsymbol{\theta}}_t$ from posterior distribution.
   (c) Select action $a_t = \text{argmax}_{a \in \mathcal{A}} \hat{\boldsymbol{\theta}}_t^T \boldsymbol{\phi}(\mathbf{x}_t, a)$.
   (d) Observe reward r_t.
   (e) Update the posterior distribution based on observation.

Linear models offer several advantages, making them a popular choice in contextual bandit algorithms. They are computationally efficient, requiring fewer resources and enabling faster computations, which is particularly valuable in large-scale applications. Their simplicity makes them easier to understand and implement, reducing the complexity of algorithm design. Additionally, linear models often come with well-established theoretical guarantees, providing performance bounds that enhance their reliability. However, these models also have notable limitations. They struggle to capture complex non-linear relationships, which can result in poor performance when the underlying connection between context, action, and reward deviates from linearity. Furthermore, their effectiveness heavily depends on the quality of feature mappings, as the performance hinges on well-designed features in $\phi(\mathbf{x}_t, a_t)$. To mitigate these limitations, proper feature engineering and careful tuning of hyperparameters are critical to achieving good performance with linear bandit algorithms. To address the limitations of linear models in capturing complex relationships, non-linear models offer a more flexible and expressive alternative.

2.3.5.5 Non-Linear Bandits

Non-linear models relax the linearity assumption and allow the expected reward for a given context and action to be represented by more complex, non-linear functions. These models aim to capture intricate relationships between the context, action, and reward, which are often observed in real-world scenarios. Formally, we assume the expected reward can be expressed as:

$$\mathbb{E}[r_t|\mathbf{c}_t, a_t] = f(\mathbf{x}_t, \boldsymbol{theta_t}), \tag{2.52}$$

where f is a non-linear function, such as a neural network, and $\mathbf{x}_t = \phi(\mathbf{c}_t, a_t)$ is the feature mapping of the context $\mathbf{c}_t$ and action a_t.

Neural Upper Confidence Bound (NeuralUCB) Algorithm NeuralUCB extends the idea of LinUCB by using a neural network to model the reward function. Instead of maintaining a linear estimate of $\boldsymbol{\theta}$, the algorithm trains a neural network to approximate the reward function, while maintaining an exploration bonus based on the model's uncertainty. The steps are as follows:

1. Initialize the neural network f_θ with random weights and set the exploration parameter α.
2. At each time step t:
   (a) Observe context $\mathbf{x}_t$.
   (b) For each action $a \in \mathcal{A}$, compute:

$$\hat{r}_t(a) = f_\theta(\phi(\mathbf{x}_t, a)) + \alpha\sqrt{u_t(\phi(\mathbf{x}_t, a))},$$

   where u_t is an uncertainty estimate (e.g., derived from a Bayesian approximation or ensemble models).
   (c) Select action $a_t = \text{argmax}_{a\in\mathcal{A}}\hat{r}_t(a)$.
   (d) Observe reward r_t and update the neural network parameters θ to minimize the prediction error.

Thompson Sampling for Non-Linear Models Thompson Sampling can also be extended to non-linear models by maintaining a distribution over the parameters of the non-linear model (e.g., neural network weights). At each time step, the algorithm samples a model from the posterior distribution, selects the action that maximizes the reward based on the sampled model, and updates the posterior using the observed data. The steps are as follows:

1. Initialize a prior distribution over the parameters of the non-linear model (e.g., a Bayesian neural network or ensemble).
2. At each time step t:
   (a) Observe context $\mathbf{x}_t$.
   (b) Sample model parameters $\hat{\theta}_t$ from the posterior distribution.
   (c) Select action $a_t = \text{argmax}_{a \in \mathcal{A}} f_{\hat{\theta}_t}(\phi(\mathbf{x}_t, a))$.
   (d) Observe reward r_t and update the posterior distribution.

Non-linear models address the limitations of linear models by providing the flexibility to capture complex and non-linear relationships in the data. This makes them particularly effective in scenarios with intricate dependencies between context, action, and reward. However, this flexibility comes with trade-offs. Non-linear models often require significantly more computational resources and larger datasets to achieve reliable generalization. Moreover, they are more prone to overfitting and require careful regularization and hyperparameter tuning. Despite these challenges, advances in neural network architectures and training techniques have made non-linear models increasingly viable for bandit problems, enabling their application to more complex and high-dimensional contexts.

Introduction to Non-Linear Models Non-linear models extend the capabilities of contextual bandits to handle situations where the reward function cannot be adequately approximated using linear models. These models allow for more complex relationships between contexts, actions, and rewards.

Key Algorithms

- **Kernelized Bandits (KernelUCB, KernelTS)**: Kernel methods are an alternative to feature-mapping based approaches. They map the data into a higher-dimensional space using a kernel function $k(x_i, x_j)$. Kernelized bandits operate by maintaining estimates of the reward function in the kernel space. KernelUCB and KernelTS are extensions of LinUCB and Thompson Sampling for linear models, which utilize the kernel trick to implicitly perform the computation of the features in the high-dimensional feature space.
- **Neural Network Based Bandits**: Neural networks can be used to model non-linear reward functions. A neural network can be trained to approximate the reward given a specific context and action. These models can be incorporated in bandit algorithms using strategies like:
  - Using ϵ-greedy strategy using neural network to predict best action.
  - Incorporating uncertainty in the neural network outputs, and then using UCB or Thompson sampling strategies to balance exploration/exploitation.

Advantages of Non-Linear Models

- **Flexibility**: They can capture more complex non-linear relationships, potentially leading to improved performance when a linear assumption is inappropriate.
- **Improved Performance**: Non-linear models often achieve superior results in scenarios where complex relationships exist between the context, action, and the reward.

Limitations of Non-Linear Models

- **Computational Costs**: Non-linear models are generally more computationally intensive and can be slower to train and apply, specially compared to linear methods.
- **Need for more data**: Require more data for training to avoid overfitting.
- **Hyperparameter Tuning**: Require careful tuning of hyperparameters to achieve optimal performance.

2.3.6 Performance Metrics in Black-box Optimization

Evaluating the performance of a Black-box Optimization (BO) algorithm involves quantifying its efficiency in identifying the optimal solution of the objective function f. Two commonly used performance metrics in the literature are *simple regret* and *cumulative regret*, which capture different aspects of the optimization process.

2.3.6.1 Simple Regret

The *simple regret* measures the difference between the best function value obtained after t evaluations and the true global maximum. Formally, it is defined as:

$$r_t = f(\mathbf{x}^*) - f(\mathbf{x}_t^*),$$

where:

- $f(\mathbf{x}^*) = \max_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x})$ is the global maximum of the objective function f,
- $\mathbf{x}_t^* = \arg\max_{i=1,\dots,t} f(\mathbf{x}_i)$ is the best input queried among the t evaluations.

Simple regret evaluates the quality of the best solution found so far without considering the total cost incurred during the optimization process. This metric is particularly relevant for applications where the goal is to identify a high-quality solution at the end of the optimization rather than during the intermediate steps.

2.3.6.2 Cumulative Regret

The *cumulative regret* quantifies the total loss incurred due to querying suboptimal points during the optimization process with optimization budget T. It is defined as:

$$R_T = \sum_{t=1}^{T} \left(f(\mathbf{x}^*) - f(\mathbf{x}_t) \right),$$

where $\mathbf{x}_t$ denotes the point queried at the t-th step.

Cumulative regret provides a measure of how efficiently the algorithm balances exploration and exploitation over t iterations. A lower cumulative regret indicates that the algorithm has made better use of its queries to approach the global optimum efficiently.

2.3.6.3 Comparison and Relevance

Simple regret and cumulative regret capture different trade-offs in BO. Simple regret focuses solely on the quality of the final solution and is particularly suited for offline optimization problems, where the cost of intermediate evaluations is less critical. In contrast, cumulative regret is more relevant in settings where every evaluation incurs a cost, such as online optimization, and emphasizes the efficiency of the entire optimization process.

Both metrics are essential for evaluating the performance of BO algorithms, offering complementary insights into their behavior and effectiveness across various applications.

2.4 Black-box Optimization with Unknown Black-box Constraints

As real-world problems often involve constraints that are also black-box in nature, Constrained Black-box Optimization (CBO) has become a vital extension of BO. CBO methods adjust the acquisition function to account for these constraints, seeking feasible solutions that satisfy the conditions while optimizing the objective function. A prominent method in CBO is the Expected Improvement with Constraints (cEI), first introduced by (Schonlau, Welch, and Jones, 1998) and later extended by (Gardner et al., 2014) and (Gelbart, Snoek, and Adams, 2014). cEI integrates feasibility into the acquisition function, directing the optimization process toward regions where feasible solutions are likely. Letham et al. (2019) further improved cEI by using a quasi-Monte Carlo approximation to better manage observation noise, enhancing its effectiveness in noisy environments.

EI-based methods for constrained optimization face several challenges. When no feasible point exists, the EI cannot be computed, leading to modifications that focus solely on finding feasible regions, ignoring the objective function. Additionally, numerical challenges further limit some methods like EVR and IECI to small-dimensional problems. To address this, alternative methods have been proposed. For example, Predictive Entropy Search with Constraints (PESC, Hernández-Lobato et al., 2015) offers a heuristic approach that selects feasible candidates directly from the search space, reducing uncertainty more effectively. However, the computational challenges associated with quadrature calculations during sampling have limited its practical applicability. Recently, Takeno et al. (2022) proposed a Min-Value Entropy Search method that simplifies the sampling process, making it more tractable.

Numerical optimization has also been taken into consideration as an effective tool for solving the unknown constraint problem. The idea is to reformulate constraints into simpler unconstrained problems solved through alternating iterations. The Augmented Lagrangian method is mostly used in this category. For example, Gramacy et al. (2016) with Augmented Lagrangian Bayesian Optimization (ALBO) and its improvement Slack-AL (Picheny et al., 2016) use Augmented Lagrangian Function (ALF) to formulate unconstrained surrogate problems and then solve them using EI as an acquisition function. Recently, ADMMBO (Ariafar et al., 2019) first

applied the ADMM technique to transform the constrained problem into an equivalent unconstrained optimization, then solved an augmented Lagrangian relaxation. However, this method requires the introduction of additional variables, leading to increased computational costs.

Recent research has explored penalty functions and primal-dual methods to handle constraint violations during optimization. For example, Lu and Paulson (2022) introduced a penalty-function approach that adds a penalty term for constraint violations to the objective function, transforming the constrained problem into an unconstrained one. Similarly, (Zhou and Ji, 2022) proposed a primal-dual approach that balances the trade-off between optimizing the objective and minimizing constraint violations. While these methods are promising, their effectiveness is sensitive to the choice of parameters set, often requiring considerable effort in parameter tuning during implementation.

Alongside empirical advancements, recent theoretical works have started to address the absence of formal guarantees in Constrained Black-box Optimization (CBO). For example, Lu and Paulson (2022) introduced a penalty-based regret bound that combines the regret from the objective function with penalties for constraint violations. Xu et al. (2023) expanded this analysis by separately evaluating cumulative regret and constraint violations. In contrast, Nguyen et al. (2023) provided a theoretical performance guarantee for CBO under unknown constraints in a *decoupled* setting, where cumulative regret is calculated as the sum of both objective function regret and constraint violations.

2.5 Black-box Optimization with Physical Information

In the realm of objective functions encountered in scientific and engineering domains, many are governed by Partial Differential Equations (PDEs). These equations encapsulate the fundamental laws of physics that describe how systems evolve over time and space. For example, the heat equations describe the distribution of heat in a given space over time, the Navier-Stokes equations describe how the velocity, pressure, and density of a fluid change over time and space. They take into account factors such as the viscosity (resistance to flow) of the fluid and external forces acting upon it. Furthermore, physical laws being implied in PDE can also be found in structural analysis or electromagnetics, to name a few. Notably, recent efforts have been made to integrate PDE knowledge into objective function models. Raissi, Perdikaris, and Karniadakis (2017)introduced a new method using Gaussian Processes Regression (GPR) with a unique four-block covariance kernel, enabling the utilization of observations from both the objective function and the PDEs. Another approach, as described in Jidling et al., 2017, proposes a specialized covariance kernel to constrain Gaussian Processes (GPs) using differential equations. Unlike the method in Raissi, Perdikaris, and Karniadakis, 2017, this approach enforces the constraint globally instead of relying on specific data points. This not only provides a stronger constraint but also eliminates the computational burden associated with the four-block covariance matrix. For more details, see Swiler et al., 2020. Recently, Chen et al., 2021 proposed a numerical algorithm to approximate the solution of a given non-linear PDE as a Maximum a Posteriori (MAP) estimator of a GP conditioned on a finite set of data points from the PDE. Remarkably, their approach offers guaranteed convergence for a broad and inclusive class of PDEs. Despite the promising potential of GPs in solving PDEs, GPs have a limitation as their computational scalability is a

critical problem. The kernel matrix inversion when updating the posterior of a GP exhibits cubic complexity in the number of data points.

Recently, the Physics-Informed Neural Network (PINN) was introduced Raissi, Perdikaris, and Karniadakis, 2019; Yang, Meng, and Karniadakis, 2021, which offers a viable approach for tackling general PDEs. Unlike GP, PINN leverages the expressive power of neural networks to approximate complex, nonlinear relationships within the data. This inherent flexibility enables PINN to handle a broader range of PDEs including nonlinear PDEs, making them well-suited for diverse scientific and engineering applications. Further, there is research providing a deeper insight into the theoretical aspect of incorporating the PDEs using PINN Schiassi et al., 2021; Wang et al., 2022, and analyzing the connection between GP and PINN models to learn the underlying functions that satisfy PDE equations Wang, Yu, and Perdikaris, 2022.

As the PDEs hold promise as a valuable source of information, they could greatly improve the modeling of the black-box function and thus help towards sample-efficient optimization, reducing the number of function evaluations. It is worth noting that there has been relatively little exploration of how PDEs can be leveraged in the context of black-box optimization settings, where the objective function is treated as an unknown and potentially noisy function. In this paper, we consider a global optimization problem setting where the objective function $f\colon \mathcal{D} \to \mathbb{R}$ is associated with a PDE:

$$\min_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x}) \text{ s.t. } \mathcal{N}[f](\mathbf{x}) = g(\mathbf{x}),$$

where $\mathcal{D} \subset \mathbb{R}^d$ is a d-dimensional bounded domain and $\mathcal{N}[f]$ denotes a differential operator of the function f with respect to the input $\mathbf{x}$. The function f is an expensive, black-box function, and its evaluations are obtainable only through noisy measurements in the form of $y = f(\mathbf{x}) + \epsilon$, where ϵ represents sub-Gaussian noise, as elaborated later in Section **??**. Additionally, the function $g(\mathbf{x})$ is a cheap-to-evaluate function, which may also involve noise, with respect to the PDE-constraint. For example, consider the damped harmonic oscillator from classical physics as a real-world example where $f(x)$ is the displacement of the oscillator as a black-box function of time x, and measuring the displacement of a damped harmonic oscillator accurately can be expensive due to several factors, e.g., instrumentation cost, environmental factors, and calibration requirements. This displacement is governed by the PDE: $m\frac{d^2f}{dx^2} + c\frac{df}{dx} + kf = 0$, where $\mathcal{N}[f](x) = m\frac{d^2f}{dx^2} + c\frac{df}{dx} + kf$, $g(x) = 0$, and m, c, k are mass, damping and spring values, respectively. Furthermore, it is assumed that the boundary conditions of the PDEs are either unknown or inaccessible. These assumptions widely hold in many problem settings. As an example, Cai et al., 2020 examines a two-dimensional heat transfer problem with forced heat convection around a circular cylinder. The heat measurement entails high costs due to the material, size, and shape of the system. The problem has known incompressible Navier-Stokes and heat transfer equations. However, the thermal boundary conditions are difficult to ascertain precisely because of the complex and large instruments. The unavailability of these boundary conditions prevents the straightforward solution of the underlying function f using traditional numerical techniques.

2.6 Further Research in Black-box Optimisation

Chapter 3

Neural-BO: A Black-box Optimization Algorithm using Deep Neural Networks

Gaussian Processes (GPs) are a popular probabilistic model for BO. However, they are not well-suited for optimizing functions in complex, structured, high-dimensional spaces, such as those encountered in the optimization of images and text documents. This limitation arises because GPs rely on kernels to model functions, and widely used kernels, such as the Square Exponential and Matérn kernels, struggle to accurately capture similarities within complex structured data. Consequently, BO using GPs is limited in its applicability to fields like computer vision and natural language processing. Moreover, GPs face significant scalability challenges due to the cubic computational complexity of kernel inversion.

Deep Neural Networks (DNNs), by contrast, have demonstrated exceptional scalability and generalization capabilities, particularly for high-dimensional and structured data. These qualities make them a potential alternative to GPs for optimization tasks. Recent efforts to leverage DNNs in black-box function optimization and contextual bandits have shown promise. However, many existing approaches depend on computationally expensive Bayesian Neural Networks (BNNs), which require sampling-based methods to model predictive uncertainty, or they lack theoretical guarantees for convergence and sample efficiency.

To address these challenges, this chapter introduces Neural-BO, a novel framework for black-box optimization. In Neural-BO, the unknown function is modeled using an over-parameterized DNN, eliminating the need for a *Bayesian Neural Network (BNN)* and making the algorithm computationally tractable. The framework leverages recent advances in neural network theory, particularly Neural Tangent Kernel (NTK), to estimate confidence intervals around the neural network model. These confidence intervals are used within Thompson Sampling to draw random samples of the function, which are then maximized to determine the next evaluation point. We analyze the theoretical properties of Neural-BO, providing a regret bound that establishes its convergence and improved sample efficiency compared to existing methods. We summary **the contributions of this chapter** as follows:

- A deep neural network based black-box optimization (Neural-BO) that can perform efficient optimization by accurately modeling the complex structured data (e.g. image, text, etc) and also being computationally favorable, i.e. scaling linearly with the number of data points.
- A theoretical analysis of our proposed Neural-BO algorithm showing that our algorithm is convergent with a sublinear regret bound.

- Experiments with both synthetic benchmark optimization functions and real-world optimization tasks showing that our algorithm outperforms current state-of-the-art black-box optimization methods.

This chapter begins with an outline of the problem setting and assumptions, followed by a detailed description of the Neural-BO framework and its theoretical analysis.

3.1 Problem Setting

We consider a global optimization problem to maximize $f(\mathbf{x})$ subject to $\mathbf{x} \in \mathcal{D} \subset \mathbb{R}^d$, where $\mathbf{x}$ is the input with d dimensions. The function f is a noisy black-box function that can only be evaluated via noisy evaluations of f in the form $y = f(x) + \epsilon$, where ϵ is a sub-Gaussian noise which we will discuss more clearly in Assumption 2 in our regret analysis section. In our work, we consider input space $\mathcal{D}$ of the form: $a \leq \|\mathbf{x}\|_2 \leq b$, where a, b are absolute constants and $a \leq b$. We note that this assumption is mild compared to current works in neural contextual bandits (Zhou, Li, and Gu, 2020; Zhang et al., 2021; Xu et al., 2020; Kassraie and Krause, 2022) that required $\|\mathbf{x}\|_2 = 1$.

3.2 Proposed Neural-BO Method

In this section, we present our proposed DNN-based black-box algorithm, Neural-BO. Following the principle of BO, our algorithm consists of two main steps: (1) build a model of the black-box objective function, and (2) use the black-box model to choose the next function evaluation point at each iteration. For the first step, unlike traditional BO algorithms which use GP to model the objective function, our idea is to use a fully connected neural network $h(\mathbf{x}; \boldsymbol{\theta})$ to learn the function f as follows:

$$h(\mathbf{x}; \boldsymbol{\theta}) = \sqrt{m}\mathbf{W}_L \phi(\mathbf{W}_{L-1}\phi(\cdots \phi(\mathbf{W}_1 \mathbf{x})),$$

where $\phi(x) = \max(x, 0)$ is the Rectified Linear Unit (ReLU) activation function, $\mathbf{W}_1 \in \mathbb{R}^{m\times d}, \mathbf{W}_i \in \mathbb{R}^{m\times m}, 2 \leq i \leq L-1, \mathbf{W}_L \in \mathbb{R}^{1\times m}$, and $\boldsymbol{\theta} = (vec(\mathbf{W}_1), \cdots, vec(\mathbf{W}_L)) \in \mathbb{R}^p$ is the collection of parameters of the neural network, $p = md + m^2(L-2) + m$. To make the analysis tractable, we assume that the width m is the same for all hidden layers. We also denote the gradient of the neural network by $\mathbf{g}(\mathbf{x}; \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} h(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}^p$.

For the second step, we choose the next sample point $\mathbf{x}_t$ by using a Thompson Sampling strategy. In particular, given each $\mathbf{x}$, our algorithm maintains a Gaussian distribution for $f(\mathbf{x})$. To choose the next point, it samples a random function $\widetilde{f}_t$ from the posterior distribution $\mathcal{N}(h(\mathbf{x}, \boldsymbol{\theta}_{t-1}), \nu_t^2 \sigma_t^2(\mathbf{x}))$, where

$$\sigma_t^2(\mathbf{x}) = \lambda \mathbf{g}(\mathbf{x}; \boldsymbol{\theta}_0) \mathbf{U}_{t-1}^{-1} \mathbf{g}(\mathbf{x}; \boldsymbol{\theta}_0)$$
$$\mathbf{U}_{t-1} = \sum_{i=0}^{t-1} \mathbf{g}(\mathbf{x}_i; \boldsymbol{\theta}_0) \mathbf{g}(\mathbf{x}_i; \boldsymbol{\theta}_0)^\top / m,$$

and $\nu_t = \sqrt{2}B + \frac{R}{\sqrt{\lambda}}\sqrt{2\log(1/\delta)}$. From here, $\sigma_t(\mathbf{x})$ and ν_t construct a confidence interval, where for all $\mathbf{x} \in \mathcal{D}$, we have $|f(\mathbf{x}) - f(\mathbf{x}, \boldsymbol{\theta})| \leq \nu_t \sigma_t(\mathbf{x})$ that holds with probability $1 - \delta$. We also note the difference from Zhang et al. (2021) that uses $\mathbf{g}(\mathbf{x}; \boldsymbol{\theta}_t)/\sqrt{m}$ as dynamic feature map, we here use a fixed feature map $\mathbf{g}(\mathbf{x}; \boldsymbol{\theta}_0)/\sqrt{m}$

which can be viewed as a finite approximation of the feature map $\phi(\mathbf{x})$ of k_{NTK}. This allows us to eliminate $\sqrt{m}$ away the regret bound of our algorithm. Then, the next evaluation is selected as $\mathbf{x}_t = \text{argmax}_{\mathbf{x}\in\mathcal{D}} \widetilde{f}_t(\mathbf{x})$.

Besides, to initialize the network $h(\mathbf{x};\boldsymbol{\theta}_0)$, we randomly generate each element of $\boldsymbol{\theta}_0$ from an appropriate Gaussian distribution: for each $1 \leq l \leq L-1$, each entry of $\mathbf{W}$ is generated independently from $\mathcal{N}(0, 2/m)$; while each entry of the last layer $\mathbf{W}_L$ is set to zero to make $h(\mathbf{x}, \boldsymbol{\theta}_0) = 0$. Here we inherit so-called *He initialization* (He et al., 2015), which ensures the stability of the expected length of the output vector at each hidden layer.

The proposed Neural-BO is summarized using Algorithm 1 and Algorithm 2. In section 3.3, we will demonstrate that our algorithm can achieve a sub-linear regret bound, and works well in practice (See our section 3.5).

Discussion A simple way to have a DNN-based black-box algorithm is to extend the existing works of Zhou, Li, and Gu (2020) and Zhang et al. (2021) or Kassraie and Krause (2022) to our setting where the search space is continuous. However, this is difficult because these algorithms are designed for a finite set of actions. A simple adaptation can yield non-vanishing bounds on cumulative regret. For example, the regret bound in Kassraie and Krause (2022) depends on the size of finite action spaces $|\mathcal{A}|$. If $|\mathcal{A}| \to \infty$ then the regret bound goes to infinity. In Zhou, Li, and Gu (2020) and Zhang et al. (2021), the regret bounds are built on gram matrix $\mathbf{H}$ which is defined through the set of actions. However, such a matrix cannot even be defined in our setting where the search space (action space) is infinite. We solve this challenge by using a discretization of the search space at each iteration as mentioned in Section 3.3. In addition, for our confidence bound calculation, using $\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)/\sqrt{m}$ instead of $\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_t)/\sqrt{m}$ as in Zhou, Li, and Gu (2020) and Zhang et al. (2021) allows us to reduce a factor $\sqrt{m}$ from our regret bound.

Our algorithm is also different from Chowdhury and Gopalan (2017) in traditional Bayesian optimization which uses a Gaussian process to model the function with posterior computed in closed form. In contrast, our algorithm computes posterior by gradient descent.

Algorithm 1 Neural Black-box Optimization (Neural-BO)

Input: The input space $\mathcal{D}$, the number of rounds T, exploration parameter ν_t, network width m, RKHS norm bound B, regularization parameter λ, $\delta \in (0, 1)$.

Set $\mathbf{U}_0 = \lambda\mathbf{I}$
for $t = 1$ to T **do**
  $\sigma_t^2(\mathbf{x}) = \lambda\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top\mathbf{U}_{t-1}^{-1}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)/m$
  $\widetilde{f}_t(\mathbf{x}) \sim \mathcal{N}(h(\mathbf{x};\boldsymbol{\theta}_{t-1}), \nu_t^2\sigma_t^2(\mathbf{x}))$
  Choose $\mathbf{x}_t = \text{argmax}_{\mathbf{x}\in\mathcal{D}} \widetilde{f}_t(\mathbf{x})$ and receive observation $y_t = f(\mathbf{x}_t) + \epsilon_t$
  Update $\boldsymbol{\theta}_t \leftarrow \text{TrainNN}(\{\mathbf{x}_i\}_{i=1}^t\{y_i\}_{i=1}^t, \boldsymbol{\theta}_0)$
  $\mathbf{U}_t \leftarrow \mathbf{U}_{t-1} + \frac{\mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0)\mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0)^\top}{m}$
end for

Algorithm 2 TrainNN

Input: Chosen points $\{\mathbf{x}_i\}_{i=1}^t$, noisy observations $\{y_i\}_{i=1}^t$, initial parameter $\boldsymbol{\theta}_0$, number of gradient descent steps J, regularization parameter λ, network width m, learning rate η.

1: Define objective function $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{T}(f(\mathbf{x}_i;\boldsymbol{\theta}) - y_i)^2 + \frac{1}{2}m\lambda\left\|\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}\right\|_2^2$
2: **for** $k = 0, \cdots, J-1$ **do**
$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta\nabla\mathcal{L}(\boldsymbol{\theta}^{(k)})$
3: **end for**
4: return $\boldsymbol{\theta}^{(J)}$.

3.3 Theoretical Analysis

In this section, we provide a regret bound for the proposed Neural-BO algorithm. Our regret analysis is built upon the recent advances in NTK theory (Allen-Zhu, Li, and Song, 2019; Cao and Gu, 2019) and proof techniques of GP-TS (Chowdhury and Gopalan, 2017). Unlike most previous neural-based contextual bandits works (Zhang et al., 2021; Zhou, Li, and Gu, 2020) that restrict necessarily the search space to a set $\mathbb{S}^{d-1} = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|_2 = 1\}$, we here derive our results for a flexible condition, where the norm of inputs are bounded: $a \leq \|\mathbf{x}\|_2 \leq b$, where $0 < a \leq b$.

For theoretical guarantees, we need to use the following definition and assumptions on the observations $\mathcal{D}_t = \{\mathbf{x}_i, y_i\}_{i=1}^T$.

Definition 3.3.1 (Jacot, Gabriel, and Hongler, 2018)**.** Consider the same set $\mathcal{D}_t$ as above and define

$$\widetilde{\mathbf{H}}_{i,j}^{(1)} = \boldsymbol{\Sigma}_{i,j}^{(1)} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \,, \mathbf{A}_{i,j}^{(l)} = \begin{pmatrix} \boldsymbol{\Sigma}_{i,i}^{(l)} & \boldsymbol{\Sigma}_{i,j}^{(l)} \\ \boldsymbol{\Sigma}_{i,j}^{(l)} & \boldsymbol{\Sigma}_{j,j}^{(l)} \end{pmatrix}$$

$$\boldsymbol{\Sigma}_{i,j}^{(l+1)} = 2\mathbb{E}_{(u,v)\sim\mathbf{N}(\mathbf{0},\mathbf{A}_{i,j}^{(l)})}[\phi(u)\phi(v)]$$

$$\widetilde{\mathbf{H}}_{i,j}^{(l+1)} = 2\widetilde{\mathbf{H}}_{i,j}^{(l)}\mathbb{E}_{(u,v)\sim\mathbf{N}(\mathbf{0},\mathbf{A}_{i,j}^{(l)})}[\phi'(u)\phi'(v)] + \boldsymbol{\Sigma}_{i,j}^{(l+1)},$$

where $\phi(\cdot)$ is the activation function of the neural network $h(\mathbf{x};\boldsymbol{\theta})$. Then $\mathbf{H}_t = \frac{\widetilde{\mathbf{H}}^{(L)} + \boldsymbol{\Sigma}^{(L)}}{2}$ is called the NTK matrix on the set $\mathcal{D}_t$.

Assumption 3.3.2. There exists $\lambda_0 > 0$, $\mathbf{H}_T \geq \lambda_0 \mathbf{I}_T$.

This is a common assumption in the literature (Zhou, Li, and Gu, 2020; Xu et al., 2020; Zhang et al., 2021), and is satisfied as long as we sufficiently explore the input space such that no two inputs $\mathbf{x}$ and $\mathbf{x}'$ are identical.

Assumption 3.3.3. We assume the objective function f is from an RKHS (See Section 2.1.5) corresponding to a positive definite NTK k_{NTK}. In particular, $\|f\|_{\mathcal{H}_{k_{\text{NTK}}}} \leq B$, for a finite $B > 0$. These assumptions indicate the smoothness of function f and are regular as they have been typically used in many previous works (Chowdhury and Gopalan, 2017; Srinivas et al., 2009; Vakili, Khezeli, and Picheny, 2021).

Assumption 3.3.4. We assume the noises $\{\epsilon_i\}_{i=1}^T$ where $\epsilon_i = y_i - f(\mathbf{x}_i)$, are i.i.d and sub-Gaussian with parameter $R > 0$ and are independent with $\{\mathbf{x}_i\}_{i=1}^T$.

This assumption is mild and similar to the work of Vakili et al. (2021) where the assumption is used to prove an optimal order of the regret bound for Gaussian process bandits. We use it here to provide a regret bound with the sublinear order for our proposed algorithm.

Now we are ready to bound the regret of our proposed Neural-BO algorithm. To measure the regret of the algorithm, we use the cumulative regret which is defined as follows: $R_T = \sum_{t=1}^{T} r_t$ after T iterations, where $\mathbf{x}^* = \text{argmax}_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x})$ is the maximum point of the unknown function f and $r_t = f(\mathbf{x}^*) - f(\mathbf{x}_t)$ is instantaneous regret incurred at time t. For further details on cumulative regret and its role as a performance metric, refer to Section 2.3.6. We present our main theoretical result.

Theorem 3.3.5. *Let $\delta \in (0,1)$. Assume that the true underlying f lies in the RKHS $\mathcal{H}_{k_{\text{NTK}}}(\mathcal{D})$ corresponding to the NTK kernel $k_{\text{NTK}}(\mathbf{x},\mathbf{x}')$ with RKHS norm bounded by B. Set the parameters in Algorithm 1 as $\lambda = 1 + \frac{1}{T}$, $\nu_t = \sqrt{2}B + \frac{R}{\sqrt{\lambda}}\sqrt{2\log(1/\delta)}$, where R is the noise sub-Gaussianity parameter. Set $\eta = (m\lambda + mLT)^{-1}$ and $J = (1 + LT/\lambda)\left(1 + \log\left(T^3 L\lambda^{-1}\log(1/\delta)\right)\right)$. If the network width m satisfies:*

$$m \geq \text{poly}\left(\lambda, T, \log(1/\delta), \lambda_0^{-1}\right),$$

then with probability at least $1-\delta$, the regret of Algorithm 1 is bounded as

$$R_T \leq C_1(1+c_T)\nu_T\sqrt{L}\sqrt{\frac{\lambda BT}{\log(B+1)}(2\gamma_T+1)} + (4 + C_2(1+c_T)\nu_T L)\sqrt{2\log(1/\delta)T} + \frac{(2B+1)\pi^2}{6} + \frac{b(a^2+b^2)}{a^3}$$

where C_1, C_2 are absolute constants, a and b are lower and upper norm bounds of $\mathbf{x} \in \mathcal{D}$ and $c_T = \sqrt{4\log T + 2\log|\mathcal{D}_t|}$.

Remark 3.3.6. There exists a component $\frac{b(a^2+b^2)}{a^3}$ in the regret bound R_T. This component comes from the condition of the search space. If we remove the influence of constants a, b, B and R, Theorem 3.3.5 implies that our regret bound follows an order $\widetilde{O}(\sqrt{T\gamma_T})$. This regret bound has the same order as the one of the Maximum Variance Reduction algorithm in Vakili et al. (2021), but is significantly better than traditional BO algorithms (e.g., (Srinivas et al., 2009; Chowdhury and Gopalan, 2017)) and existing neural contextual bandits (e.g., (Zhou, Li, and Gu, 2020; Zhang et al., 2021)) that have the order $\widetilde{O}(\gamma_T\sqrt{T})$. We note that the regret bounds in Zhou, Li, and Gu (2020) and Zhang et al. (2021) are expressed through the effective dimension $\widetilde{d} = \frac{\log\det(\mathbf{I}+\mathbf{H}/\lambda)}{\log(1+TK/\lambda)}$, where K is the number of arm in contextual bandits setting. However, γ_T and the effective dimension are of the same order up to a ratio of $1/(2\log(1+TK/\lambda))$. The improvement of factor $\sqrt{\gamma_T}$ is significant because for NTK kernel as shown by Kassraie and Krause (2022), γ_T follows order $\mathcal{O}(T^{1-1/d})$, where d is the input dimension. For $d \geq 2$, existing bounds become vacuous and are no longer sub-linear. In contrast, our regret bound remains sub-linear for any $d > 0$.

3.4 Proof of the Main Theorem

To perform analysis for continuous actions as in our setting, we use a discretization technique. At each time t, we use a discretization $\mathcal{D}_t \subset \mathcal{D}$ with the property that

$|f(\mathbf{x}) - f([\mathbf{x}]_t)| \leq \frac{1}{t^2}$ where $[\mathbf{x}]_t \in \mathcal{D}_t$ is the closest point to $\mathbf{x} \in \mathcal{D}$. Then we choose $\mathcal{D}_t$ with size $|\mathcal{D}_t| = \left((b-a)BC_{\text{lip}}t^2\right)^d$ that satisfies $\|\mathbf{x} - [\mathbf{x}]_t\|_1 \leq \frac{b-a}{(b-a)BC_{\text{lip}}t^2} = \frac{1}{BC_{\text{lip}}t^2}$ for all $\mathbf{x} \in \mathcal{D}$, where $C_{\text{lip}} = \sup_{\mathbf{x}\in\mathcal{D}} \sup_{j\in[d]} \left(\frac{\partial^2 k_{\text{NTK}}(\mathbf{p},\mathbf{q})}{\partial \mathbf{p}_j \mathbf{q}_j} | \mathbf{p} = \mathbf{q} = \mathbf{x}\right)$. This implies, for all $\mathbf{x} \in \mathcal{D}$:

$$|f(\mathbf{x}) - f([\mathbf{x}]_t)| \leq \|f\|_{\mathcal{H}_{k_{\text{NTK}}}} C_{\text{lip}} \|\mathbf{x} - [\mathbf{x}]_t\|_1 \leq BC_{\text{lip}} \frac{1}{BC_{\text{lip}}t^2} = 1/t^2, \tag{3.1}$$

is Lipschitz continuous of any $f \in \mathcal{H}_{k_{\text{NTK}}}(\mathcal{D})$ with Lipschitz constant BC_{lip}, where we have used our assumption $\|f\|_{\mathcal{H}_{k_{\text{NTK}}}} \leq B$. We bound the regret of the proposed algorithm by starting from the instantaneous regret $r_t = f(\mathbf{x}^*) - f(\mathbf{x}_t)$ can be decomposed to $r_t = [f(\mathbf{x}^*) - f([\mathbf{x}^*]_t)] + [f([\mathbf{x}^*]_t - f(\mathbf{x}_t)]$. While the first term is bounded by Eqn. (3.1), we bound the second term in Lemma A.1.9 provided in our Appendix. The proof of Lemma A.1.9 requires a few steps, namely (a) introduce a saturated set $\mathcal{S}_t$ (see Definition A.2), then combine it with the results of Lemma A.1.3 (deriving a bound on $|\widetilde{f}_t(\mathbf{x}) - h(\mathbf{x}; \boldsymbol{\theta}_{t-1})|$) necessitated due to the Thompson sampling and Lemma A.1.4 (deriving a bound on $|f(\mathbf{x}) - h(\mathbf{x}; \boldsymbol{\theta}_{t-1})|$) utilizing the generalization result of the over-parameterized neural networks. Note that, in our proof, we consider the effect of our general search space setting (where input $\mathbf{x} \in \mathcal{D}$ is norm-bounded, i.e., $0 < a \leq \|\mathbf{x}\|_2 \leq b$) on the output of the over-parameterized deep neural network at each layer in Lemma A.1.2 and on the gradient of this network utilized in Lemma A.1.5.1. These results contribute to the appearance of the lower and the upper norm bounds a and b, the first in the generalization property of over-parameterized neural networks in Lemma A.1.4 and the latter affect the cumulative regret bound in Theorem 3.3.5. Our proof follows the proof style of Lemma 4.1 of Cao and Gu (2019), Lemma B.3 of Cao and Gu (2019) and Theorem 5 of Allen-Zhu, Li, and Song (2019).

On the other hand, Lemma A.1.5.2, based on Assumption 3.3.4, provides a tighter bound on the confidence interval that eliminated the factor $\sqrt{\gamma_T}$, in comparison with previous relevant works Chowdhury and Gopalan (2017) and Zhou, Li, and Gu (2020), leading to the sub-linear regret bound in Theorem 3.3.5. By these arguments, we achieve a cumulative regret bound for our proposed algorithm in terms of the maximum information gain γ_T (Lemma A.1.8, A.1.9, A.1.10). **Our detailed proof is provided in Appendix A.1**.

Proof Sketch for Theorem 3.3.5. With probability at least $1-\delta$, we have

$$\begin{aligned}
R_T &= \sum_{t=1}^{T} f(\mathbf{x}^*) - f(\mathbf{x}_t) \\
&= \sum_{t=1}^{T} [f(\mathbf{x}^*) - f([\mathbf{x}^*]_t)] + [f([\mathbf{x}^*]_t) - f(\mathbf{x}_t)] \\
&\leq 4T\epsilon(m) + \frac{(2B+1)\pi^2}{6} + \bar{C}_1(1+c_T)\nu_T\sqrt{L}\sum_{i=1}^{T}\min(\sigma_t(\mathbf{x}_t), B) \\
&\quad + (4 + \bar{C}_2(1+c_T)\nu_T L + 4\epsilon(m))\sqrt{2\log(1/\delta)T} \\
&\leq \bar{C}_1(1+c_T)\nu_T\sqrt{L}\sqrt{\frac{\lambda BT}{\log(B+1)}(2\gamma_T+1)} + \frac{(2B+1)\pi^2}{6} + 4T\epsilon(m) \\
&\quad + 4\epsilon(m)\sqrt{2\log(1/\delta)T} + (4 + \bar{C}_2(1+c_T)\nu_T L)\sqrt{2\log(1/\delta)T} \\
&= \bar{C}_1(1+c_T)\nu_t\sqrt{L}\sqrt{\frac{\lambda BT}{\log(B+1)}(2\gamma_T+1)} + \frac{(2B+1)\pi^2}{6} \\
&\quad + \epsilon(m)(4T + \sqrt{2\log(1/\delta)T}) + (4 + \bar{C}_2(1+c_T)\nu_t L)\sqrt{2\log(1/\delta)T}
\end{aligned}$$

The first inequality is due to Lemma A.1.9, which provides the bound for cumulative regret R_T in terms of $\sum_{t=1}^{T}\min(\sigma_t(\mathbf{x}_t), B)$. The second inequality further provides the bound of term $\sum_{t=1}^{T}\min(\sigma_t(\mathbf{x}_t), B)$ due to Lemma A.1.10, while the last equality rearranges addition. Picking

$$\begin{aligned}
\eta &= (m\lambda + mLT)^{-1} \text{ and} \\
J &= (1 + LT/\lambda)\left(\log(C_{\epsilon,2}) + \log\left(T^3 L\lambda^{-1}\log(1/\delta)\right)\right),
\end{aligned}$$

we have

$$\begin{aligned}
&\frac{b}{a}C_{\epsilon,2}(1-\eta m\lambda)^J\sqrt{TL/\lambda}\left(4T + \sqrt{2\log(1/\delta)T}\right) \\
=&\frac{b}{a}C_{\epsilon,2}\left(1 - \frac{1}{1+LT/\lambda}\right)^J\left(4T + \sqrt{2\log(1/\delta)T}\right) \\
=&\frac{b}{a}C_{\epsilon,2}e^{-\left(\log(C_{\epsilon,2}) + \log\left(T^3 L\lambda^{-1}\log(1/\delta)\right)\right)}\left(4T + \sqrt{2\log(1/\delta)T}\right) \\
=&\frac{b}{a}\frac{1}{C_{\epsilon,2}}.T^{-3}L^{-1}\lambda\log^{-1}(1/\delta)\left(4T + \sqrt{2\log(1/\delta)T}\right) \leq \frac{b}{a}
\end{aligned}$$

Then choosing m that satisfies:

$$\left(\frac{b}{a}C_{\epsilon,1}m^{-1/6}\lambda^{-2/3}L^3\sqrt{\log m} + \left(\frac{b}{a}\right)^3 C_{\epsilon,3}m^{-1/6}\sqrt{\log m}L^4T^{5/3}\lambda^{-5/3}(1+\sqrt{T/\lambda})\right)$$
$$\left(4T + \sqrt{2\log(1/\delta)T}\right) \leq \left(\frac{b}{a}\right)^3$$

We finally achieve the bound of R_T as:

$$R_T \leq \bar{C}_1(1+c_T)\nu_T\sqrt{L}\sqrt{\frac{\lambda BT}{\log(B+1)}(2\gamma_T+1)} + (4+\bar{C}_2(1+c_T)\nu_T L)\sqrt{2\log(1/\delta)T} + \frac{(2B+1)\pi^2}{6} + \frac{b(a^2+b^2)}{a^3}$$

□

3.5 Experiments

In this section, we demonstrate the effectiveness of our proposed Neural-BO algorithm compared to traditional BO algorithms and several other BO algorithms on various synthetic benchmark optimization functions and various real-world optimization problems that have complex and structured inputs. The real-world optimization problems consist of (1) generating sensitive samples to detect tampered model trained on MNIST (LeCun and Cortes, 2010) dataset; (2) optimizing control parameters for robot pushing considered in Wang and Jegelka (2017); and (3) optimizing the number of evaluations needed to find a document that resembles an intended (unknown) target document.

3.5.1 Experimental Setup

For all experiments, we compared our algorithm with common classes of surrogate models used in black-box optimization, including GPs, RFs and DNNs. For GPs, we have three baselines: GP-UCB (Srinivas et al., 2009), GP-TS (Chowdhury and Gopalan, 2017) and GP-EI (Jones, Schonlau, and Welch, 1998) with the common Squared Exponential (SE) kernel. Our implementations for GP-based Bayesian Optimization baselines utilize public library GPyTorch [1] and BOTorch [2]. Next, we consider SMAC (Hutter, Hoos, and Leyton-Brown, 2011) as a baseline using RF as a surrogate model. Lastly, we also include two recent DNNs-based works for black-box optimization: DNGO (Snoek et al., 2015) and Neural Greedy (Paria et al., 2022). We further describe the implementations for RF and DNN-based baselines as follows:

- RF-based surrogate model is implemented using public library GPyOpt [3] and optimization is performed with EI acquisition function.
- DNGO (Snoek et al., 2015) models the black-box function by a Bayesian linear regressor, using low-dimensional features (extracted by DNN) as inputs. We run DNGO algorithm with the implementation of AutoML [4] with default settings.
- NeuralGreedy (Paria et al., 2022) fits a neural network to the current set of observations, where the function values are randomly perturbed before learning the neural network. The learned neural network is then used as the acquisition function to determine the next query point. Our re-implementation of NeuralGreedy follows the setting described in Appendix F.2 (Paria et al., 2022).

[1] https://gpytorch.ai/
[2] https://botorch.org/
[3] https://github.com/SheffieldML/GPyOpt
[4] https://github.com/automl/pybnn

For our proposed Neural-BO algorithm, we model the functions using two-layer neural networks $f(\mathbf{x};\boldsymbol{\theta}) = \sqrt{m}\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x})$ with network width $m = 500$. The weights of the network are initialized with independent samples from a normal distribution $\mathcal{N}(0, 1/m)$. To train the surrogate neural network models, we use SGD optimizer with batch size 50, epochs 50, learning rate $\eta = 0.001$ and regularizer $\lambda = 0.01$. We set exploration parameter in Algorithm 1: $\nu_t = \nu$ and do a grid search over $\{0.1, 1, 10\}$.

3.5.2 Synthetic Benchmarks

We optimized several commonly used synthetic benchmark functions: Ackley, Levy, and Michalewics. The exact expression of these functions can be found at: `http://www.sfu.ca/~ssurjano/optimization.html`. To emphasize the adaptation of our method to various dimensions, for each function, we performed optimization for four different dimensions, including 10, 20, 50, and 100. This is done to evaluate the methods across a range of dimensions. The function evaluation noise follows a normal distribution with zero mean and the variance is set to 1% of the function range.

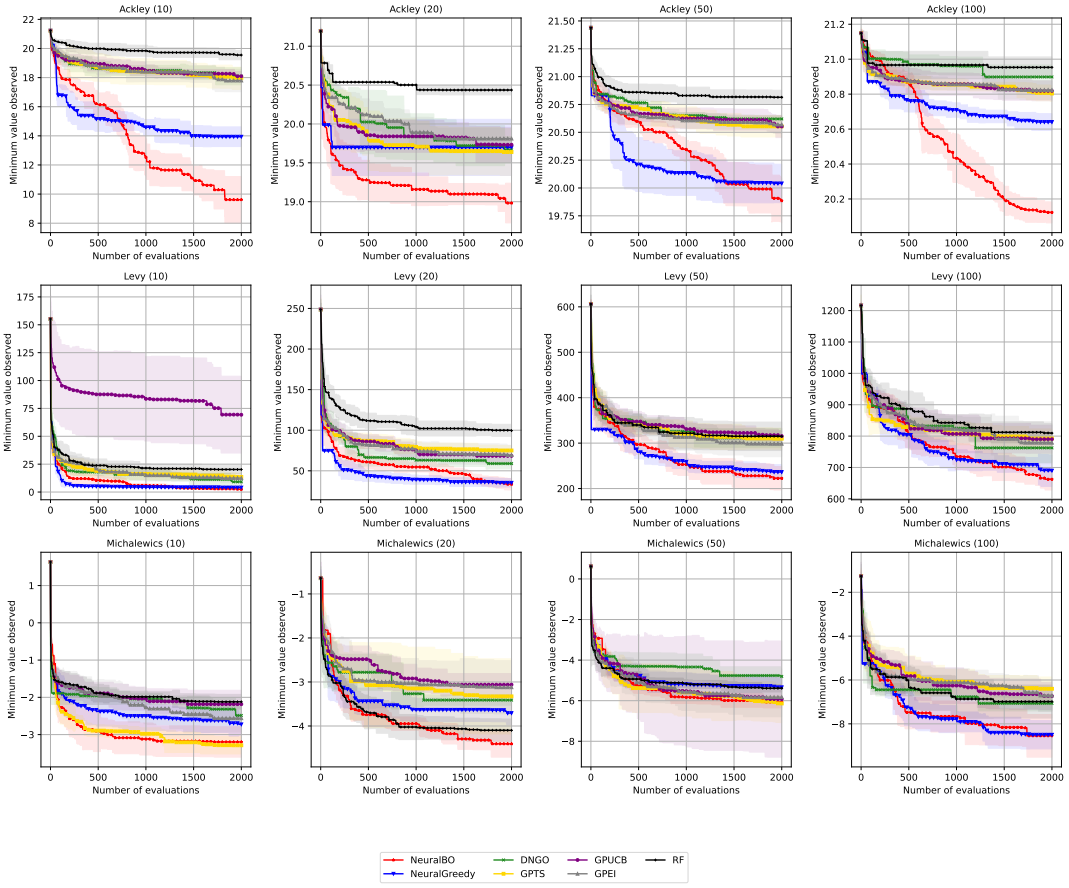


FIGURE 3.1: The plots show the minimum true value observed after optimizing several synthetic functions over 2000 iterations of our proposed algorithm and 6 baselines. The dimension of each function is shown in the parenthesis.

All reported experiments are averaged over 10 runs, each using a random initialization. All the methods start with the same set of initial points. As seen from Figure

		Neural -BO	**Neural Greedy**	**GP -UCB**	**GP -TS**	**GP -EI**	**DNGO**	**RF**
Ackley	D=10	0.85	0.83	0.95	0.95	0.45	0.23	0.79
	D=20	0.31	0.46	0.91	0.90	1.00	1.00	0.93
	D=50	0.95	0.38	0.81	0.99	0.52	0.85	0.67
	D=100	0.84	0.77	0.94	0.82	0.82	0.38	0.97
Levy	D=10	0.75	0.45	0.79	0.72	0.72	0.58	0.63
	D=20	0.84	1.00	1.00	0.79	0.79	0.91	0.89
	D=50	0.76	0.34	0.75	0.75	0.98	0.77	0.37
	D=100	0.76	0.59	0.87	0.90	0.99	0.93	0.95
Micha-lewics	D=10	0.54	0.56	0.95	0.88	0.83	0.77	0.55
	D=20	0.60	0.99	0.43	0.33	0.31	0.56	0.85
	D=50	0.90	0.80	0.64	0.39	0.68	0.99	0.95
	D=100	0.70	0.58	0.96	0.57	0.96	0.93	0.52

TABLE 3.1: The p-values of KS-test "whether the data obtained from running our methods Neural-BO and all baselines are normally distributed".

3.1, Neural-BO is comparable to or better than all other baselines, including GP-based BO algorithms, both NN-based BO algorithms (DNGO and NeuralGreedy) and one algorithm using random forest. Moreover, our algorithm is also promising for high dimensions.

In order to assess the statistical significance of whether our proposed method outperforms all baselines, we conducted a series of tests. First, a Kolmogorov-Smirnov (KS) test was performed to check if the sample sets for the hypothesis test follow a normal distribution. The null hypothesis assumes no difference between the observed and normal distribution. The p-values obtained from the KS test are presented in Table 3.1. As all p-values exceeded 0.05, we failed to reject the null hypothesis, indicating that all data can be considered as normally distributed. Subsequently, one-sided t-tests were employed (as the variance is unknown), along with the Benjamini-Hochberg correction for multiple hypotheses testing, for each pair of (baseline, Neural-BO). These tests aimed to determine whether the baseline achieves a lower objective function value than our proposed method, Neural-BO. The alternative hypothesis $H_a : \mu_{\text{baseline}} > \mu_{\text{Neural-BO}}$ was tested against the null hypothesis $H_0 : \mu_{\text{baseline}} \leq \mu_{\text{Neural-BO}}$. Detailed test results are provided in Table 3.2, where each cell contains two values: the first value represents the p-value obtained from the t-test, and the second value (T or F) indicates the outcome of the Benjamini-Hochberg correction where "T" indicates that the null hypothesis is rejected, whereas an "F" indicates that it is not rejected.

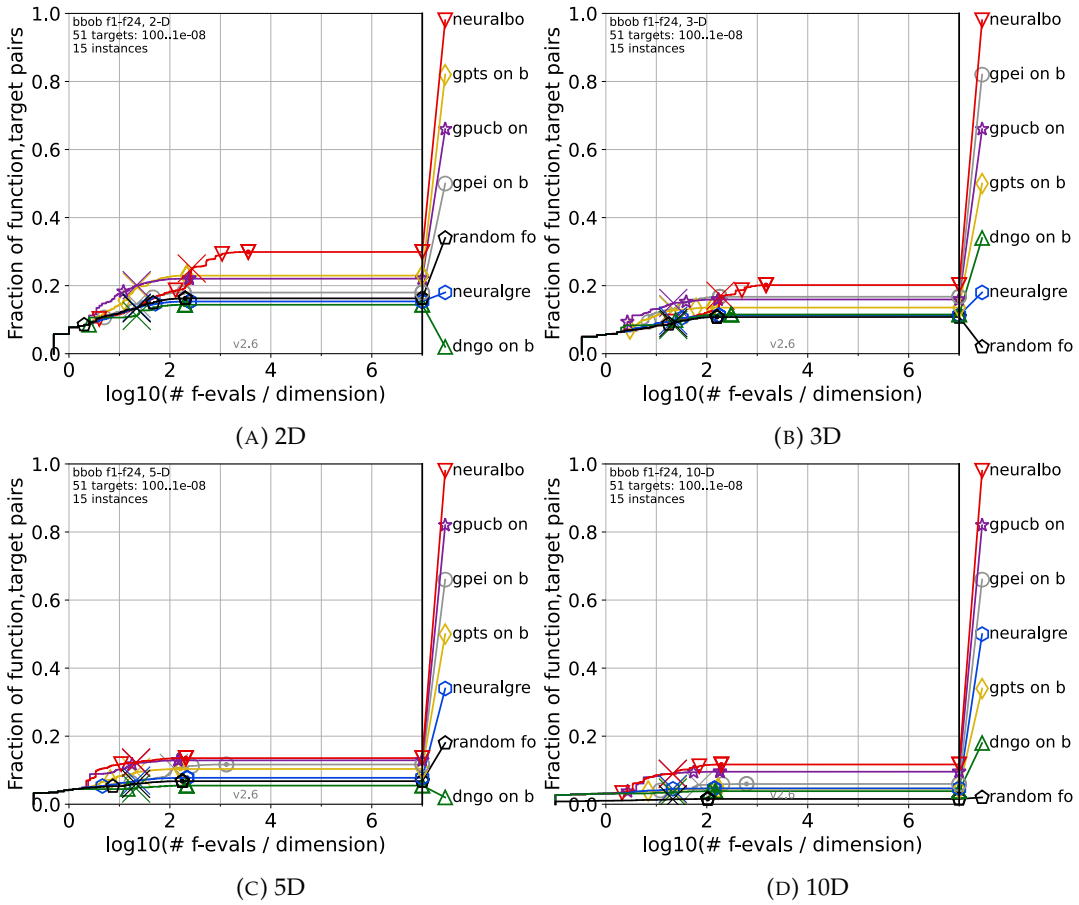
Next, we used the COmparing Continuous Optimizers (COCO) framework[5] to

[5] https://github.com/numbbo/coco

		Neural Greedy	**GPUCB**	**GPTS**	**GPEI**	**DNGO**	**RF**
Ackley	D=10	(2.98e-07, T)	(3.11e-12, T)	(1.64e-11, T)	(2.3e-11, T)	(6.15e-09, T)	(2.09e-13, T)
	D=20	(6.39e-05, T)	(1.61e-06, T)	(4.72e-05, T)	(4.19e-08, T)	(0.000111, T)	(1.29e-05, T)
	D=50	(0.0467, T)	(2.23e-08, T)	(1.89e-08, T)	(8.5e-09, T)	(1.86e-08, T)	(2.11e-10, T)
	D=100	(3.92e-14, T)	(7.98e-16, T)	(2.43e-16, T)	(4.7e-16, T)	(6.44e-16, T)	(5.78e-17, T)
Levy	D=10	(0.000171, T)	(7.98e-06, T	(8.81e-09, T)	(6.6e-07, T)	(1.68e-05, T)	(1.62e-11, T)
	D=20	(0.278, F)	(3.28e-09, T)	(4.32e-11, T)	(1.8e-07, T)	(4.75e-05, T)	(8.01e-14, T)
	D=50	(0.0971, F)	(5.37e-09, T)	(2.72e-07, T)	(6.47e-06, T)	(0.000188, T)	(1.99e-06, T)
	D=100	(0.096, F)	(1.09e-06, T)	(4.85e-08, T)	(9.87e-05, T)	(0.00456, T)	(5.57e-07, T)
Micha-lewics	D=10	(5.9e-06, T)	(7.79e-08, T)	(0.472, F)	(1.68e-05, T)	(7.49e-06, T)	(1.87e-11, T)
	D=20	(2.63e-05, T)	(3e-09, T)	(0.00108, T)	(1.02e-05, T)	(0.000163, T)	(0.0161, T)
	D=50	(0.000344, T)	(3.47e-06, T)	(0.315, F)	(0.101, F)	(0.000871, T)	(0.000132, T)
	D=100	(0.45, F)	(8.2e-05, T)	(8.49e-06, T)	(0.000126, T)	(0.0404, T)	(0.00579, T)

TABLE 3.2: One-sided t-tests were employed to assess whether the baseline achieves a lower function value compared to our proposed method, Neural-BO. The null hypothesis $H_0 : \mu_{\text{baseline}} \leq \mu_{\text{Neural-BO}}$ and the alternative hypothesis: $H_a : \mu_{\text{baseline}} > \mu_{\text{Neural-BO}}$. The p-value corresponding to each test is provided as the first value in each cell. Moreover, to account for multiple hypotheses testing, the Benjamini-Hochberg correction was applied and is reported as the second value in each cell. In the outcome, a "T" indicates that the null hypothesis is rejected, whereas an "F" signifies that it is not rejected.

compare the performance of our method with baseline methods. COCO is a widely used benchmarking platform for continuous optimization algorithms, providing researchers with a standardized set of test functions to evaluate different optimization methods. The framework includes a collection of test functions that are designed to be challenging to optimize and exhibit various properties, such as multimodality, ill-conditioning, or weak global structure. To evaluate our algorithms, we used the BBOB test functions, which consist of 24 noiseless, single-objective, and scalable functions. The quality indicator needs to reach or exceed a predefined target value t for COCO to consider a single run over a problem successful. The performance of different optimizers is measured by the *Fraction of function, target pairs*, which indicates the ratio of problems solved at optimization step T. For more details on how COCO measures the performance of different optimizers, please refer to Hansen et al. (2021). We set the number of evaluations for our optimizers to be 20 times the dimension of the problem.



(A) 2D (B) 3D

(C) 5D (D) 10D

FIGURE 3.2: The results of benchmarking our Neural-BO and the baselines with COCO framework on 24 BBOB noiseless objective functions with four different dimensions {2,3,5,10}.

In Figure 3.2, we present the results of our experiment using the COCO benchmarking framework to evaluate all methods. The benchmarking comprised 24 noiseless functions with 15 instances, where each instance represented a unique condition of the function, such as the location of the optimum. Our method was found to outperform other baselines when assessed using the well-designed and publicly available COCO framework.

3.5.3 Real-world Applications

3.5.3.1 Designing Sensitive Samples for Detection of Model Tampering

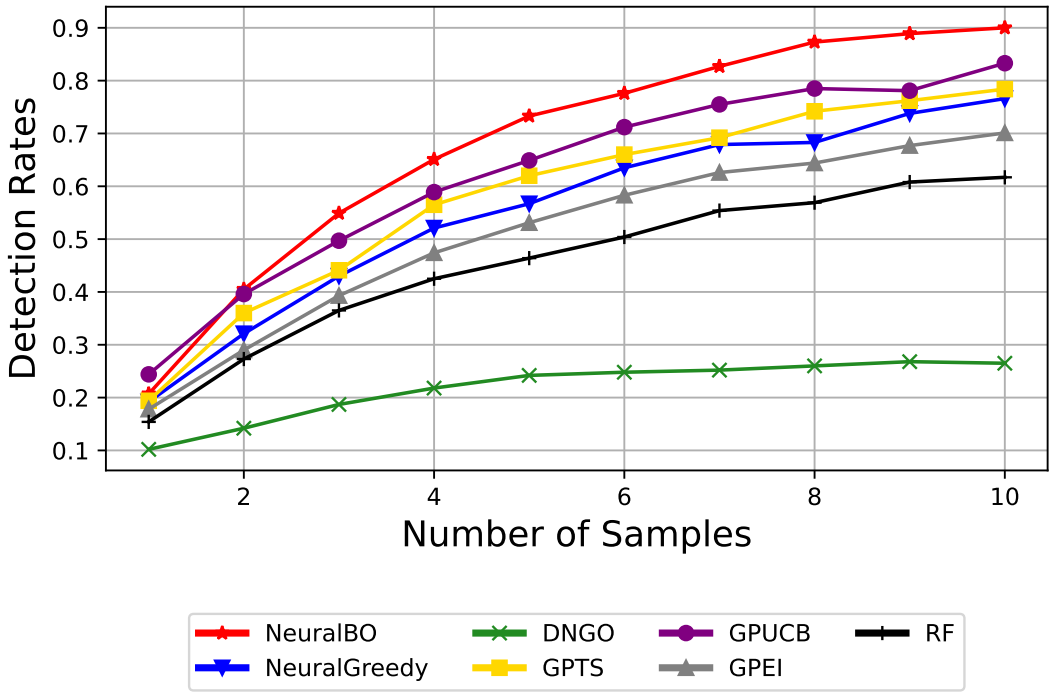


FIGURE 3.3: The plot shows the **detection rates** corresponding to the number of samples on the MNIST dataset. The larger the number of sensitive samples, the higher the detection rate. As shown in the figure, Neural-BO can generate sensitive samples that achieve nearly 90% of the detection rate with at least 8 samples.

We consider an attack scenario where a company offering Machine Learning as a service (MLaaS) hosts its model on a cloud. However, an adversary with backdoor access may tamper with the model to change its weight. This requires the detection of model tampering. To deal with this problem, He, Zhang, and Lee (2018) suggests generating a set of test vectors named *Sensitive-Samples* $\{v_i\}_{i=1}^{n}$, whose outputs predicted by the compromised model will be different from the outputs predicted by the original model. As formalized in He, Zhang, and Lee (2018), suppose we suspect a pre-trained model $f_\theta(x)$ of having been modified by an attacker after it was sent to the cloud service provider. Finding sensitive samples for verifying the model's integrity is equivalent to optimizing task: $v = \text{argmax}_x \left\| \frac{\partial f_\theta(x)}{\partial \theta} \right\|_F$, where $\|\cdot\|_F$ is the Frobenius norm of a matrix. A *successful detection* is defined as "given N_S sensitive samples, there is at least one sample, whose top-1 label predicted by the compromised model is different from the top-1 label predicted by the correct model". Clearly, optimizing this expensive function requires a BO algorithm to be able to work with high-dimensional structured images, unlike usual inputs that take values in hyper-rectangles.

We used a hand-written digit classification model (pre-trained on MNIST data) as the original model and tampered it by randomly adding noise to each weight of this model. We repeat this procedure 1000 times to generate 1000 different tampered

models. The top-1 accuracy of the MNIST original model is 93% and is reduced to 87.73% $\pm$ 0.08% after modifications.

To reduce the computations, we reduce the dimension of images from 28×28 to 7×7 and do the optimization process in 49-dimensional space. After finding the optimal points, we recover these points to the original dimension by applying an upscale operator to get sensitive samples. We compare our method with all other baselines by the average detection rate with respect to the number of sensitive samples. From Figure 4.2, it can be seen that our method can generate samples with better detection ability than other baselines. This demonstrates the ability of our method to deal with complex structured data such as images.

3.5.3.2 Unknown target document retrieval

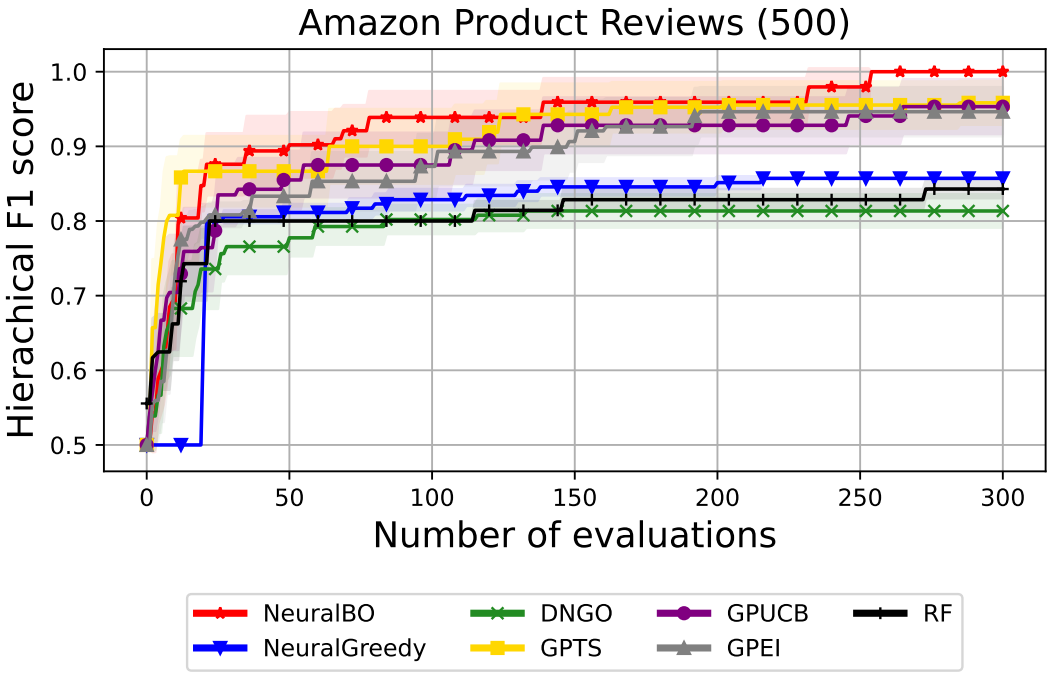


FIGURE 3.4: We search for the most related document for a specified target document in **Amazon product reviews** dataset and report the maximum **hierachical F1 score** found by all baselines. All methods show similar behaviour and Neural-BO performs comparably and much better than GP-based baselines.

Next, we evaluate our proposed method on a retrieval problem where our goal is to retrieve a document from a corpus that matches user's preference. The optimization algorithm works as follows: it retrieves a document, receives user's feedback score and then updates its belief and attempts again until it reaches a high score, or its budget is depleted. The objective target function is defined as the user's evaluation of each document, which usually has a complex structure. It is clear that evaluations are expensive since the user must read each suggestion. Searching for the most relevant document is considered as finding document d in the dataset S_{text} that maximizes the match to the target document d_t: $d = \text{argmax}_{d \in S_{text}} \text{Match}(d, d_t)$, where $\text{Match}(d, d_t)$ is a matching score between documents d and d_t. We represent each

document by a word frequency vector $x_n = (x_{n1}, \cdots, x_{nJ})$, where x_{nj} is the number of occurrences of the j-th vocabulary term in the n-th document, and J is the vocabulary size.

Our experiment uses Amazon product reviews dataset [6], which are taken from users' product reviews from Amazon's online selling platform. This dataset has hierarchical categories, where the category classes are sorted from general to detail. The dataset are structured as: 6 classes in "level 1", 64 classes in "level 2" and 464 classes in "level 3". The number of users' reviews was originally 40K, which was reduced to 37738 after ignoring reviews with "unknown" category. We choose the size of vocabulary to be 500 and use the hierarchical F1-score introduced in Kiritchenko, Matwin, Famili, et al. (2005) as a scoring metric for the target and retrieved documents. We report the mean and standard deviation of hierarchical F1-score between target and retrieved documents over ten runs for all methods in Figure 3.4. Figures 3.4 indicate that our method shows a better performance for the Amazon product review dataset in comparison with other approaches.

3.5.3.3 Optimizing control parameters for robot pushing

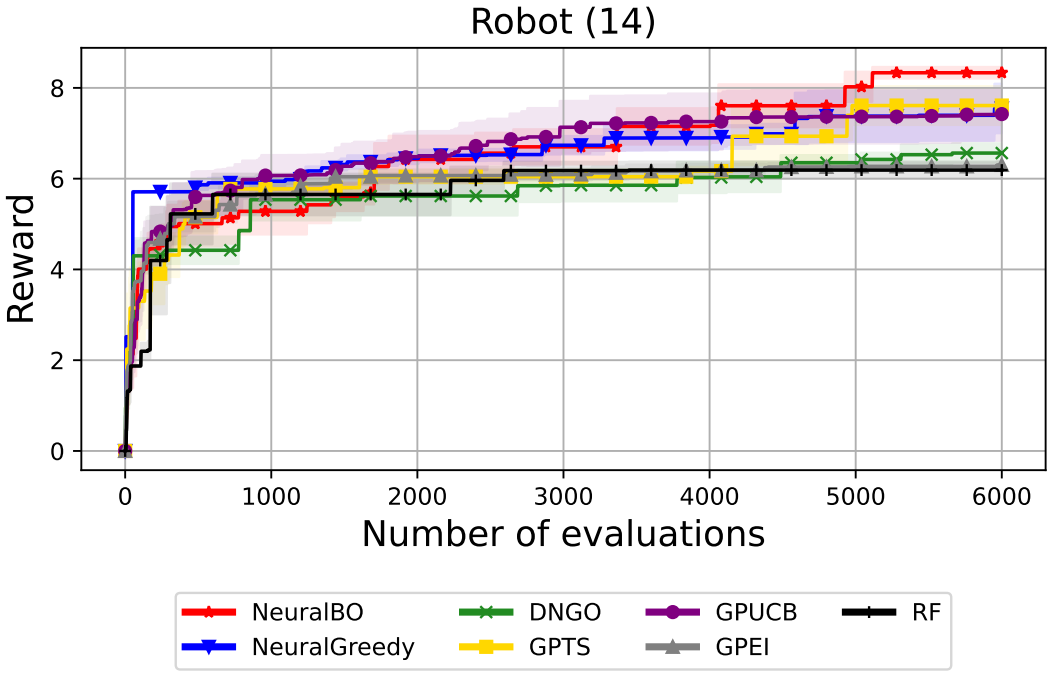


FIGURE 3.5: Optimization results for control parameters of 14D robot pushing problem. The X-axis shows iterations, and the y-axis shows the median of the best reward obtained.

Finally, we evaluate our method on tuning control parameters for the robot pushing problem considered in Wang and Jegelka (2017). We run each method for a total of 6000 evaluations and repeat ten times to take average optimization results. Neural-BO and all other methods are initialized with 15 points. Figure 3.5 summarizes the median of the best rewards achieved by all methods. It can be seen that Neural-BO achieves the highest reward after 5K optimization steps.

[6] https://www.kaggle.com/datasets/kashnitsky/hierarchical-text-classification

3.6 Conclusion

We proposed a new algorithm for Black-box Optimization using Deep Neural Networks. A key advantage of our algorithm is that it is computationally efficient and performs better than traditional Gaussian Process based methods, especially for complex structured design problems. We provided rigorous theoretical analysis for our proposed algorithm and showed that its cumulative regret converges with a sub-linear regret bound. Using both synthetic benchmark optimization functions and a few real-world optimization tasks, we showed the effectiveness of our proposed algorithm.

Chapter 4

Black-box Optimization with Unknown Black-box Constraints via Overparametrized Deep Neural Networks

Black-box Optimization with *unknown constraints* (CBO) methods modify the acquisition function to account for constraints, ensuring feasible solutions that satisfy these conditions while optimizing the objective function. Among these methods, Expected Improvement with Constraints (cEI), first introduced by Schonlau, Welch, and Jones (1998) and later extended by Gardner et al. (2014) and Gelbart, Snoek, and Adams (2014), has been particularly influential. By incorporating feasibility into the acquisition process, cEI directs the optimization toward regions likely to contain feasible solutions. Recent advancements, such as the quasi-Monte Carlo approximation by Letham et al. (2019), have further enhanced cEI's robustness in noisy environments.

Despite the effectiveness of EI-based methods, they encounter challenges in specific scenarios, such as when no feasible points are available or in high-dimensional search spaces. These limitations have driven the development of alternative approaches like Predictive Entropy Search with Constraints (PESC), introduced by Hernández-Lobato et al. (2015), which focuses on uncertainty reduction within feasible regions. However, computational complexities associated with quadrature sampling have restricted its practical applicability. Recent innovations, such as the Min-Value Entropy Search method proposed by Takeno et al. (2022), have simplified sampling procedures, offering a more tractable solution.

Numerical optimization has also emerged as a promising tool for addressing black-box constraints by reformulating constrained problems into simpler unconstrained ones. Techniques like Augmented Lagrangian Bayesian Optimization (ALBO) by Gramacy et al. (2016) and its variant Slack-AL by Picheny et al. (2016) use Augmented Lagrangian Functions (ALF) to iteratively solve reformulated surrogate problems. Recently, Ariafar et al. (2019) introduced ADMMBO, which applies the Alternating Direction Method of Multipliers (ADMM) technique to transform constrained problems into equivalent unconstrained optimization problems. While effective, these methods often require additional variables, increasing computational costs and limiting scalability.

To further enhance CBO, penalty functions and primal-dual methods have been employed to handle constraint violations during optimization. For instance, Lu and Paulson (2022) proposed a penalty-function approach that adds a penalty term to

the objective function for violations, converting the constrained problem into an unconstrained one. Similarly, Zhou and Ji (2022) introduced a primal-dual approach that balances optimizing the objective with minimizing violations. However, these methods often rely on careful parameter tuning, making their implementation more challenging.

In parallel with empirical advancements, theoretical research has begun addressing the lack of formal guarantees in CBO. For example, Lu and Paulson (2022) proposed a penalty-based regret bound that combines regret from the objective function and penalties for constraint violations. Xu et al. (2023) further expanded this analysis by separately evaluating cumulative regret and constraint violations. In contrast, Nguyen et al. (2023) provided theoretical performance guarantees for CBO under unknown constraints in a decoupled setting, where cumulative regret is defined as the sum of both objective regret and constraint violations.

While Gaussian Processes (GPs) have been a traditional selection for modeling in CBO, their poor computational scalability remains a significant limitation. The cubic complexity of kernel matrix inversion grows prohibitive with increasing constraints. In contrast, Deep Neural Networks (DNNs) offer a scalable alternative, excelling in feature extraction and maintaining linear complexity with dataset size. Although DNNs have seen success in unconstrained black-box optimization (Snoek et al., 2015), contextual bandits in discrete search spaces (Zhou, Li, and Gu, 2020; Zhang et al., 2021), and constrained optimization with Augmented Lagrangian methods (Ariafar et al., 2019), applying DNNs to constrained black-box optimization with theoretical guarantees remains an open problem. This gap motivates the exploration of neural network-based approaches for CBO.

In Chapter 3, we inherited a similar motivation as in Chapter 3 to apply for BO with unknown, expensive constraints setting. We provided Neural-CBO, a method to alternate canonical GP by DNN as the surrogate model in BO in CBO.

4.1 Proposed Neural-CBO Method

Here, we will introduce the general DNNs architect used to model objective function and the constraints, followed by basis theory accompanying with these networks. Then, the Neural-CBO algorithm is introduced.

4.1.1 The Deep Neural Network for an Arbitrary Function f_a

Given a black-box, expensive function f_a, we use a fully connected neural network, denoted as $a(\mathbf{x}; \mathbf{W})$, to model f_a:

$$a(\mathbf{x}; \mathbf{W}) = \frac{\mathbf{q}^\top}{\sqrt{m}} \mathbf{D}^{(L)}(\mathbf{x}) \mathbf{W}^{(L)} \dots \frac{1}{\sqrt{m}} \mathbf{D}^{(1)}(\mathbf{x}) \mathbf{W}^{(1)} \mathbf{x}, \tag{4.1}$$

where $\mathbf{q} \in \mathbb{R}^m$ is the last layer weight, $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}$, $\mathbf{W}^{(l)} \in \mathbb{R}^{m \times m}$ for $2 \leq l \leq L$ is the weight of the l-th hidden layer. The matrix $\mathbf{D}^{(l)}(\mathbf{x})$ is associated with the ReLU activation function and is defined as:

$$\mathbf{D}^{(l)}(\mathbf{x}) = \text{diag}\{\mathbf{1}_{\{\langle w_i^{(l)}, \mathbf{h}^{(l-1)}(\mathbf{x}) \rangle \geq 0\}}\} \in \mathbb{R}^{m \times m},$$

with m as the number of neurons in the hidden layer l, and $\mathbf{h}^{(l)}(\mathbf{x})$ is the output of the l-th layer given by

$$\mathbf{h}^{(l)}(\mathbf{x}) = \frac{1}{\sqrt{m}}\mathbf{D}^{(l)}(\mathbf{x})\mathbf{W}^{(l)} \dots \frac{1}{\sqrt{m}}\mathbf{D}^{(1)}(\mathbf{x})\mathbf{W}^{(1)}\mathbf{x},$$

with $\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$.

At time $t = 0$, each weight matrix $\mathbf{W}^{(l)}, 2 \leq l \leq L$ is initialized as $\begin{bmatrix} \mathbf{\Psi} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Psi} \end{bmatrix}$, where $\mathbf{\Psi}$ is a Gaussian random matrix with independent and identically distributed (i.i.d.) standard normal entries. Additionally, the outer weights $\mathbf{q} = (\hat{\mathbf{q}}, -\hat{\mathbf{q}})^\top$ are set as random variables, and each entry of $\mathbf{b}$ is set with an equal probability of being either -1 or 1, and remain fixed throughout the training process. This initialization method is commonly employed in the literature, as seen in works like Du et al. (2018) and Arora et al. (2019), and it can be verified that, with this initialization scheme, $a(\mathbf{x}; \mathbf{W}_0) = 0$, for all input $\mathbf{x}$.

The neural network is trained by running the stochastic gradient descent on the streaming data in *one pass*. In particular, given the initialization $\{\mathbf{W}_0^{(l)}\}_{l=1}^L$ and last layer weight $\mathbf{q}$, the l-th layer weight matrix at the t-th iteration is updated by minimizing the L_2 loss as:

$$\mathbf{W}_{t+1}^{(l)} = \mathbf{W}_t^{(l)} + \alpha_t(y_t - a(\mathbf{x}_t; \mathbf{W}_t))\frac{\partial a(\mathbf{x}_t; \mathbf{W}_t)}{\partial \mathbf{W}^{(l)}}, \tag{4.2}$$

where α_t is the step size, and $\{\mathbf{x}_t, y_t\}$ is the observation at the t-th optimization iteration.

To estimate the uncertainty of the function f_a modeled by $a(\mathbf{x}; \mathbf{W})$, we adopt the variance formula from recent advances in neural contextual bandits research (Zhou, Li, and Gu, 2020; Kassraie and Krause, 2022):

$$\sigma_{a,t}(\mathbf{x}) = \sqrt{\mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)}, \tag{4.3}$$

where

$$\begin{aligned} \mathbf{g}_a(\mathbf{x}; \mathbf{W}) &= \nabla_{\mathbf{W}} a(\mathbf{x}; \mathbf{W}), \text{ and} \\ \mathbf{U}_{a,t} &= \mathbf{U}_{a,t-1} + \mathbf{g}_a(\mathbf{x}_t; \mathbf{W}_0)\mathbf{g}_a(\mathbf{x}_t; \mathbf{W}_0)^\top, \end{aligned} \tag{4.4}$$

We again inherit the concept of Neural Tangent Kernel (NTK) introduced in Section 2.3.2.1 for our theretical analysis. We briefly remind this concept through the following definition:

Definition 4.1.1. Given the L-layer neural network $a(\mathbf{x}; \mathbf{W})$ with input $\mathbf{x}$ and parameter $\mathbf{W}$ as defined in Eqn. 4.1, a NTK matrix $\mathbf{H}_t$ for a sequence of weights $\mathbf{W}_t$ can be defined as:

$$\mathbf{H}_t[i,j] := \left\langle \frac{\partial a(\mathbf{x}_i; \mathbf{W}_t)}{\partial \mathbf{W}}, \frac{\partial a(\mathbf{x}_j; \mathbf{W}_t)}{\partial \mathbf{W}} \right\rangle = \sum_{l=1}^{L} \mathbf{H}_t^{(l)}[i,j],$$

where $\mathbf{H}_t^{(l)}[i,j] := \left\langle \frac{\partial a(\mathbf{x}_i; \mathbf{W}_t)}{\partial \mathbf{W}^{(l)}}, \frac{\partial a(\mathbf{x}_j; \mathbf{W}_t)}{\partial \mathbf{W}^{(l)}} \right\rangle$ is the NTK from the l-th hidden layer, for all $1 \leq i, j \leq T$.

Next, we present the common and well-established assumptions. The following assumption indicates the smoothness property of the unknown function f_a.

Assumption 4.1.2. We assume that $f_a \in \mathcal{H}_{k_a}(\mathcal{D})$, where $\mathcal{H}_{k_a}(\mathcal{D})$ is the Reproducing Kernel Hilbert Space (RKHS) associated with a real-valued function f_a defined on the domain $\mathcal{D}$. This space is induced by the Neural Tangent Kernel k_a, which arises from a neural network $a(\mathbf{x}; \mathbf{W})$. In particular, the RKHS $\mathcal{H}_{k_a}$ induces an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}_{k_a}}$ with the reproducing property: for all $f_a \in \mathcal{H}_{k_a}(\mathcal{D})$, we have $f_a(\mathbf{x}) = \langle f_a, k_a(\cdot, \mathbf{x}) \rangle_{\mathcal{H}_{k_a}}$. The induced norm is bounded and serves as a measure of the smoothness of f_a w.r.t the kernel function k_a: $\|f_a\|_{\mathcal{H}_{k_a}} = \sqrt{\langle f_a, f_a \rangle_{\mathcal{H}_{k_a}}} \leq B_a$.

To ensure that the noise arising from querying unknown function f_a remains bounded and manageable, we impose the following assumption:

Assumption 4.1.3. We assume the noises $\{\zeta_t\}_{t=1}^T$ where $\zeta_t = o_t - f_a(\mathbf{x}_t)$ are conditionally sub-Gaussian with parameter $R_a > 0$, where $\{\zeta_t\}_{t=1}^T$ is assumed to capture the noises induced by querying the black-box, expensive function $f_a(\cdot)$.

$$\forall t \geq 0,\ \forall \lambda_a \in \mathbb{R},\ \mathbb{E}[e^{\lambda_a \zeta_t} | \mathcal{F}_{a,t-1}] \leq \exp\left(\frac{\lambda_a^2 R_a^2}{2}\right),$$

where $\mathcal{F}_{a,t-1}$ are the σ-algebra generated by the random variables $\{\mathbf{x}_i, \zeta_i\}_{i=1}^{t-1} \cup \{\mathbf{x}_t\}$.

To manage the approximation error, several technical lemmas impose the following condition on the width of the neural network introduced in Eqn. 4.1.

Condition 4.1.4. *Throughout the section, the width of each hidden layer m satisfies is assumed to satisfy:*

$$m \geq d^9 \exp\Big(\Omega(\nu L C^L \log T)\Big), \tag{4.5}$$

for some absolute constant C. Besides, the step size $\alpha_t \leq \frac{\nu}{t+1}$, where ν is a parameter and independent of dimension d and width m.

Before going to our main algorithm, we provide the confidence bound, the fundamental component in almost all BO algorithms to guide algorithm design and ensure theoretical guarantee. The lemma demonstrates that by following the network width condition stated in Condition 4.1.4, the prediction of the trained neural network $a(\cdot; \mathbf{W}_{t-1})$ is concentrated around the actual value of the function $f_a(\cdot)$.

Lemma 4.1.5. *Let Assumptions 4.1.2 and 4.1.3 hold. Using neural network $a(\mathbf{x}; \mathbf{W})$ satisfied Condition 4.1.4 to model an arbitrary function f_a. Setting the step size at training step t as $\alpha_t \leq \frac{\nu}{(T+1)^2}$, then for any $\delta \in (0,1)$, with probability at least $1 - \delta \exp\big(\Omega(C^{-L} m^{1/36})\big)$, the following holds for all $\mathbf{x} \in \mathcal{D}$ and $1 \leq t \leq T$:*

$$\begin{aligned}
&|f_a(\mathbf{x}) - a(\mathbf{x}; \mathbf{W}_{t-1})| \leq \beta_{a,t} \sigma_{a,t-1}(\mathbf{x}) + \frac{\mathcal{E}(m)}{T+1}, \\
&\beta_{a,t} = \left(B_a + R_a \sqrt{\gamma_{t,a} + 2 + 2\log(1/\delta)}\right), \\
&\mathcal{E}(m) = \mathcal{O}(C^{2L} L^{3/2} m^{11/36}).
\end{aligned}$$

Here, the coefficient $\beta_{a,t}$ control the uncertainty of $a(\mathbf{x}; \mathbf{W}_{t-1})$ about $f_a(\mathbf{x})$ at $\mathbf{x}$, while $\mathcal{E}(m)$ indicates the approximation error appeared when using the neural network's output $a(\mathbf{x}; \boldsymbol{W})$ to learn the underlying function f_a.

To facilitate the following algorithm design and discussion, we introduce the Lower Confidence Bound (LCB) and Upper Confidence Bound (UCB) functions w.r.t

the *arbitrary* function f_a:

$$\text{LCB}_{a,t}(\mathbf{x}, \mathbf{W}_t) = a(\mathbf{x}, \mathbf{W}_t) - \beta_{a,t}\sigma_{a,t}(\mathbf{x}) - \frac{\mathcal{E}(m)}{T+1},$$
$$\text{UCB}_{a,t}(\mathbf{x}, \mathbf{W}_t) = a(\mathbf{x}, \mathbf{W}_t) + \beta_{a,t}\sigma_{a,t}(\mathbf{x}) + \frac{\mathcal{E}(m)}{T+1},$$

where $\sigma_{a,t}(\mathbf{x})$ is calculated using the formulate given in Eqn. 4.3. Then, with high probability, f_a is bounded by $\text{LCB}_{a,t}(\mathbf{x}, \mathbf{W}_t)$ and $\text{UCB}_{a,t}(\mathbf{x}, \mathbf{W}_t)$ as in the following corollary:

Corollary 4.1.6. *Let Assumptions 4.1.2, 4.1.3 and Condition 4.1.4 hold. Then with probability at least* $1 - \delta \exp\big(\Omega(C^{-L}m^{1/36})\big)$*, the following holds for all* $\mathbf{x} \in \mathcal{D}$ *and* $1 \leq t \leq T$*:*

$$f_a(\mathbf{x}) \in [\text{LCB}_{a,t}(\mathbf{x}, \mathbf{W}_t), \text{UCB}_{a,t}(\mathbf{x}, \mathbf{W}_t)].$$

4.1.2 Neural-CBO Algorithm

In the remaining parts of this chapter, we refer to $v(\mathbf{x}; \boldsymbol{\theta})$ and $\{u_{c_i}(\mathbf{x}; \boldsymbol{\omega}_{c_i})\}_{i=1}^K$ as the neural network models for the unknown objective function f and constraints $\{c_i\}_{i=1}^K$, respectively.

Our algorithm starts by initializing the neural networks $v(\mathbf{x}; \boldsymbol{\theta})$ and $\{u_{c_i}(\mathbf{x}; \boldsymbol{\omega}_{c_i})\}_{i=1}^K$ using the initialization scheme described in Section 4.1.1. We use the EI acquisition function to identify the next samples within the feasible region, determined by applying LCB-based conditions to all constraints. These conditions ensure that constraints are satisfied, while EI maintains a balance between exploration and exploitation, particularly when the feasible region is much smaller than the overall search space (Line 4 of Alg 4). At each optimization iteration t, the next evaluation point $\mathbf{x}_t$ is determined by maximizing the acquisition function $\text{EI}_{f,t}(\mathbf{x})$ subject to the lower confidence bound constraints for all unknown black-box constraints $\{c_i(\mathbf{x})\}_{i=1}^K$:

$$\text{LCB}_{c_i,t}(\mathbf{x}, \boldsymbol{\omega}_{c_i,t}) \leq 0, \forall i \in [K].$$

To handle noisy observations, we utilized the standard choice of the incumbent, which is the best value of the mean function so far: $\mu_t^+ = \max_{\mathbf{x}_k \in \mathcal{D}_{t-1}} v(\mathbf{x}_k; \boldsymbol{\theta}_{t-1})$, where the evaluations of both objective function and constraints on $\mathbf{x}_k$ yield noisy observations, such as the objective value $y_k = f(\mathbf{x}_k) + \epsilon_k$ and constraint values $\{z_{c_i,k}\}_{i=1}^K$, with each constraint observation given by $z_{c_i,k} = c_i(\mathbf{x}_k) + \eta_{c_i,k}$ and $\mathcal{D}_{t-1} = \{\mathbf{x}_k, y_k, z_{c_1,k}, \ldots, z_{c_K,k}\}_{k=1}^{t-1}$.

Further, to control the error $\mathcal{E}(m)$ incurred by neural network model approximation, this noisy EI version can be written in closed form, following Tran-The et al. (2022) as:

$$\begin{aligned}\text{EI}_{f,t}(\mathbf{x}) &= \mathbb{E}[\max\{0, v(\mathbf{x}; \boldsymbol{\theta}_{t-1}) - \mu_t^+ + \mathcal{E}(m)\}] \\ &= \rho(v(\mathbf{x}; \boldsymbol{\theta}_{t-1}) - \mu_t^+ + \mathcal{E}(m), \sigma_{f,t}(\mathbf{x})),\end{aligned}$$

where $\rho(u, v) = \begin{cases} u\boldsymbol{\Phi}(\frac{u}{v}) + v\phi(\frac{u}{v}), & \text{if } v > 0, \\ \max\{u, 0\}, & \text{if } v = 0. \end{cases}$

Then, we updated the dataset $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t, y_t, z_{c_1,t}, \ldots, z_{c_K,t}\}$. The parameters $\boldsymbol{\theta}$ (for the objective function) and $\{\boldsymbol{\omega}_{c_i}\}_{i=1}^K$ (for the constraints) are then updated separately by minimizing the L_2 loss on the new observation using stochastic gradient descent (SGD) described in Eqn. 4.2.

Algorithm 3

Input: The input space $\mathcal{D}$, the optimization budget T, the number of constraints N

1: Initialize neural network models parameters $\boldsymbol{\theta}_0, \{\boldsymbol{\omega}_{c_i,0}\}_{i=1}^K$.
2: Initialize $\mathbf{U}_{f,0} = \mathbf{I}, \mathbf{U}_{c_i,0} = \mathbf{I}, \forall i \in [1 \dots K]$,
3: **for** $t = 1$ to T **do**
4: Choose $\mathbf{x}_t = \text{argmin}_{\mathbf{x}\in\mathcal{D}} \text{EI}_{f,t}(\mathbf{x})$ subject to $\text{LCB}_{c_i,t}(\mathbf{x}, \boldsymbol{\omega}_{c_i,t}) \leq 0, \forall i \in [K]$
5: Observe the noisy evaluations of objective function $y_t = f(\mathbf{x}_t) + \epsilon_t$ and constraints $\{z_{c_i,t} = c_i(\mathbf{x}_t) + \eta_{c_i,t}\}_{i=1}^K$.
6: Update observations set $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t, y_t, z_{c_1,t}, \dots, z_{c_K,t}\}$
7: Update the neural network parameters $\boldsymbol{\theta}_0, \{\boldsymbol{\omega}_{c_i,0}\}_{i=1}^K$ using Eqn 4.2.
8: Update $\mathbf{U}_{f,t}$ and $\mathbf{U}_{c_i,t}, \forall i \in [1 \cdots K]$ separately using Eqn. 4.4.
9: **end for**

4.2 Theoretical Analysis

To evaluate the performance of black-box optimization methods, much of the prior research on unconstrained BO has focused on minimizing cumulative regret. As introduced in Section 2.3.6, the cumulative regret after T iterations is defined as:

$$R_T = \sum_{t=1}^{T} r_t,$$

where $r_t = f(\mathbf{x}_t) - f(\mathbf{x}^*)$ represents the instantaneous regret, quantifying the difference between the value of the unknown function f at the optimal point, $\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x})$, and the value of the function at point $\mathbf{x}_t$, which is selected by the algorithm at iteration t. However, since $f(\mathbf{x}^*)$ represents the optimal value under constraints, the algorithm may sometimes sample infeasible points with lower objective values than $f(\mathbf{x}^*)$. To account for this, following Xu et al. (2023) and Nguyen et al. (2023), we inherited the *positive regret* definition as $r_t^+ = [f(\mathbf{x}_t) - f(\mathbf{x}^*)]^+$, where $[\cdot]^+ := \max\{0, \cdot\}$. Additionally, to measure constraint satisfaction, constraint *violation* is defined as $v_{c_i,t} = [c_i(\mathbf{x}_t)]^+$. Then, we introduce the *cumulative positive regret* for the objective function, R_T^+, and the *cumulative violation* for each constraint, $V_{c_i,T}$. These metrics measure the additional cost incurred due to sub-optimal decisions and violations of the constraints over time by running the algorithm.

Definition 4.2.1 (Cumulative Positive Regret and Cumulative Violation).

$$R_T^+ = \sum_{t=1}^{T} [f(\mathbf{x}_t) - f(\mathbf{x}^*)]^+,$$
$$V_{c_i,T} = \sum_{t=1}^{T} [c_i(\mathbf{x}_t)]^+, \forall i \in [K]$$

4.2.1 Detailed Assumptions for Objective Function and Constraints

We apply the general assumption stated in the Assumption 4.1.2 and 4.1.3 on both objective function and constraints:

- **Objective function**: $f \in \mathcal{H}_{k_f}(\mathcal{D})$, $\|f\|_{\mathcal{H}_{k_f}} \leq B$, where k_f is corresponding to $v(\cdot, \boldsymbol{\theta})$. The noisy observation at step t is $y_t = f(\mathbf{x}_t) + \epsilon_t$, where $\{\epsilon_i\}_{i=1}^t$ is sub-Gaussian with parameter R_f and variance λ_f.

- **Constraint**: $c_i \in \mathcal{H}_{k_{c_i}}(\mathcal{D}), \|c_i\|_{\mathcal{H}_{k_{c_i}}} \leq S_i$, where k_{c_i} is corresponding to $u_{c_i}(\cdot, \omega_{c_i}), \forall i = 1, \ldots, K$. The noisy observation at step t is $z_{c_i,t} = c_i(\mathbf{x}_t) + \eta_{c_i,t}$, where $\{\eta_{c_i,t}\}_{i=1}^{t}$ is sub-Gaussian with parameter R_{c_i} and variance λ_{c_i}.

Now we can now state our main theorem:

Theorem 4.2.2. *Under Assumption 4.1.2, Assumption 4.1.3 and Condition 4.1.4, set the step size used to train the neural networks in Alg. 3 as* $\alpha_t \leq \frac{\nu}{(T+1)^2}$, *then for any* $\delta \in (0,1)$, *with probability at least* $1 - \delta \exp\big(\Omega(C^{-L} m^{1/36})\big)$, *the Cumulative Regret* R_T *and Cumulative Violation* $V_{c_i,T}$ *after* T *iterations are bounded as:*

$$V_{c_i,T} \leq 2\beta_{c_i,T} \sqrt{\frac{S_i T}{\log(S_i+1)}(2\gamma_{c_i,T}+1)} + 2\mathcal{E}(m),$$

$$R_T \leq R_T^+ \leq 2\beta_{f,T} \sqrt{\frac{BT}{\log(B+1)}(2\gamma_{f,T}+1)} + 2\mathcal{E}(m),$$

where $\mathcal{E}(m) = \mathcal{O}(C^{2L} L^{3/2} m^{11/36})$. *Especially, by choosing* $m = \Omega(d^9 \exp\big(\nu L C^L \log T\big)))$, *the Cumulative Regret and Cumulative Violation enjoy the following results:*

$$V_{c_i,T} = \mathcal{O}(\gamma_{c_i,T}\sqrt{T}), \quad R_T = \mathcal{O}(\gamma_{f,T}\sqrt{T}).$$

Remark 4.2.3. Unlike previous works (Zhou, Li, and Gu, 2020; Zhang et al., 2021) that require a neural network width of $m = \Omega(T^6)$ for convergence when modeling the objective function, our analysis builds on recent analyses from Xu and Zhu (2024), which show that only a linear condition of $m = \Omega(T)$ is needed. Furthermore, while Xu and Zhu (2024) focus on the input domain $\mathbb{S}^{d-1}$, we can adapt to inputs $\mathbf{x} \in \mathbb{R}^d$ with $0 < n_l < \|\mathbf{x}\| < n_b$ (where n_l and n_b are positive constants) without changing the order of T in the width condition for m. Similar arguments are noted in (Du et al., 2018; Cao and Gu, 2020).

4.3 Experiments

In this section, we outline and discuss our experimental results. We apply our method to synthetic, real-world functions.

4.3.1 Experimental Setup

For all experiments, we compared our algorithm with well-known Constrained EI (cEI), the extension of EI into constrained BO from Gardner et al. (2014). Besides, we also compare our algorithm with recent state-of-the-art algorithms in unknown constrained BO, including ADMMBO (Ariafar et al., 2019), UCB-C (Nguyen et al., 2023) and ConfigOpt (Xu et al., 2023). For our proposed Neural-CBO algorithm, we employ fully connected deep neural networks as the surrogate models for both objective function and constraints. Implementation details of our Neural-CBO algorithm as well as baselines are as follows:

- **Constrained EI** (cEI) (Gardner et al., 2014) integrates feasibility into the acquisition function by multiplying the probability of feasibility into EI value at every point in the search space.

- **ConfigOpt** (Xu et al., 2023): Optimize LCB-based acquisition function for the objective, which satisfies LCB-based conditions for constraints. For cEI and ConfigOpt, we used the public implementation provided at GitHub repository: `https://github.com/PREDICT-EPFL/ConfigOPT`.
- **ADMMBO** (Ariafar et al., 2019): Reformulates the constrained optimization problem into an unconstrained one using the Alternating Direction Method of Multipliers (ADMM) framework. As the official implementation of ADMMBO is written in Matlab and available at `https://github.com/SetarehAr/ADMMBO`, we use our own Python implementation based on the official implementation.
- **UCB-C** (Nguyen et al., 2023): Similar to ConfigOpt, but using a UCB-based acquisition function for the objective, we utilized the implementation obtained directly from the authors.

Neural-CBO implementation details: As described in Section 4.1, the network's weights are initialized with independent samples drawn from a normal distribution $\mathcal{N}(0,1)$. We also initialize fixed outer weight $\mathbf{q}$ to be a symmetric Bernoulli random variable with equal probability to be -1 or 1. To train the surrogate neural network models, we use a Gradient Descent optimizer described in the main paper with a learning rate $\alpha = 0.001$. The network width depends on the tasks and is set to be $m = T$, where T is the number of optimization iterations. We choose the network depth $L = 2$ to reduce computational costs.

4.3.2 Synthetic Benchmark Functions

We conducted optimization experiments on four synthetic objective functions: Branin, Ackley, Simionescu and Hartmann. The input dimension of each objective function and the corresponding number of constraints are summarized in Table 4.1. Due to space limitations, we present the expression of each function and its constraints in the Supplementary material. The noise in function evaluations follows

TABLE 4.1: The input dimension and number of constraints for each synthetic objective function.

Obj	Branin	Simionescu	Ackley	Hartmann
Dim	2	2	5	6
Constraints	1	1	2	1

a normal distribution with zero mean, and the variance is set to 1% of the function range. All experiments reported here are averaged over 20 runs, each with random initialization. We report the (Log10 of) the Best Positive Regret plus Violation in Figure 4.1

To ensure statistical significance, we performed one-sided t-tests to assess whether a baseline outperforms Neural-CBO in terms of the best positive regret plus violation. The null hypothesis is $H_0 : \mu_{\text{baseline}} \leq \mu_{\text{Neural-CBO}}$, and the alternative hypothesis is $H_a : \mu_{\text{baseline}} > \mu_{\text{Neural-CBO}}$, where μ_{baseline} and $\mu_{\text{Neural-CBO}}$ represent the means of the (Log10 of) Best Positive Regret plus Violation values of the baseline and our proposed Neural-CBO, respectively. Note that lower values indicate better performance. We present the statistical test results for four synthetic benchmark

functions and two real-world tasks (described in Section 4.3.3 and 4.3.4) in Table 4.2. Each cell in the table shows the p-value from the t-test as the first value. To account for multiple comparisons, the Benjamini-Hochberg correction was applied, with the corrected value provided as the second value. A result is labeled as "T" if the null hypothesis is rejected, meaning that Neural-CBO is statistically better to the compared baselines. Conversely, a result is labeled "F" if we cannot reject the null hypothesis, meaning that the baselines and the Neural-CBO are comparable.

TABLE 4.2: One-sided t-tests to evaluate whether the baseline outperforms Neural-CBO in terms of best positive regret plus violation.

	ConfigOpt	**cEI**	**UCB-C**	**ADMMBO**
Branin	(8.25e-06, T)	(6.41e-20, T)	(4.68e-73, T)	(6.86e-102, T)
Simionescu	(6.19e-07, T)	(2.42e-13, T)	(2.13e-25, T)	(1.32e-78, T)
Ackley	(6.95e-29, T)	(0.13, F)	(6.4e-97, T)	(0.21, F)
Hartmann	(0.00594, T)	(1.54e-59, T)	(4.88e-118, T)	(1.2e-87, T)
Gas Transmission	(1.52e-46, T)	(1.78e-48, T)	(1.66e-25, T)	(1.54e-73, T)
Speed Reducer	(0.77, F)	(6.99e-74, T)	(5.22e-77, T)	(0.38, F)

We analyze three real-world constrained black-box optimization tasks: gas transmission compressor and speed reducer designs from Kumar et al. (2020), and a third inspired by He, Zhang, and Lee (2018). Details of each task will follow in the upcoming sections.

4.3.3 Gas Transmission Compressor Design

The main objective is to minimize operational costs or energy consumption. This requires identifying the optimal configuration of the compressor by optimizing four design variables. The problem involves $d = 4$ input dimensions and includes $K = 1$ constraint. The mathematics formula for this problem is:

$$\begin{aligned} f(\mathbf{x}) &= 8.61 \times 10^5 \mathbf{x}_1^{1/2} \mathbf{x}_2 \mathbf{x}_3^{-2/3} \mathbf{x}_4^{-1/2} + 3.69 \times 10^4 \mathbf{x}_3 + 7.72 \times 10^8 \mathbf{x}_1^{-1} \mathbf{x}_2^{0.219} \\ &\quad - 765.43 \times 10^6 \mathbf{x}_1^{-1}, \\ \text{s.t } c(\mathbf{x}) &= \mathbf{x}_4 \mathbf{x}_2^{-2} + \mathbf{x}_2^{-2} - 1 \leq 0 \end{aligned}$$

4.3.4 Speed Reducer Design

This task involves designing a speed reducer for a small aircraft engine, focusing on minimizing weight while meeting several constraints, including bending stress on gear teeth, surface stress, transverse deflections of shafts, and shaft stresses. The

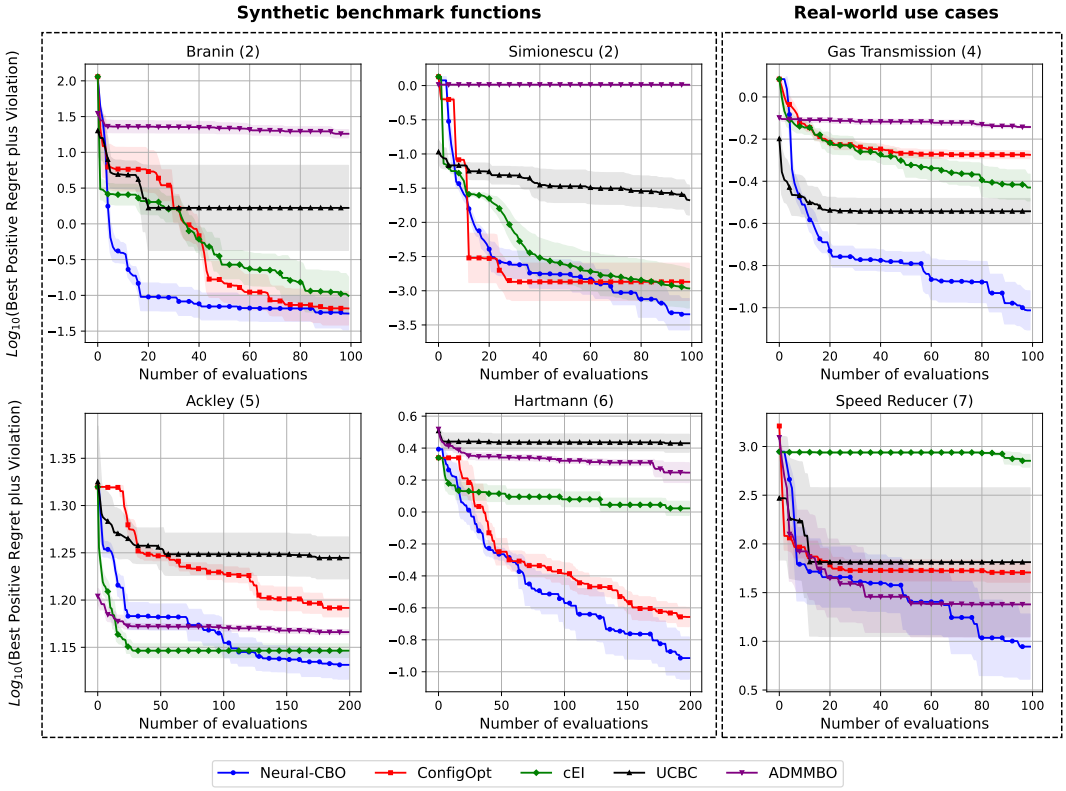


FIGURE 4.1: The plots show (Log10 of) the Best Positive Regret plus Violation up to step t, which is $\min_{t\in[T]}[f(\mathbf{x}_t) - f^*]^+ + \sum_{k=1}^{K}[c_k(\mathbf{x}_t)]^+]$, comparing our proposed algorithm and four baselines. The dimension of each objective function is shown in the parenthesis. The left group is four synthetic functions introduced in Section 4.3.2, while the right group is the optimization results of Gas Transmission Compressor Design and Speed Reducer Design, described in Section 4.3.3 and 4.3.4.

problem includes 7 decision variables and 11 constraints, resulting in an input dimension of $d = 7$ and $K = 11$ constraints and can be formulated as:

$$
\begin{aligned}
f(\mathbf{x}) &= 0.7854\mathbf{x}_2^2\mathbf{x}_1(14.9334\mathbf{x}_3 - 43.0934 + 3.3333\mathbf{x}_3^2) \\
&\quad + 0.7854(\mathbf{x}_5\mathbf{x}_7^2 + \mathbf{x}_4\mathbf{x}_6^2) - 1.508\mathbf{x}_1(\mathbf{x}_7^2 + \mathbf{x}_6^2) + 7.477(\mathbf{x}_7^3 + \mathbf{x}_6^3), \\
\text{s.t.} \quad & \begin{cases}
c_1(\mathbf{x}) = -\mathbf{x}_1\mathbf{x}_2^2\mathbf{x}_3 + 27 & \leq 0 \\
c_2(\mathbf{x}) = -\mathbf{x}_1\mathbf{x}_2^2\mathbf{x}_3^2 + 397.5 & \leq 0 \\
c_3(\mathbf{x}) = -\mathbf{x}_2\mathbf{x}_6^4\mathbf{x}_3\mathbf{x}_4^{-3} + 1.93 & \leq 0 \\
c_4(\mathbf{x}) = -\mathbf{x}_2\mathbf{x}_7^4\mathbf{x}_3\mathbf{x}_5^{-3} + 1.93 & \leq 0 \\
c_5(\mathbf{x}) = 10\mathbf{x}_6^{-3}\sqrt{16.91 \times 10^6 + (745\mathbf{x}_4\mathbf{x}_2^{-1}\mathbf{x}_3^{-1})^2} - 1100 & \leq 0 \\
c_6(\mathbf{x}) = 10\mathbf{x}_7^{-3}\sqrt{157.5 \times 10^6 + (745\mathbf{x}_5\mathbf{x}_2^{-1}\mathbf{x}_3^{-1})^2} - 850 & \leq 0 \\
c_7(\mathbf{x}) = \mathbf{x}_2\mathbf{x}_3 - 40 & \leq 0 \\
c_8(\mathbf{x}) = -\mathbf{x}_1\mathbf{x}_2^{-1} + 5 & \leq 0 \\
c_9(\mathbf{x}) = -\mathbf{x}_1\mathbf{x}_2^{-1} - 12 & \leq 0 \\
c_{10}(\mathbf{x}) = 1.5\mathbf{x}_6 - \mathbf{x}_4 + 1.9 & \leq 0 \\
c_{11}(\mathbf{x}) = 1.1\mathbf{x}_7 - \mathbf{x}_5 + 1.9 & \leq 0
\end{cases}
\end{aligned}
$$

We report numerical results of Section 4.3.3 and 4.3.4 in Figure 4.1 and Table 4.2.

4.3.5 Designing Sensitive Samples for Detection of Model Tampering

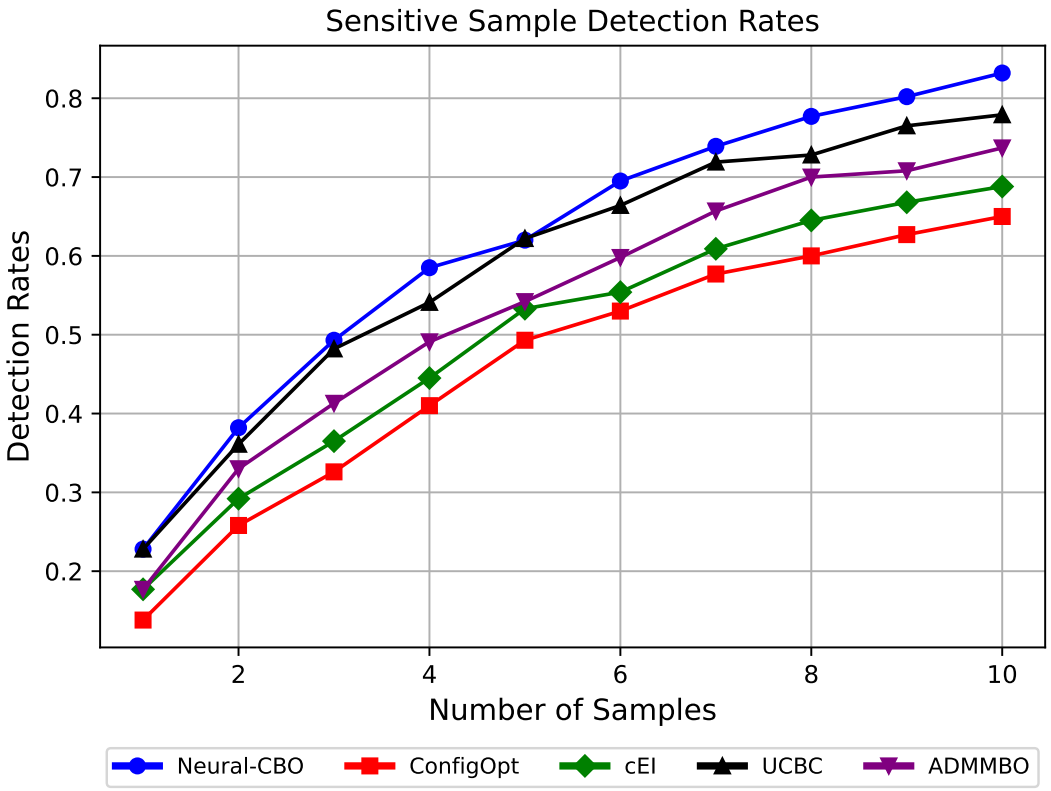


FIGURE 4.2: **Detection Rates** corresponding to the number of samples on the MNIST dataset. As shown in the figure, Neural-CBO can generate sensitive samples that achieve nearly 85% of the detection rate with at least 10 samples.

We build on the sensitive sample generation example described in Section 3.5.3.1 to address an additional practical challenge: incorporating constraints on the generated samples to ensure their realism. Specifically, in scenarios where tampered machine learning models are hosted on cloud services, attackers may attempt to bypass detection by modifying models while keeping their behavior on plausible inputs unchanged. To prevent attackers from evading detection, sensitive samples must resemble normal inputs. Therefore, a human-in-the-loop process is employed, where reviewers rate the realism of each sample on a scale of $(0, 1)$; higher scores indicate more realistic samples. These scores serve as constraints in the optimization, where obtaining human feedback can be costly. Assuming a pre-trained model $s_\varphi(\mathbf{x})$ may have been altered after being uploaded, the goal is to find sensitive samples by solving the optimization problem:

$$\begin{aligned} v = \underset{\mathbf{x}}{\operatorname{argmax}} & \left\| \frac{\partial s_\varphi(\mathbf{x})}{\partial \varphi} \right\|_F \\ \text{s.t. } & \mathit{realistic}(v) \geq h, \end{aligned}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, $realistic(\cdot)$ is the human-based scoring function and h is a pre-defined threshold. A detection is *successful* if at least one of the N_S sensitive samples shows a different top-1 prediction between the tampered and original models.

We employed the same experimental setup outlined in Section 3.5.3.1, using a pre-trained MNIST hand-written digit classification model and compared our method's performance against several baselines based on average detection rates for sensitive samples. The model was tampered with by adding noise to its weights 1,000 times, producing 1,000 distinct versions. While the original model had a top-1 accuracy of 93%, this dropped to $87.73\% \pm 0.08\%$ after tampering. To reduce computational costs, we downscaled the images from 28×28 to 7×7, optimized in this 49-dimensional space, and then restored them to the original resolution to generate sensitive samples. Feasible samples were chosen based on their realistic scores. Figure 4.2 shows the detection rates of (feasible) sensitive samples generated by our method compared to four baselines, demonstrating that our samples achieved higher detection rates. As expected, the detection rate improves with more samples and our method is consistently competitive.

4.4 Conclusion

We proposed a novel algorithm for black-box optimization with unknown constraints, utilizing deep neural networks as surrogate models for both the objective function and constraints. Our algorithm leverages the bounded nature of constraint values by applying LCB conditions at each iteration to ensure feasibility. We also employ EI as the acquisition function to balance exploration and exploitation, especially in scenarios where feasible regions are significantly smaller than the search space. Our theoretical analysis shows that, under mild conditions regarding neural network width, our algorithm achieves upper bounds on cumulative regret and constraint violations comparable to previous GPs-based methods. We validate our approach through experiments on synthetic and real-world benchmark tasks involving structural data, with results demonstrating competitive performance against state-of-the-art methods.

Chapter 5

PINN-BO: A Black-Box Optimization Algorithm Using Physics-Informed Neural Networks

In scientific and engineering problems, many objective functions are governed by Partial Differential Equations (PDEs), which capture fundamental physical laws that describe how systems change over time and space. For example, the heat equation explains how heat spreads in a given area over time, while the Navier-Stokes equations model the movement of fluids, accounting for factors like viscosity and external forces. PDEs also play a central role in areas such as structural analysis and electromagnetics, highlighting their widespread importance across various fields.

Recently, researchers have started incorporating PDEs knowledge into models of objective functions. For instance, Raissi, Perdikaris, and Karniadakis (2017) proposed a Gaussian Processes Regression (GPR) approach that uses a four-block covariance kernel to combine observations from the objective function and PDEs. Similarly, Jidling et al. (2017) introduced a global constraint for Gaussian Processes (GPs) using differential equations, which eliminates the computational challenges of the four-block kernel while providing stronger constraints. Later, Chen et al. (2021) developed a method to solve non-linear PDEs by treating their solutions as Maximum a Posteriori (MAP) estimators of Gaussian processes. Although these methods are effective, GPs face scalability issues because updating their posterior involves inverting a kernel matrix, which has cubic complexity as the dataset grows.

To overcome these limitations, Physics-Informed Neural Networks (PINNs) have been introduced (Raissi, Perdikaris, and Karniadakis, 2019; Yang, Meng, and Karniadakis, 2021). PINNs use neural networks to capture complex, nonlinear relationships, making them a more flexible and scalable alternative to GPs. Unlike GPs, PINNs can handle a wide range of PDEs, including non-linear equations, and are well-suited for large-scale problems in science and engineering. Recent studies have also explored the theoretical aspects of PINNs, showing how they can effectively incorporate PDE constraints (Schiassi et al., 2021; Wang et al., 2022). Additionally, researchers have investigated connections between PINNs and GPs, demonstrating their ability to model functions that satisfy PDE constraints (Wang, Yu, and Perdikaris, 2022).

PDEs offer valuable information that can significantly improve the modeling of black-box objective functions, enabling more sample-efficient optimization by reducing the need for expensive evaluations. However, there has been limited exploration of how to effectively use PDEs in black-box optimization, where the objective function is often unknown and noisy.

This chapter presents PINN-BO, a black-box optimization framework that employs a PINN to model the unknown function, incorporates physical knowledge expressed by PDEs constraints into the optimization process. Hence, PINN-BO aims to enhance model accuracy and improve constraint handling, particularly in problems governed by established physical laws. We summary **the contributions of this chapter** as follows:

- We introduce a novel black-box optimization problem with physics information, described by Partial Differential Equations (PDEs), which is used to govern the objective function.
- We propose PINN-BO, a black-box optimization algorithm employing Physics-Informed Neural Network with Partial Differential Equations (PDEs) induced by natural laws to perform efficient optimization, bringing several benefits: improved sample-efficiency of optimization, scalable computation that only grows linearly with the number of function evaluations, and the ability to incorporate a broad class of PDEs.
- We provide a theoretical analysis of our proposed PINN-BO algorithm to illustrate that incorporating linear PDEs can lead to $\mathcal{O}\left(\sqrt{T\gamma_T}\sqrt{\gamma_T - I(f;\mathbf{Y}_T;\mathbf{U}_r)}\right)$ regret, where T is the number of black-box function evaluations and $I(f;\mathbf{Y}_T;\mathbf{U}_r)$ is interaction information between the black-box function f, its observations $\mathbf{Y}_T$ and the PDE data $\mathbf{U}_r$ (see Section 5.3).
- We perform experiments with a variety of tasks showing that our algorithm outperforms current state-of-the-art black-box optimization methods.

This chapter begins by introducing the problem setting and give an example to illustrate our proposed setting, followed by a detailed description of the PINN-BO framework and its theoretical analysis.

5.1 Problem Setting

In this chapter, we consider a global optimization problem setting where the objective function $f\colon \mathcal{D} \to \mathbb{R}$ is associated with a PDE:

$$\min_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x}) \text{ s.t. } \mathcal{N}[f](\mathbf{x}) = g(\mathbf{x}),$$

where $\mathcal{D} \subset \mathbb{R}^d$ is a d-dimensional bounded domain and $\mathcal{N}[f]$ denotes a differential operator of the function f with respect to the input $\mathbf{x}$. The function f is an expensive, black-box function, and its evaluations are obtainable only through noisy measurements in the form of $y = f(\mathbf{x}) + \epsilon$, where ϵ represents sub-Gaussian noise. Additionally, the function $g(\mathbf{x})$ is a cheap-to-evaluate function, which may also involve noise, with respect to the PDE-constraint.

Remark 5.1.1. Considering the damped harmonic oscillator from classical physics as a real-world example where $f(x)$ is the displacement of the oscillator as a black-box function of time x, and measuring the displacement of a damped harmonic oscillator accurately can be expensive due to several factors, e.g., instrumentation cost, environmental factors, and calibration requirements. This displacement is governed by the PDE: $m\frac{d^2f}{dx^2} + c\frac{df}{dx} + kf = 0$, where $\mathcal{N}[f](x) = m\frac{d^2f}{dx^2} + c\frac{df}{dx} + kf$, $g(x) = 0$, m, c, k are mass, damping and spring values, respectively. Furthermore, it is assumed that the boundary conditions of the PDEs are either unknown or inaccessible.

These assumptions widely hold in many problem settings. As an example, Cai et al. (2020) examines a two-dimensional heat transfer problem with forced heat convection around a circular cylinder. The heat measurement entails high costs due to the material, size, and shape of the system. The problem has known incompressible Navier-Stokes and heat transfer equations. However, the thermal boundary conditions are difficult to ascertain precisely because of the complex and large instruments. The unavailability of these boundary conditions prevents the straightforward solution of the underlying function f using traditional numerical techniques.

5.2 Proposed PINN-BO Method

In this section, we present our proposed Physics-Informed Neural Network based Black-box Optimization (PINN-BO). PINN-BO algorithm combines optimization and machine learning techniques to efficiently optimize an unknown black-box function over a given input space while leveraging physics-informed constraints described by a PDE. Following the fundamental principles of Bayesian optimization, our algorithm consists of two primary steps: (1) constructing a model of the black-box objective function, and (2) employing this model to select the next function evaluation point during each iteration. In the first step, our approach diverges from traditional Bayesian Optimization algorithms that typically utilize Gaussian Processes (GPs) to model the objective function. Instead, we employ a fully connected neural network denoted as $h(\mathbf{x};\boldsymbol{\theta})$ to learn the function f as follows:

$$h(\mathbf{x};\boldsymbol{\theta}) = \frac{1}{\sqrt{m}}\mathbf{W}_L\psi(\mathbf{W}_{L-1}\psi(\cdots\psi(\mathbf{W}_1\mathbf{x})),$$

where $\psi\colon \mathbb{R}\to\mathbb{R}$ is a coordinate-wise smooth activation function (e.g., ReLU, Tanh), $\mathbf{W}_1\in\mathbb{R}^{m\times d}, \mathbf{W}_i\in\mathbb{R}^{m\times m}, 2\le i\le L-1, \mathbf{W}_L\in\mathbb{R}^{1\times m}$, and $\boldsymbol{\theta}\in\mathbb{R}^p$ is the collection of parameters of the neural network, $p = md + m^2(L-2) + m$ and d is the dimension of inputs, i.e., $\mathbf{x}\in\mathcal{D}\subset\mathbb{R}^d$. We initialize all the weights to be independent and identically distributed as standard normal distribution $\mathcal{N}(0,1)$ random variables. To leverage the information embedded within the PDE governing the objective function f, our algorithm generates a set of N_r PDE data points denoted as $\mathcal{R} = \{\mathbf{z}_j, u_j\}_{j=1}^{N_r}$. Here, u_j represents the noisy evaluations of the function g at the corresponding point $\mathbf{z}_j$, where $u_j = g(\mathbf{z}_j) + \eta_j$. Besides, we denote $\mathcal{D}_t = \{\mathbf{x}_i, y_i\}_{i=1}^{t}$ as the set of noisy observations of the unknown function f after t optimization iterations, where $y_t = f(\mathbf{x}_t) + \epsilon_t$. We further define some other notations:

$$\phi(\cdot) = \nabla_{\boldsymbol{\theta}} h(\cdot;\boldsymbol{\theta}_0); \quad \omega(\cdot) = \nabla_{\boldsymbol{\theta}}\mathcal{N}[h](\cdot;\boldsymbol{\theta}_0)$$

where $\phi(\cdot)$ is the gradient of $h(\cdot;\boldsymbol{\theta}_0)$ with respect to the parameter $\boldsymbol{\theta}$, evaluated at initialization $\boldsymbol{\theta}_0$. Similarly, $\omega(\cdot)$ represents the gradient of $\mathcal{N}[h](\cdot;\boldsymbol{\theta}_0)$ with respect to model parameters $\boldsymbol{\theta}$ and evaluated at initialization $\boldsymbol{\theta}_0$, where $\mathcal{N}[h](\cdot;\boldsymbol{\theta}_0)$ is the result of applying differential operator $\mathcal{N}$ (with respect to the input) to $h(\cdot;\boldsymbol{\theta}_0)$. Both $\mathcal{D}_t$ and $\mathcal{R}$ play an important role in the subsequent stages of the algorithm, specifically in the minimization of the loss function associated with learning the network $h(\mathbf{x},\boldsymbol{\theta}_t)$ at optimization iteration t:

$$\mathcal{L}(t) = \textstyle\sum_{i=1}^{t-1}[y_i - \nu_t h(\mathbf{x}_i;\boldsymbol{\theta}_{t-1})]^2 + \sum_{j=1}^{N_r}[u_j - \nu_t\mathcal{N}[h](\mathbf{z}_j;\boldsymbol{\theta}_{t-1})]^2, \quad (5.1)$$

where ν_t is a scale parameter that controls the exploration-exploitation trade-off.

For the second step, we employ a greedy strategy to pick the next sample point $\mathbf{x}_t$. At each iteration t, the algorithm updates the neural network by optimizing the loss function described in Eqn 5.1 by gradient descent, with scaled function value predictions $\nu_t h(\cdot;\boldsymbol{\theta}_{t-1})$ and scaled predictions with respect to the governed PDE $\nu_t \mathcal{N}[h](\cdot;\boldsymbol{\theta}_{t-1})$. In the proof of Section 5.3, we show this action is equivalent to placing the GP prior over function values $f_{1:t}$ and PDE values $g_{1:N_r}$ (Corollary C.2.1.1 in the Appendix C). Then the posterior distribution of the function prediction $\widetilde{f}_t(\mathbf{x}) = h(\mathbf{x},\boldsymbol{\theta}_{t-1})$ at a new data point $\mathbf{x}$ can be viewed as being sampled from a GP with specific posterior mean and variance function (Lemma 5.3.4). This allows us to directly use the network prediction as an acquisition function following the principle of Thompson Sampling. Then, the next evaluation point $\mathbf{x}_t$ is selected by minimizing this acquisition function $\widetilde{f}_t(\mathbf{x}) = h(\mathbf{x},\boldsymbol{\theta}_{t-1})$. Then, the black-box function is queried at point $\mathbf{x}_t$, resulting in a (noisy) observation y_t, which is subsequently used to update the dataset $\mathcal{D}_t$. The PDE observations set $\mathcal{R}$, in combination with the observations in $\mathcal{D}_t$, is integrated into the training process of the neural network by minimizing the squared loss, as described in Eqn 5.1. We provide a concise step-by-step summary of our approach in Algorithm 4.

To enhance the exploration step, it is important to bring the additional information to improve our model of objective function, especially in the regions where optima lies. In our case, this task of exploration is easier as we have access to PDE which provides knowledge about the objective function and reduces the amount of information that is needed to model the function. In Section 5.3 (Theoretical Analysis), we derive a scaling factor $\nu_t = B + \widetilde{R}\sqrt{2\gamma_t - 2I(f;\mathbf{Y}_t;\mathbf{U}_r) + \log\left(\frac{1}{\delta}\right)}$, which directly reflects this intuition. It reduces the maximum information gain (which can be thought of as the complexity of the function modeling) by the interaction information $I(f;\mathbf{Y}_t;\mathbf{U}_r)$, which is a generalization of the mutual information for three variables: unknown function f, its observations $\mathbf{Y}_t$, and the PDE data $\mathbf{U}_r$. This information can be calculated as $I(f;\mathbf{Y}_t;\mathbf{U}_r) = \frac{1}{2}\log\left(\frac{\det\left(\frac{\boldsymbol{\Phi}_t^\top\boldsymbol{\Phi}_t}{\lambda_1}+\mathbf{I}\right)\det\left(\frac{\boldsymbol{\Omega}_r^\top\boldsymbol{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}{\det\left(\frac{\boldsymbol{\Phi}_t^\top\boldsymbol{\Phi}_t}{\lambda_1}+\frac{\boldsymbol{\Omega}_r^\top\boldsymbol{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}\right)$, where $\boldsymbol{\Phi}_t = [\phi(\mathbf{x}_1)^\top,\ldots,\phi(\mathbf{x}_t)^\top]^\top$ and $\boldsymbol{\Omega}_r = [\omega(\mathbf{z}_1)^\top,\ldots,\omega(\mathbf{z}_{N_r})^\top]^\top$. The value of ν_t signifies how our algorithm continues the exploration in regions of search space where the function f has no implicit knowledge through PDE observations. These are the regions indicated by $\gamma_t - I(f;\mathbf{Y}_t;\mathbf{U}_r)$, which is the amount of information about the unknown function f remains after our algorithm interacts with PDE data $\mathbf{U}_r$.

Algorithm 4 Physics-informed Neural Network based Black-box optimization (PINN-BO)

Input: The input space $\mathcal{D}$, the optimization budget T, PDE training set size N_r, $\delta \in (0,1)$, parameters $B, R_1, R_2, \lambda_1, \lambda_2$ (see Assumption 5.3.2 and 5.3.3 in Section 5.3).

1: Initialize $\mathcal{D}_0 = \emptyset$ and $\boldsymbol{\theta}_0 \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
2: Generate set $\mathcal{R} = \{\mathbf{z}_j, u_j\}_{j=1}^{N_r}$ from the PDE.
3: **for** $t = 1$ to T **do**
4: Set $\nu_t = B + \widetilde{R}\sqrt{2\gamma_t - 2I(f;\mathbf{Y}_t;\mathbf{U}_r) + \log\left(\frac{1}{\delta}\right)}$, where $\widetilde{R} = \sqrt{\left(\frac{R_1}{\lambda_1}\right)^2 + \left(\frac{R_2}{\lambda_2}\right)^2}$.
5: $\widetilde{f}_t(\mathbf{x}) = h(\mathbf{x};\boldsymbol{\theta}_{t-1})$
6: Choose $\mathbf{x}_t = \operatorname{argmin}_{\mathbf{x}\in\mathcal{D}} \widetilde{f}_t(\mathbf{x})$ and receive observation $y_t = f(\mathbf{x}_t) + \epsilon_t$
7: Update $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t, y_t\}$
8: Update $\boldsymbol{\theta}_t = \operatorname{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ using Eqn. 5.1 by gradient descent with $\nu = \nu_t$.
9: **end for**

5.3 Theoretical Analysis

In this section, we provide a regret bound for the proposed PINN-BO algorithm. As presented in Section 2.3.6, to quantify the algorithm's regret, we employ the cumulative regret, defined as $R_T = \sum_{t=1}^{T} r_t$ after T iterations. Here, $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x})$ represents the optimal point of the unknown function f, and $r_t = f(\mathbf{x}^*) - f(\mathbf{x}_t)$ denotes the instantaneous regret incurred at time t. Our regret analysis is built upon the recent NTK-based theoretical work of Wang, Yu, and Perdikaris (2022) and proof techniques of GP-TS Chowdhury and Gopalan (2017). Before proceeding with the theoretical analysis, we now introduce a set of definitions and assumptions. They clarify our proof and set up the basis and conditions for our analysis. **A detailed proof can be found in Section 5.3 of the Appendix.**

Definition 5.3.1. We define matrix $\mathbf{K}_{\text{NTK}-\text{PINN}}$ as the *neural tangent kernel of a Physics-Informed Neural Network (NTK of PINNs)*:

$$\mathbf{K}_{\text{NTK}-\text{PINN}} = \begin{bmatrix} \mathbf{K}_{uu} & \mathbf{K}_{ur} \\ \mathbf{K}_{ru} & \mathbf{K}_{rr} \end{bmatrix}, \tag{5.2}$$

where $(\mathbf{K}_{uu})_{ij} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, $(\mathbf{K}_{ur})_{ij} = \langle \phi(\mathbf{x}_i), \omega(\mathbf{z}_j) \rangle$, $(\mathbf{K}_{rr})_{ij} = \langle \omega(\mathbf{z}_i), \omega(\mathbf{z}_j) \rangle$ and $\mathbf{K}_{ru} = \mathbf{K}_{ur}^{\top}$, defined using $\mathcal{D}_t$ and $\mathcal{R}$.

Assumption 5.3.2. We assume the noises $\{\epsilon_i\}_{i=1}^{T}$ where $\epsilon_i = y_i - f(\mathbf{x}_i)$ and $\{\eta_j\}_{j=1}^{N_r}$ where $\eta_j = u_j - g(\mathbf{z}_j)$ are conditionally sub-Gaussian with parameter $R_1 > 0$ and $R_2 > 0$, where $\{\epsilon_i\}_{i=1}^{T}$ and $\{\eta_j\}_{j=1}^{N_r}$ is assumed to capture the noises induced by querying the black-box, expensive function $f(\cdot)$ and cheap-to-evaluate PDE-related function $g(\cdot)$ respectively.

$$\forall i \geq 0, \forall \lambda_1 \in \mathbb{R},\ \mathbb{E}[e^{\lambda_1 \epsilon_i} | \mathcal{F}_{t-1}] \leq e^{\frac{\lambda_1^2 R_1^2}{2}}$$
$$\forall j \geq 0, \forall \lambda_2 \in \mathbb{R},\ \mathbb{E}[e^{\lambda_2 \eta_j} | \mathcal{F}'_{N_r-1}] \leq e^{\frac{\lambda_2^2 R_2^2}{2}}$$

where $\mathcal{F}_{t-1}, \mathcal{F}'_{N_r-1}$ are the σ-algebra generated by the random variables $\{\mathbf{x}_i, \epsilon_i\}_{i=1}^{t-1} \cup \{\mathbf{x}_t\}$ and $\{\mathbf{z}_j, \eta_j\}_{j=1}^{N_r-1} \cup \{\mathbf{z}_{N_r}\}$, respectively.

Assumption 5.3.3. We assume f to be an element of the Reproducing Kernel Hilbert Space (RKHS) associated with real-valued functions defined on the set $\mathcal{D}$ (For more details about RKHS, see Section 2.1.5). This specific RKHS corresponds to the Neural Tangent Kernel (NTK) of a physics-informed neural network (NTK-PINN) and possesses a bounded norm denoted as $\|f\|_{\mathcal{H}_{k_{\text{NTK-PINN}}}} \leq B$. Formally, this RKHS is denoted as $\mathcal{H}_{k_{\text{NTK-PINN}}}(\mathcal{D})$, and is uniquely characterized by its kernel function $k_{\text{NTK-PINN}}(\cdot,\cdot)$. The RKHS induces an inner product $\langle \cdot, \cdot \rangle$ that obeys the reproducing property: $f(\mathbf{x}) = \langle f, k_{\text{NTK-PINN}}(\cdot, \mathbf{x}) \rangle$ for all $f \in \mathcal{H}_{k_{\text{NTK-PINN}}}(\mathcal{D})$. The norm induced within this RKHS, $\|f\|_{\mathcal{H}_{k_{\text{NTK-PINN}}}} = \sqrt{\langle f, f \rangle_{\mathcal{H}_{k_{\text{NTK-PINN}}}}}$, quantifies the smoothness of f concerning the kernel function $k_{\text{NTK-PINN}}$, and satisfies: $f \in \mathcal{H}_{k_{\text{NTK-PINN}}}(\mathcal{D})$ if and only if $\|f\|_{k_{\mathcal{H}_{\text{NTK-PINN}}}} < \infty$.

Assumptions 5.3.2 and 5.3.3 represent commonly employed and well-established assumptions in GP-based Bandits and Bayesian Optimization (Chowdhury and Gopalan, 2017; Vakili et al., 2021). We are now prepared to establish an upper bound on the regret incurred by our proposed PINN-BO algorithm.

We begin by presenting key lemmas for establishing the regret bound in Theorem 5.3.11 of the proposed algorithms. The following lemma demonstrates that, given the assumption of an infinitely wide network, the output of the trained physics-informed neural network used to model the unknown function f governed by a linear PDE, after running t optimization iterations in Algorithm 4, can be regarded as sampling from a GP with specific mean and covariance functions.

Lemma 5.3.4. *Conditioned on* $\mathcal{D}_t = \{\mathbf{x}_i, y_i\}_{i=1}^t, \mathcal{R} = \{\mathbf{z}_j, u_j\}_{j=1}^{N_r}$, *the acquisition function* $\widetilde{f}_t(\mathbf{x}) = h(\mathbf{x}; \boldsymbol{\theta}_{t-1})$ *can be viewed as a random draw from a* GP $\left(\mu_t^f(\mathbf{x}), \nu_t^2 \left(\sigma_t^f\right)^2(\mathbf{x})\right)$ *with the following mean and covariance functions:*

$$\mu_t^f(\mathbf{x}) = \phi(\mathbf{x})^\top \boldsymbol{\xi}_t^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \begin{bmatrix} \mathbf{Y}_t \\ \mathbf{U}_r \end{bmatrix}$$
$$\left(\sigma_t^f\right)^2(\mathbf{x}) = \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - \phi(\mathbf{x})^\top \boldsymbol{\xi}_t^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \boldsymbol{\xi}_t \phi(\mathbf{x}),$$

where

$$\widehat{\mathbf{K}}_{\text{PINN}}^{-1} = \begin{bmatrix} \mathbf{K}_{uu} + \lambda_1 \mathbf{I} & \mathbf{K}_{ur} \\ \mathbf{K}_{ru} & \mathbf{K}_{rr} + \lambda_2 \mathbf{I} \end{bmatrix}^{-1} = \begin{bmatrix} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} \\ \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} \end{bmatrix}$$
$$\boldsymbol{\Phi}_t = [\phi(\mathbf{x}_1)^\top, \ldots, \phi(\mathbf{x}_t)^\top]^\top$$
$$\boldsymbol{\Omega}_r = [\omega(\mathbf{z}_1)^\top, \ldots, \omega(\mathbf{z}_{N_r})^\top]^\top, \ \boldsymbol{\xi}_t = \begin{bmatrix} \boldsymbol{\Phi}_t^\top & \boldsymbol{\Omega}_r^\top \end{bmatrix}^\top$$
$$\mathbf{K}_{uu} = \boldsymbol{\Phi}_t \boldsymbol{\Phi}_t^\top, \ \mathbf{K}_{ur} = \boldsymbol{\Phi}_t \boldsymbol{\Omega}_r^\top, \mathbf{K}_{ru} = \mathbf{K}_{ur}^\top, \mathbf{K}_{rr} = \boldsymbol{\Omega}_r \boldsymbol{\Omega}_r^\top$$
$$\mathbf{Y}_t = [y_1, y_2, \ldots, y_t]^\top, \mathbf{U}_r = [u_1, u_2, \ldots, u_{N_r}]^\top$$

Generic BO methods (without the extra information from PDE) utilized the *maximum information gain* over search space $\mathcal{D}$ at time t: $\gamma_t := \max_{\mathcal{A} \subset \mathcal{D}: |\mathcal{A}| = t} I(\mathbf{Y}_\mathcal{A}, f_\mathcal{A})$, where $I(\mathbf{Y}_\mathcal{A}, f_\mathcal{A})$ denotes the mutual information between $f_\mathcal{A} = [f(\mathbf{x})]_{\mathbf{x} \in \mathcal{A}}$ and noisy observations $\mathbf{Y}_\mathcal{A}$, which quantifies the reduction in uncertainty about the objective function f after observing y_A. The maximum information gain is the fundamental component when analyzing regret bound for their algorithm (Srinivas et al., 2009; Vakili et al., 2021). However, in our work, the PDE evaluations $\{u_j\}_{j=1}^{N_r}$ of the function g is considered as the second source of information that contributes to reduce the uncertainty of f. Therefore, we introduce the *interaction information* as the generalization of *mutual information* for three random variables:

Definition 5.3.5. The **interaction information** between f, its observations $\mathbf{Y}_\mathcal{A}$, (where $\mathcal{A} \subset \mathcal{D}$), and the PDE data $\mathbf{U}_r$ can be defined as:

$$I(f; \mathbf{Y}_\mathcal{A}; \mathbf{U}_r) = I(f; \mathbf{Y}_\mathcal{A}) - I(f; \mathbf{Y}_\mathcal{A} | \mathbf{U}_r),$$

where $I(f; \mathbf{Y}_\mathcal{A})$ quantifies the reduction in uncertainty in f due to observing $\mathbf{Y}_\mathcal{A}$, while $I(f; \mathbf{Y}_\mathcal{A} | \mathbf{U}_r)$ represents the additional information contributed by $\mathbf{U}_r$ to enhance the mutual information between f and $\mathbf{Y}_\mathcal{A}$.

The next lemma provides the closed-form expression of the interaction information.

Lemma 5.3.6. *The interaction information between f and observation $\mathbf{Y}_t$ and PDE data $\mathbf{U}_r$, for the points chosen from Algorithm 4 can be calculated as:*

$$I(f;\mathbf{Y}_t;\mathbf{U}_r) = \frac{1}{2}\log\left(\frac{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)\det\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}\right)$$

Remark 5.3.7. Following Remark 3.3 in Wang, Yu, and Perdikaris (2022), both matrices $\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}$ and $\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}$ are positive semi-definite. It can be clearly seen that the interaction information given in Lemma 5.3.6 is non-negative:

$$\begin{aligned} I(f;\mathbf{Y}_t;\mathbf{U}_r) &= \frac{1}{2}\log\left(\frac{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)\det\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}\right) \\ &= \frac{1}{2}\log\left(\frac{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}+\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_1\lambda_2}\right)}{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}\right) \geq 0 \end{aligned}$$

The inequality uses the identity $\det(\mathbf{A}+\mathbf{B}) \geq \det(\mathbf{A})$, where $\mathbf{A},\mathbf{B}$ are two positive semi-definite matrices.

Our next result shows how the prediction of the neural network model is concentrated around the unknown reward function f, which is the key to a tighter regret bound.

Lemma 5.3.8. *Assume that $\|\omega(\cdot)\|_2 \leq L$, where $\omega(\cdot) = \nabla_{\boldsymbol{\theta}}\mathcal{N}[h](\cdot;\boldsymbol{\theta}_0)$ and $\rho_{min}(\mathbf{K}_{uu})$ the smallest eigenvalue of kernel matrix $\mathbf{K}_{uu}$ defined in lemma 1. Set $N_r = c_r\left(1+\frac{\rho_{min}(\mathbf{K}_{uu})}{\lambda_1}\right)/L^2$ for a positive constant c_r. Under the same hypotheses as stated in Assumption 5.3.2 and Assumption 5.3.3, and denote $\widetilde{R} = \sqrt{\left(\frac{R_1}{\lambda_1}\right)^2+\left(\frac{R_2}{\lambda_2}\right)^2}$. Let $\delta \in (0,1)$. Then, with probability at least $1-\delta$, the following confidence bound holds for all $\mathbf{x} \in \mathcal{D}$ and $t \geq 1$:*

$$\begin{aligned} &|f(\mathbf{x}) - \mu_t^f(\mathbf{x})| \\ &\leq \sigma_t^f(\mathbf{x})\left(B + \widetilde{R}\sqrt{2I(f;\mathbf{Y}_t) - 2I(f;\mathbf{Y}_t;\mathbf{U}_r) + \mathcal{O}(1) + \log(1/\delta)}\right) \\ &\leq \sigma_t^f(\mathbf{x})\left(B + \widetilde{R}\sqrt{2\gamma_t - 2I(f;\mathbf{Y}_t;\mathbf{U}_r) + \mathcal{O}(1) + \log(1/\delta)}\right) \end{aligned}$$

Proof sketch for Lemma 5.3.8 We split the problem into two terms: The prediction error of an element f in the RKHS as assumed in Assumption 5.3.3 with noise-free observations and the noise effect. Our proof differs from most GP-based Bayesian Optimization methods, which use single-block kernel matrices. In contrast, our predictive mean and covariance function involve the inversion of a block matrix $\widehat{\mathbf{K}}_{\text{PINN}}^{-1}$, as stated in Lemma 5.3.4. We employ the block matrix inversion formula (see Appendix A, (Rasmussen, Williams, et al., 2006)) to express $\widehat{\mathbf{K}}_{\text{PINN}}^{-1}$ as four distinct matrices. Subsequently, using equivalent transformations, intermediate matrix identities, and utilizing the expression *Interaction information* provided in Lemma 5.3.6, we derive the final bound.

Remark 5.3.9. The upper bound of the confidence interval presented in Lemma 5.3.8 shares a similar form with the existing confidence interval of GP-TS as outlined in

Chowdhury and Gopalan (2017). It is worth emphasizing, however, that our bound offers valuable insights into the significance of integrating PDEs to attain a tighter confidence bound. This insight can be summarized as follows: The expression $I(f;\mathbf{Y}_t) - I(f;\mathbf{Y}_t;\mathbf{U}_r)$ equals to $I(f;\mathbf{Y}_t|\mathbf{U}_r)$, which represents the expected mutual information between the function f and the observations $\mathbf{Y}_t$, given $\mathbf{U}_r$. Lemma 5.3.6 quantifies $I(f;\mathbf{Y}_t;\mathbf{U}_r)$ in terms of the kernel Gram matrices induced by black-box function and the PDE observations. As mentioned in Remark 5.3.7, the condition $I(f;\mathbf{Y}_t;\mathbf{U}_r) \geq 0$ implies that $I(f;\mathbf{Y}_t) \geq I(f;\mathbf{Y}_t|\mathbf{U}_r)$. This inequality signifies that knowing the values of the PDE component $\mathbf{U}_r$ reduces the statistical information between the observations $\mathbf{Y}_t$ and the unknown function f. In other words, knowing the values of PDE component $\mathbf{U}_r$ can diminish the number of observations $\mathbf{Y}_t$ required to estimate the unknown function f.

Remark 5.3.10. The value of $I(f;\mathbf{Y}_t;\mathbf{U}r)$ depends on the specific problem. For instance, if we assume that f is a function in RKHS with a linear kernel and $\mathcal{N}[f] = \sum_{i=1}^{n} \frac{\partial f}{\partial \mathbf{x}_i}$, the lower bound for the interaction information is:

$$I(f;\mathbf{Y}_t;\mathbf{U}_r) = \Theta\left(\frac{dN_r}{dN_r+1}(1-1/T)\right) = \Theta(1),$$

which is a constant. This is because the linear kernel differential feature map sends all PDE points to the same vector in the RKHS. This example aims to show how to bound the interaction information for a known PDE.

We are now ready to present the main theoretical result of the paper:

Theorem 5.3.11. *Let* $\mathbf{K}_{uu}, \mathbf{K}_{ur}, \mathbf{K}_{ru}$, *and* $\mathbf{K}_{rr}$ *be four matrices as defined in Lemma 5.3.4. Let* $\delta \in (0,1)$. *Assume that* $\|\omega(\cdot)\|_2 \leq L$ *and* $\rho_{min}(\mathbf{K}_{uu})$ *be the smallest eigenvalue of kernel matrix* $\mathbf{K}_{uu}$. *Set* $N_r = c_r\left(1 + \frac{\rho_{min}(\mathbf{K}_{uu})}{\lambda_1}\right)/L^2$ *for a constant* $c_r > 0$. *Additionally, let* $\widetilde{R} = \sqrt{\left(\frac{R_1}{\lambda_1}\right)^2 + \left(\frac{R_2}{\lambda_2}\right)^2}$ *and* $I_0 = \frac{1}{2}\log\frac{\det(\mathbf{K}_{rr}+\lambda_2\mathbf{I})}{\det\left(\mathbf{K}_{rr}+\lambda_2\mathbf{I}-\mathbf{K}_{ru}\mathbf{K}_{uu}^{-1}\mathbf{K}_{ur}\right)}$. *Then with probability at least* $1-\delta$, *the regret of PINN-BO running for an unknown function* f *governed by a linear PDE, lying in the* $\mathcal{H}_{k_{\text{NTK-PINN}}}$, $\|f\|_{H_{k_{\text{NTK-PINN}}}} \leq B$ *as stated in Assumption 5.3.3, after* T *iterations satisfies:*

$$R_T = \mathcal{O}\Bigg(\sqrt{Td\log BdT}\Bigg[B\sqrt{\gamma_T - I_0 + \log(2/\delta)} + \widetilde{R}\sqrt{\gamma_T}\sqrt{\gamma_T - I(f;\mathbf{Y}_T;\mathbf{U}_r) - I_0 + \log(2/\delta)}\Bigg]\Bigg)$$

5.4 Experiments

In this section, we demonstrate the effectiveness of our proposed PINN-BO algorithm through its application of synthetic benchmark optimization functions as well as real-world optimization problems. Our implementations of both problems are available at: https://github.com/phantrdat/pinn-bo.

5.4.1 Experimental Setup

For all experiments, we compared our algorithm with common classes of surrogate models used in black-box optimization, including Gaussian Processes (GPs)

and Deep Neural Networks (DNNs). For GPs, we employ the most popular strategy GP-EI (Mockus, Tiesis, and Zilinskas, 1978) and GP-UCB (Srinivas et al., 2009) with the Matérn Kernel. Our implementations for GP-based Bayesian Optimization baselines utilize public library GPyTorch `https://gpytorch.ai/` and BOTorch `https://botorch.org/`. We also include two recent DNNs-based works for black-box optimization: Neural Greedy (Paria et al., 2022) and Neural-BO (Phan-Trong, Tran-The, and Gupta, 2023) described below:

- Neural Greedy in Paria et al. (2022) fits a neural network to the current set of observations, where the function values are randomly perturbed before learning the neural network. The learned neural network is then used as the acquisition function to determine the next query point. Since the NeuralGreedy code is not publicly available, we use our own implementation following the setting described in Paria et al. (2022) (see Appendix F.2 therein).
- Neural-BO in Phan-Trong, Tran-The, and Gupta (2023) utilizes the Thompson Sampling strategy for selecting the next evaluation point. In this approach, the mean function is estimated using the output of a fully connected deep neural network. To implement this baseline, we adhere to the configuration outlined in Section 7 in Phan-Trong, Tran-The, and Gupta (2023).

For our proposed PINN-BO algorithm, we employ a fully connected deep neural network (DNN) as the surrogate model. The network's weights are initialized with independent samples drawn from a normal distribution $\mathcal{N}(0, 1)$. The model's hyper-parameters, which include depth, width, and learning rate, are selected as follows: For each function, we perform a grid search for tuning, where each hyper-parameter tuple is trained with 50 initial points. The width is explored within the set $\{100, 200, 500\}$, while the depth and the learning rate are searched across the values $\{2, 3, 4\}$ and $\{0.001, 0.005, 0.01, 0.02, 0.05, 0.1\}$, respectively. Subsequently, we select the tuple of (depth, width, learning rate) associated with the lowest mean-square error during evaluation. To train the surrogate neural network models, we utilize the (stochastic) gradient descent optimizer along with an Exponential Learning Rate scheduler with a factor of $\gamma = 0.95$. To accelerate the training process, we update the parameters $\boldsymbol{\theta}_t$ of surrogate models in Algorithm 4 after every 10 optimization iterations with 100 epochs.

5.4.2 Synthetic Benchmark Functions

We conducted optimization experiments on five synthetic functions: DropWave (2), Styblinski-Tang (10), Rastrigin (20), Michalewics (30), and Cosine Mixture (50), where the numbers in parentheses indicate the input dimensions of each function. We selected them to ensure a diverse range of difficulty levels, as suggested by the difficulty rankings available at `https://infinity77.net/global_optimization/test_functions.html`. To enhance the optimization process, we incorporated the PDEs associated with these objective functions. The detailed expressions of these functions and their corresponding PDEs can be found in Section C.1.1 of the Appendix C. Additionally, the noise in function evaluations follows a normal distribution with zero mean, and the variance is set to 1% of the function range. Results for Rastrigin, Michalewics, and Cosine Mixture functions are presented in Figure 5.1 and Figure . All experiments reported here are averaged over 10 runs, each with random initialization. All methods begin with the same initial points. The

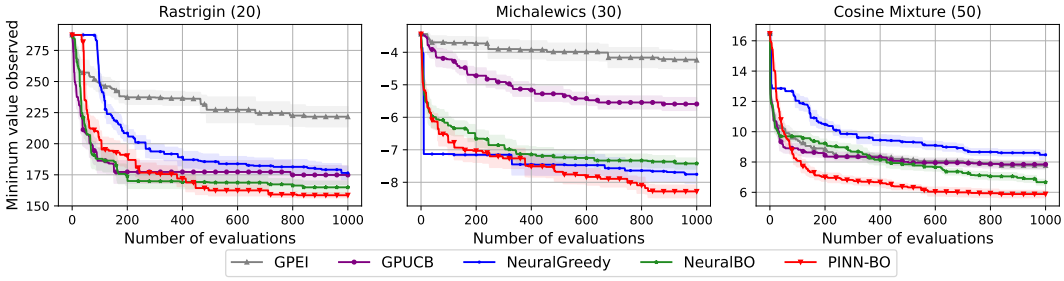


FIGURE 5.1: The optimization results for Rastrigin, Michalewics and Cosine Mixture functions comparing the proposed PINN-BO with the baselines. The standard errors are shown by color shading.

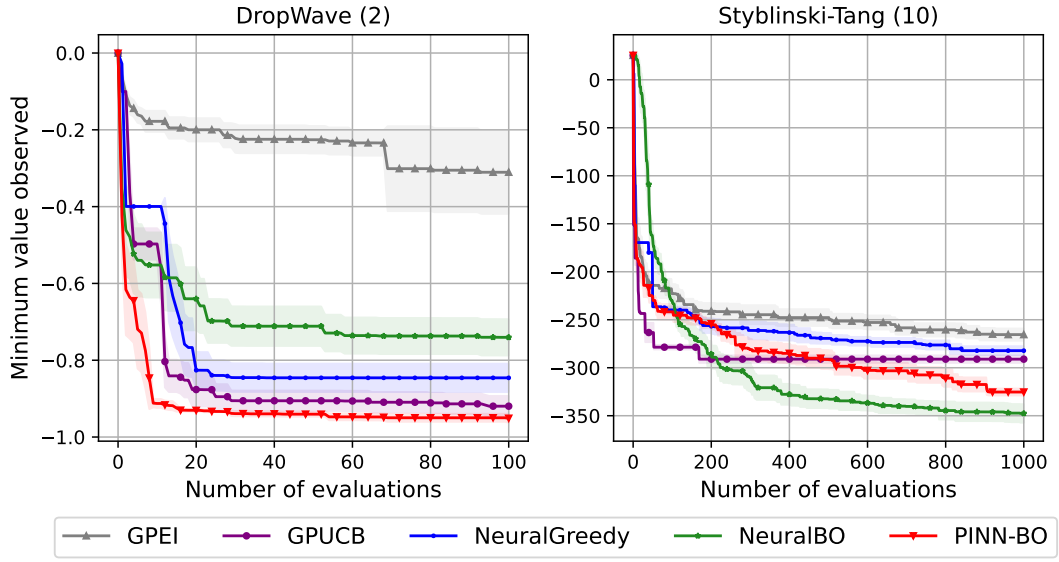


FIGURE 5.2: The optimization results for DropWave and Styblinski-Tang functions comparing the proposed PINN-BO with the baselines. The standard errors are shown by color shading.

results demonstrate that our PINN-BO is better than all other baseline methods, including GP-based BO algorithms (GP-EI, GP-UCB), and NN-based BO algorithms (NeuralBO, NeuralGreedy).

5.4.3 Real-world Applications

In this section, we explore two real-world applications where the objective functions are constrained by specific PDEs. We consider two tasks: (1) optimizing the Steady-State temperature distribution, satisfying the Laplace equation, and (2) optimizing the displacement of a beam element, adhering to the non-uniform Euler-Bernoulli equation. We continue to compare our proposed method with the baselines mentioned in Section 5.4.1.

5.4.3.1 Optimizing Steady-State Temperature

In this study, we showcase the benchmark optimization outcomes achieved by our proposed PINN-BO algorithm, comparing them with baseline methods for the

steady-state temperature optimization task. The steady-state heat equation represents a special case of the heat equation when the temperature distribution no longer changes over time. It describes the equilibrium state of a system where the temperature is constant, and no heat is being added or removed. The governing PDE for the temperature distribution is expressed as: $\nabla^2 T(x,y) = 0$, where x, y are spatial variables that represent the positions within a two-dimensional space. We explore the heat equation in a domain where x and y lie within the defined range of $[0, 2\pi]$. To thoroughly investigate the problem, we consider three different heat equations, where the solution of each problem is associated with one (unknown) boundary condition, each contributing to a deeper understanding of the system:

Heat Equation with Boundary Conditions 1

$$\begin{aligned} T(x,0) &= 5\sin(y) + \sqrt{1+y} \\ T(x,2\pi) &= y\sin\left(3\cos(y) + 2\exp(y)\sin(y)\right) \\ T(0,y) &= 10\cos(x) + x\exp\left(\sqrt{x^2+\sin(x)}\right) \\ T(2\pi,y) &= 3\sqrt{\exp(x\exp(-x))}\sin(x) + \cos(3x)\cos(3x) \end{aligned}$$

Heat Equation with Boundary Conditions 2

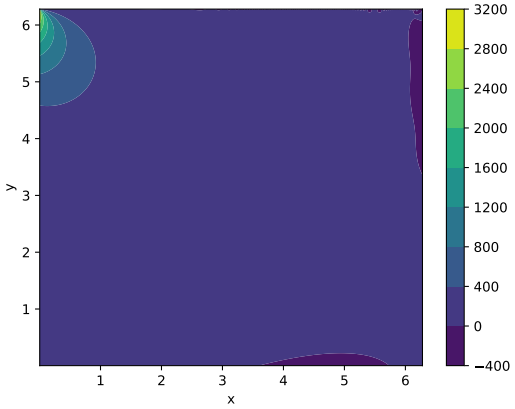
$$\begin{aligned} T(x,0) &= \sin(x)\cos(2x) + x^2\sqrt{3x} + e^{\sin(x)} \\ T(x,2\pi) &= e^{\sin(x)}\sqrt{3x} + x^2\cos(x)\sin^2(x) + e^{\cos(x)} \\ T(0,y) &= \sqrt{2y}\sin(y) + y^3\cos(2y) + e^{\cos(y)} \\ T(2\pi,y) &= \sin(y)\cos(2y) + y^3\sqrt{2y} + e^{\sin(y)} \end{aligned}$$

Heat Equation with Boundary Conditions 3

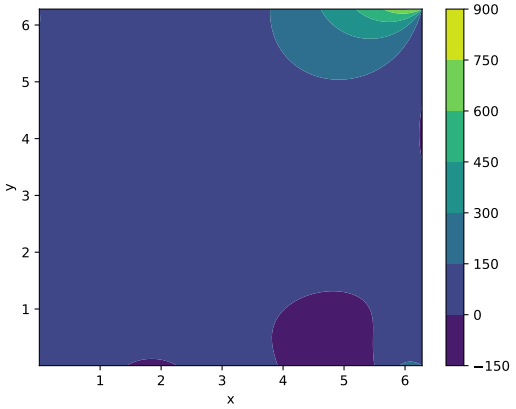
$$\begin{aligned} T(x,0) &= \left(\sin(x) + \cos(2x)\right)\sqrt{3x} + x^2 + e^{\sin(x)} \\ T(x,2\pi) &= \left(e^{\sin(x)} + \sqrt{3x}\right)\cos(x) + \left(\sin^2(x) + x^2\right)e^{\cos(x)} \\ T(0,y) &= \left(\sqrt{2y} + \sin(y)\right)\left(\cos(2y) + y^3\right) + e^{\cos(y)} \\ T(2\pi,y) &= \left(\sin(y) + \cos(2y)\right)\left(\sqrt{2y} + y^3\right) + e^{\sin(y)} \end{aligned}$$

We utilized `py-pde`, a Python package designed for solving partial differential equations (PDEs), available at the following GitHub repository: `https://github.com/zwicker-group/py-pde`. This tool enabled us to obtain solutions to heat equations at various input points, with specific boundary conditions serving as the input data. In Figure 5.3, the temperature distribution within the defined domain $[0, 2\pi]$ is visualized. It is important to emphasize that the boundary conditions used for benchmarking purposes are unknown to the methods employed. Adhering to the framework of black-box optimization, we assume that solving the PDEs incurs a substantial computational cost. The figure illustrates the spatial distribution of temperature values $T(x,y)$ across domain $[0, 2\pi] \times [0, 2\pi]$, with each subfigure corresponding to one of the mentioned boundary conditions.

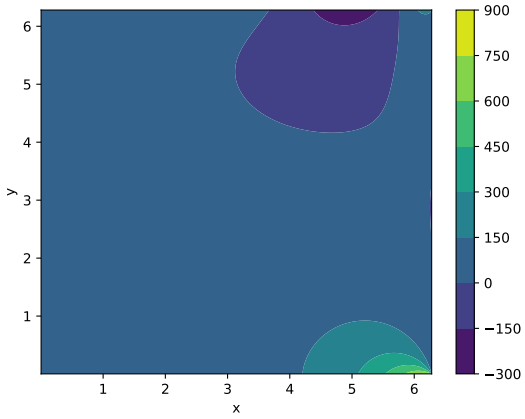
We conducted temperature optimization by identifying the locations (x,y) where the temperature reaches its maximum. As illustrated in Figure 5.3, the area with high



(A) Solution of Temperature Equation with boundary conditions 1



(B) Solution of Temperature Equation with boundary conditions 2



(C) Solution of Temperature Equation with boundary conditions 3

FIGURE 5.3: The figures depict the solutions for temperature distributions governed by the heat equation, with each figure corresponding to a specific tuple of boundary conditions described in Section 5.4.3.1. It is evident that the region with the highest temperature is relatively small in comparison to the entire domain.

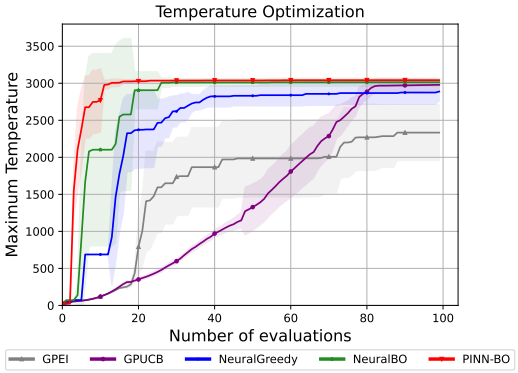
temperatures is relatively small in comparison to the regions with medium or low temperatures. For each baseline, we performed the optimization process 10 times, computing the average results. The comparative outcomes are presented in Figure 5.4.

5.4.3.2 Optimizing Beam Displacement

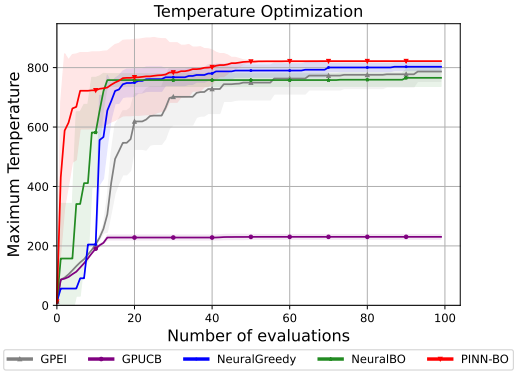
We present the benchmark optimization outcomes obtained through our proposed method, PINN-BO, and the baseline approaches, addressing the task of minimizing the deflection of a non-uniform Euler-Bernoulli beam. The governing differential equation describing the behavior of a non-uniform Euler-Bernoulli beam is provided below:

$$\frac{d^2}{dx^2}\left(EI(x)\frac{d^2w(x)}{dx^2}\right) = q(x),$$

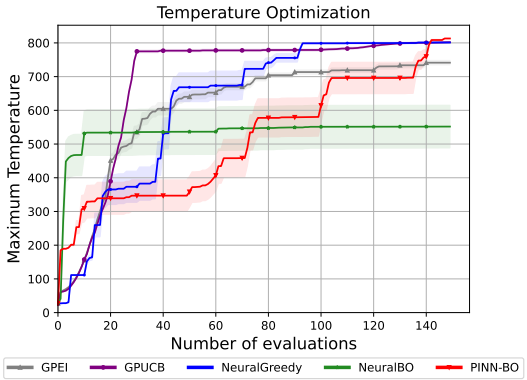
where $EI(x)$ represents the flexural rigidity of the beam, which can vary with position x, and $w(x)$ represents the vertical displacement of the beam at position x and



(A) Temperature Optimization in case of boundary conditions 1



(B) Temperature Optimization in case of boundary conditions 2



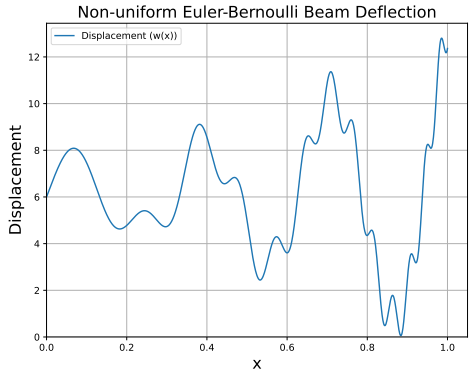
(C) Temperature Optimization in case of boundary conditions 3

FIGURE 5.4: The figure shows the temperature optimization results of our PINN-BO and other baselines. For all three cases with different positions of maximum temperature, our PINN-BO performs better than all other baselines.

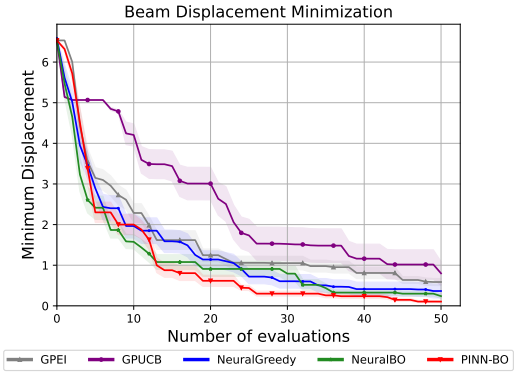
$q(x)$ represents the distributed or concentrated load applied to the beam. In our implementation, we consider the detailed expression of $\mathrm{EI}(x)$ and $q(x)$ as follows:

$$\begin{aligned}\mathrm{EI}(x) &= \frac{e^x}{\rho(x)},\\ \rho(x) &= 2.4x - 64\pi^2 e^{4x}\sin\left(4\pi e^{2x}\right) - 396e^{2x}\sin\left(20x\right)\\ &\quad + 80e^{2x}\cos\left(20x\right) + 16\pi e^{2x}\cos\left(4\pi e^{2x}\right) + 0.4\end{aligned}$$

We employed the Finite Difference Method (FDM) to solve the non-uniform Euler-Bernoulli beam equation. It's crucial to note that this step is solely for generating observations at each input point. Despite obtaining this solution, our methods and all baseline techniques continue to treat this solution as a black-box function, accessing observations solely through querying. In Figure 5.5a, the displacement values $w(x)$ for $x \in (0,1)$ are illustrated. The optimization results for both our PINN-BO and the other baseline methods are presented in Figure 5.5b.



(A) Displacement of non-uniform Euler beam



(B) Displacement Optimization of non-uniform Euler beam

FIGURE 5.5: Displacement of a non-uniform Euler Beam and minimum displacement found by our PINN-BO and the other baselines. The left panel illustrates the natural displacement profile of the non-uniform Euler beam under given loads $q(x)$, flexural rigidity $EI(x)$, and boundary conditions. The right panel depicts the optimized position on the beam where the displacement is minimized, highlighting the location where the structural response is at its lowest.

Chapter 6

Conclusion

In this thesis, we have introduced three methods that apply DNNs as an alternative to GPs for the surrogate models in BO. We also extended the use of DNN to different settings of BO. These methods improved the scalability drawback of BO with GP while ensuring the optimization convergence. Further, DNNs-based Black-box Optimization shows the potential application to high-dimensional structural data. We summarize our contributions below and outline some potential future work in this area.

6.1 Contributions

To address the computational costs and scalability limitations (emphasizing the structural inputs....) caused by the cubic complexity of kernel inversion in GPs used for BO, this thesis focused on developing methods based on DNNs for both standard and extended BO problems.

The key idea was first applied within the standard BO setting. In **Chapter** 3, a novel black-box optimization algorithm was introduced, where the unknown function was modeled using an over-parameterized DNN. NTK theory was employed to control network uncertainty, along with the TS technique for selecting the next point for evaluation. Theoretical analysis of the algorithm's regret bound was demonstrated to show convergence and improved sample efficiency over existing methods. Additionally, the Neural-BO algorithm was shown to be scalable, with computational costs growing linearly with the number of data points, making it suitable for optimizing complex structured data like images and text. Experimental results on synthetic benchmarks and real-world optimization tasks confirmed that the algorithm outperformed current state-of-the-art methods in BO.

In **Chapter** 4, the DNN-based approach was extended to address BO with unknown, expensive constraints. Neural-CBO was introduced, where both the objective function and constraints were modeled by DNNs. EI was used as the acquisition function for the objective, while the feasible region was defined using Lower Confidence Bound (LCB) conditions. Theoretical analysis indicated that cumulative regret and constraint violations had upper bounds comparable to those of GP-based methods, under a more relaxed condition on network width. Experimental verification, conducted on synthetic and real-world tasks, demonstrated that Neural-CBO performed competitively with recent state-of-the-art approaches.

In **Chapter** 5, a novel problem setting for BO was introduced, integrating physical prior knowledge about the underlying black-box function through PDEs, using PINNs. The PINN-BO algorithm enhanced sample efficiency by efficiently handling a broad class of PDEs, thereby promoting the use of physical knowledge in BO. It was ultimately demonstrated that this approach significantly improved practical

performance when compared to existing methods across various synthetic and real-world benchmark functions.

6.2 Future Directions

In this thesis, the convergence of DNN-based BO approaches is demonstrated in terms of the maximum information gain, denoted as γ_T, with respect to the NTK. As established by Kassraie and Krause (2022), γ_T is bounded by $\mathcal{O}(T^{1-d^{-1}})$, where T represents the number of observations and d the input dimensionality. In **Chapter 3**, this result was utilized to prove that the Neural-BO algorithm achieves a sub-linear regret bound, which is crucial for ensuring efficient optimization. This result is based on the assumption that the observation noise is independent of the observed values. However, in more complex scenarios where the noise depends on prior observations, deriving a sub-linear regret bound remains an open problem, suggesting an interesting avenue for future research, either by improving the upper bound of γ_T or by designing a more refined algorithm to ensure sub-linear convergence.

A related and equally promising direction for future work is the improvement of the theoretical analysis of PINN-BO, as discussed in **Chapter 5**. The current regret bound for this algorithm includes the interaction information value, which is influenced by the PDE observations, making the regret bound dependent on the specific PDE. While it has been shown that PINN-BO is capable of handling a broad class of PDE constraints, establishing a regret bound for each specific PDE class, such as linear PDEs, remains a significant challenge. In Section 5.3, a regret bound is provided for a simple class of PDEs, yet extending this analysis to more complex classes could significantly improve the understanding of how physical information, expressed through PDEs, positively influences the optimization process. By analyzing the regret bound for each PDE class, the integration of physical knowledge into Black-box Optimization can be deepened, thereby enhancing the efficiency and practicality of algorithms like PINN-BO. This approach would not only strengthen the theoretical foundations of PINN-BO but also broaden its application to real-world optimization problems governed by diverse physical laws.

Appendix A

Supplementary Material of Chapter 3

A.1 Proof of Theoretical Analysis in Chapter 3

In this part, we provide the proof for Theorem 3.3.5. The main lemmas are from Lemma A.1.2 to Lemma A.1.10. Some main lemmas require auxiliary lemmas to complete their proofs. These auxiliary lemmas are provided instantly after the main lemmas and later proved in Section A.2.

To begin, we consider the following condition of the neural network width m

Condition A.1.1. *The network width m satisfies*

$$m \geq C \max\left\{\sqrt{\lambda} L^{-3/2}[\log(TL^2\alpha)]^{3/2}, T^6 L^6 \log(TL/\alpha) \max\{\lambda_0^{-4}, 1\}]\right\}$$
$$m[\log m]^{-3} \geq CTL^{12}\lambda^{-1} + CT^7\lambda^{-8}L^{18}(\lambda + LT)^6 + CL^{21}T^7\lambda^{-7}(1+\sqrt{T/\lambda})^6,$$

where C is a positive absolute constant.

Under this flexible condition of the search space as mentioned in Section 3.3, we need some new results. Following Allen-Zhu, Li, and Song (2019), we first define

$$\mathbf{h}_{i,0} = \mathbf{x}, \mathbf{h}_{i,l} = \phi(\mathbf{W}_l \mathbf{h}_{i,l-1}), l \in [L]$$

as the output of the l-th hidden layer. With this definition, we provide a norm bound of $\mathbf{h}_{i,l-1}$ as follows.

Lemma A.1.2 (Lemma 7.1, (Allen-Zhu, Li, and Song, 2019)). *If $\epsilon \in [0,1]$, with probability at least $1 - \mathcal{O}(nL)e^{-\Omega(m\epsilon^2/L)}$, we have*

$$\forall i \in [T], l \in [L], \|\mathbf{h}_{i,l-1}\| \in [ae^{-\epsilon}, be^{\epsilon}]$$

The following lemma is the concentration property of sampling value $\widetilde{f}_t(\mathbf{x})$ from estimated mean value $h(\mathbf{x}; \boldsymbol{\theta}_{t-1})$.

Lemma A.1.3. *For any $t \in [T]$, and any finite subset $\mathcal{D}_t \subset \mathcal{D}$, pick $c_t = \sqrt{4\log t + 2\log|\mathcal{D}_t|}$. Then we have:*

$$|\widetilde{f}_t(\mathbf{x}) - h(\mathbf{x}; \boldsymbol{\theta}_{t-1})| \leq c_t \nu_t \sigma_t(\mathbf{x}), \forall \mathbf{x} \in \mathcal{D}_t,$$

holds with probability $\geq 1 - t^{-2}$.

To prove Lemma A.1.3, we need a concentration bound on Gaussian distributions (Hoffman, Shahriari, and Freitas, 2013) as follows:

Lemma A.1.3.1 ((Hoffman, Shahriari, and Freitas, 2013)). Consider a normally distributed random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ and $\beta \geq 0$. The probability that X is within a radius of $\beta\sigma$ from its mean can then be written as:

$$\mathbb{P}(|X - \mu| \leq \beta\sigma\ ) \geq 1 - \exp(\beta^2/2)$$

Proof of Lemma A.1.3. Because the sampled value $\widetilde{f}_t(\mathbf{x})$ is sampled from

$$\mathcal{N}(h(\mathbf{x}; \boldsymbol{\theta}_{t-1}), \nu_t^2 \sigma_t^2(\mathbf{x})),$$

applying the concentration property in Lemma A.1.3.1, we have:

$$\mathbb{P}(|\widetilde{f}_t(\mathbf{x}) - h(\mathbf{x}; \boldsymbol{\theta}_{t-1})| \leq c_t \nu_t \sigma_t(\mathbf{x}) | \mathcal{F}_{t-1}) \geq 1 - \exp(-c_t^2/2)$$

Taking union bound over $\mathcal{D}_t$, we have for any t:

$$\mathbb{P}(|\widetilde{f}_t(\mathbf{x}) - h(\mathbf{x}; \boldsymbol{\theta}_{t-1})| \leq c_t \nu_t \sigma_t(\mathbf{x}) | \mathcal{F}_{t-1}) \geq 1 - |\mathcal{D}_t| \exp(-c_t^2/2)$$

Picking $c_t = \sqrt{4 \log t + 2 \log |\mathcal{D}_t|}$, we get the bound:

$$\mathbb{P}(|\widetilde{f}_t(\mathbf{x}) - h(\mathbf{x}; \boldsymbol{\theta}_{t-1})| \leq c_t \nu_t \sigma_t(\mathbf{x}) | \mathcal{F}_{t-1}) \geq 1 - \frac{1}{t^2}$$

□

By combining Lemma A.1.2 with techniques used in Lemma 4.1 of Cao and Gu (2019), Lemma B.3 of Cao and Gu (2019) and Theorem 5 of Allen-Zhu, Li, and Song (2019), we achieve a concentration property of estimated value $h(\mathbf{x}; \boldsymbol{\theta}_{t-1})$ from its true value $f(\mathbf{x})$ as follows:

Lemma A.1.4. *Suppose the width of the neural network m satisfies Condition A.1.1. Given any $\alpha \in (0, 1)$ and set $\eta = C(m\lambda + mLT)^{-1}$. Then for any $t \in [T]$, we have*

$$|h(\mathbf{x}; \boldsymbol{\theta}_{t-1}) - f(\mathbf{x})| \leq \nu_t \sigma_t(\mathbf{x}) + \epsilon(m), \forall \mathbf{x} \in \mathcal{D}_t,$$

holds with probability $\geq 1 - \alpha/2$ and

$$\begin{aligned}\epsilon(m) =& \frac{b}{a} C_{\epsilon,1} m^{-1/6} \lambda^{-2/3} L^3 \sqrt{\log m} + \frac{b}{a} C_{\epsilon,2} (1 - \eta m \lambda)^J \sqrt{TL/\lambda} \\ &+ \left(\frac{b}{a}\right)^3 C_{\epsilon,3} m^{-1/6} \sqrt{\log m} L^4 T^{5/3} \lambda^{-5/3} (1 + \sqrt{T/\lambda}),\end{aligned}$$

where $\{C_{\epsilon,i}\}_{i=1}^3$ are positive constants and a, b is lower and upper norm bound of input $\mathbf{x}$ as assumed in Section 3.1.

First, we define some necessary notations about linear and kernelized models.

Definition A.1.5. Let us define terms for the convenience as follows:

$$\begin{aligned}\mathbf{G}_t &= (\mathbf{g}(\mathbf{x}_1; \boldsymbol{\theta}_0), \cdots, \mathbf{g}(\mathbf{x}_t; \boldsymbol{\theta}_0)) \\ \mathbf{f}_t &= (f(\mathbf{x}_1), \cdots, f(\mathbf{x}_t))^\top \\ \mathbf{y}_t &= (y_1, \cdots, y_t)^\top \\ \boldsymbol{\epsilon}_t &= (f(\mathbf{x}_1) - y_1, \cdots, f(\mathbf{x}_t) - y_t)^\top,\end{aligned}$$

where $\boldsymbol{\epsilon}_t$ is the reward noise at time t. We recall that the definition of $\boldsymbol{\epsilon}_t$ is simply from our setting in Section 3.1. It can be verified that $\mathbf{U}_t = \lambda\mathbf{I} + \mathbf{G}_t\mathbf{G}_t^\top/m$. We also define $\mathbf{K}_t = \lambda\mathbf{I} + \mathbf{G}_t^\top\mathbf{G}_t/m$. We next reuse a lemma from Zhang et al. (2021) to bound the difference between the outputs of the neural network and the linearized model.

Lemma A.1.5.1. Suppose the network width m satisfies Condition A.1.1. Then, set $\eta = C_1(m\lambda + mLT)^{-1}$, with probability at least $1-\alpha$ over the random initialization of $\boldsymbol{\theta}_0$, we have $\forall \mathbf{x} \in \mathcal{D}, 0 < a \leq \|\mathbf{x}\|_2 \leq b$

$$\begin{aligned}
&\left|h(\mathbf{x};\boldsymbol{\theta}_{t-1}) - \langle \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0); \mathbf{G}_{t-1}^\top \mathbf{U}_{t-1}^{-1}\mathbf{r}_{t-1}/m\rangle\right| \\
\leq &\frac{b}{a}C_{\epsilon,1}t^{2/3}m^{-1/6}\lambda^{-2/3}L^3\sqrt{\log m} + \frac{b}{a}C_{\epsilon,2}(1-\eta m\lambda)^J\sqrt{tL/\lambda} \\
&+ \left(\frac{b}{a}\right)^3 C_{\epsilon,3}m^{-1/6}\sqrt{\log m}L^4t^{5/3}\lambda^{-5/3}\left(1+\sqrt{t/\lambda}\right),
\end{aligned}$$

where $\{C_{\epsilon,i}\}_{i=1}^3$ are positive constants. We provide the proof for this lemma in A.2.1.

Lemma A.1.5.2. Let $\alpha \in (0,1)$. Recall that matrix $\mathbf{U}_{t-1}$ is defined in Algorithm 1, $\mathbf{G}_{t-1}$ is defined in Definition A.1.5 and $\bullet_{t-1}$ is i.i.d sub-Gaussian observation noises up to optimization iteration $t-1$, with parameter R. Then with probability $1-\alpha$, we have

$$\left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\boldsymbol{\epsilon}_{t-1}/m\right| \leq \sigma_t(\mathbf{x})\frac{R}{\sqrt{\lambda}}\sqrt{2\log\left(\frac{1}{\alpha}\right)}$$

The proof for Lemma A.1.5.2 is given in A.2.2. Then, the following lemma provides the upper bound for approximating the NTK with the empirical gram matrix of the neural network at initialization by the maximum information gain associated with the NTK.

Lemma A.1.5.3. Let $\alpha \in (0,1)$. If the network width m satisfies $m \geq CT^6L^6\log(TL/\alpha)$, then with probability at least $1-\alpha$, with the points chosen from Algorithm 1, the following holds for every $t \in [T]$:

$$\log\det\left(\mathbf{I} + \lambda^{-1}\mathbf{K}_t\right) \leq 2\gamma_t + 1,$$

where γ_t is maximum information gain associated with the kernel k_{NTK}. We provided the proof of Lemma A.1.5.3 in A.2.3.

Lemma A.1.5.4 (Lemma D.2, Kassraie and Krause, 2022). Let $\alpha \in (0,1)$. Under Assumption 3.3.2 , if the network width m satisfies $m \geq CT^6L^6\log(TL/\alpha)$ and f be a member of $\mathcal{H}_{k_{\text{NTK}}}$ with bounded RKHS norm $\|f\|_{k_{\text{NTK}}} \leq B$, then with probability at least $1-\alpha$, there exists $\mathbf{w} \in \mathbb{R}^p$ such that

$$f(\mathbf{x}) = \langle \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0), \mathbf{w}\rangle, \|\mathbf{w}\|_2 \leq \sqrt{\frac{2}{m}}B$$

We remark that in our proofs, we assume $k_{\text{NTK}}(\mathbf{x},\mathbf{x}) \leq 1$ for simplicity. Now we are going on to prove Lemma A.1.4

Proof of Lemma A.1.4. First of all, since m satisfies Condition A.1.1, then with the choice of η, the condition required in Lemma A.1.5.1 - A.1.5.3 are satisfied. Thus, taking a union bound, we have with probability at least $1-3\alpha$, that the bounds provided by these lemmas hold. As we assume that f is in RKHS $\mathcal{H}_{k_{\text{NTK}}}$ with

NTK kernel, and $\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)/\sqrt{m}$ can be considered as finite approximation of $\varphi(\cdot)$, the feature map of the NTK from $\mathbb{R}^d \to \mathcal{H}_{k_{\text{NTK}}}$. From Lemma A.1.5.4 , there exists $\mathbf{w} \in \mathbb{R}^p$ such that $f(\mathbf{x}) = \langle \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0), \mathbf{w}\rangle = \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{w}$. Then for any $t \in [T]$, we will first provide the difference between the target function and the linear function $\langle \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0); \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{r}_{t-1}/m\rangle$ as:

$$
\begin{aligned}
&\left|f(\mathbf{x}) - \langle \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0); \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{r}_{t-1}/m\rangle\right| \\
&= \left|f(\mathbf{x}) - \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{r}_{t-1}/m\right| \\
&\leq \left|f(\mathbf{x}) - \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{f}_{t-1}/m\right| + \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\boldsymbol{\epsilon}_{t-1}/m\right| \\
&= \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{w} - \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{G}_{t-1}^\top \mathbf{w}/m\rangle\right| + \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\boldsymbol{\epsilon}_{t-1}/m\right| \\
&= \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \left(\mathbf{I} - \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{G}_{t-1}^\top/m\right)\mathbf{w}\right| + \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\boldsymbol{\epsilon}_{t-1}/m\right| \\
&= \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \left(\mathbf{I} - \mathbf{U}_{t-1}^{-1}\left(\mathbf{U}_{t-1} - \lambda\mathbf{I}\right)\right)\mathbf{w}\right| + \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\boldsymbol{\epsilon}_{t-1}/m\right| \\
&= \left|\lambda\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{w}\right| + \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\boldsymbol{\epsilon}_{t-1}/m\right| \\
&\leq \|\mathbf{w}\|_{k_{\text{NTK}}}\left\|\lambda\mathbf{U}_{t-1}^{-1}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)\right\|_{k_{\text{NTK}}} + \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\boldsymbol{\epsilon}_{t-1}/m\right| \\
&\leq \|\mathbf{w}\|_{k_{\text{NTK}}}\sqrt{\lambda\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)} + \left|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\boldsymbol{\epsilon}_{t-1}/m\right| \\
&\leq \sqrt{2}B\sigma_t(\mathbf{x}) + \sigma_t(\mathbf{x})\frac{R}{\sqrt{\lambda}}\sqrt{2\log\left(\frac{1}{\alpha}\right)}
\end{aligned}
\tag{A.1}
$$

where the first inequality uses triangle inequality and the fact that $\mathbf{r}_{t-1} = \mathbf{f}_{t-1} + \boldsymbol{\epsilon}_{t-1}$. The second inequality is from the reproducing property of function relying on RKHS, and the fourth equality is from the verification noted in Definition A.1.5. The last inequality directly uses the results from Lemma A.1.5.2 and Lemma A.1.5.4. We have a more compact form of the inequality A.1 as:

$$
|f(\mathbf{x}) - \langle \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0); \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{r}_{t-1}/m\rangle| \leq \nu_t\sigma_t(\mathbf{x}),
$$

where we set $\nu_t = \sqrt{2}B + \frac{R}{\sqrt{\lambda}}\sqrt{2\log(1/\alpha)}$. Then, by combining this bound with Lemma A.1.5.1, we have

$$
\begin{aligned}
|h(\mathbf{x};\boldsymbol{\theta}_{t-1}) - f(\mathbf{x})| &\leq \nu_t\sigma_t(\mathbf{x}) + \frac{b}{a}C_{\epsilon,1}t^{2/3}m^{-1/6}\lambda^{-2/3}L^3 \\
&\quad + \frac{b}{a}C_{\epsilon,2}(1-\eta m\lambda)^J\sqrt{tL/\lambda} \\
&\quad + \left(\frac{b}{a}\right)^3 C_{\epsilon,3}m^{-1/6}\sqrt{\log m}L^4t^{5/3}\lambda^{-5/3}\left(1+\sqrt{t/\lambda}\right) \\
&\leq \nu_t\sigma_t(\mathbf{x}) + \epsilon(m)
\end{aligned}
$$

By setting α to $\alpha/3$ (required by the union bound discussed at the beginning of the proof) and taking $t = T$, we get the result presented in Lemma A.1.4. □

The next lemma gives a lower bound of the probability that the sampled value $\widetilde{f}_t(\mathbf{x})$ is larger than the true function value up to the approximation error $\epsilon(m)$.

Lemma A.1.6. *For any $t \in [T], \mathbf{x} \in D$, we have $\mathbb{P}(\widetilde{f}_t(\mathbf{x}) + \epsilon(m) > f(\mathbf{x})) \geq (4e\pi)^{-1}$.*

Proof of Lemma A.1.6. Following proof style of Lemma 8 in Zhou, Li, and Gu (2020), using Lemma A.1.4 and Gaussian anti-concentration property, we have

$$\begin{aligned}&\mathbb{P}(\widetilde{f}_t(\mathbf{x}) + \epsilon(m) > f(\mathbf{x})|\mathcal{F}_{t-1})\\ =&\mathbb{P}\left(\frac{\widetilde{f}_t(\mathbf{x}) - h(\mathbf{x};\boldsymbol{\theta}_{t-1})}{\nu_t\sigma_t(\mathbf{x})} > \frac{|f(\mathbf{x}) - h(\mathbf{x};\boldsymbol{\theta}_{t-1})| - \epsilon(m)}{\nu_t\sigma_t(\mathbf{x})}\middle|\mathcal{F}_{t-1}\right) \geq \frac{1}{4e\pi}\end{aligned}$$

□

For any step t, we consider how the standard deviation of the estimates for each point is, in comparison with the standard deviation for the optimal value. Following Zhang et al., 2021, we divide the discretization $\mathcal{D}_t$ into two sets: saturated and unsaturated sets. For more details, we define the set of saturated points as:

$$S_t = \{\forall \mathbf{x} \in \mathcal{D}_t, f([\mathbf{x}^*]_t) - f(\mathbf{x}) \geq (1 + c_t)\nu_t\sigma_t(\mathbf{x}) + 2\epsilon(m)\} \tag{A.2}$$

The following lemma shows that, the algorithm can pick unsaturated points $\mathbf{x}_t$ with high probability.

Lemma A.1.7 (Lemma 4.5, Zhou, Li, and Gu, 2020). *Let $\mathbf{x}_t$ be the chosen point at step $t \in [T]$. Then, $\mathbb{P}(\mathbf{x}_t \notin S_t|\mathcal{F}_{t-1}) \geq \frac{1}{4e\sqrt{\pi}} - \frac{1}{t^2}$*

The next lemma bounds the expectation of instantaneous regret at each round conditioned on history $\mathcal{F}_{t-1}$.

Lemma A.1.8. *Suppose the width of the neural network m satisfies Condition A.1.1. Set $\eta = C_1(m\lambda + mLT)^{-1}$. Then with probability at least $1 - \alpha$, we have for all $t \in [T]$ that*

$$\mathbb{E}[f(\mathbf{x}^*) - f(\mathbf{x}_t)|\mathcal{F}_{t-1}] \leq C_2(1 + c_t)\nu_t\sqrt{L}\mathbb{E}[\min(\sigma_t(\mathbf{x}_t), B)|\mathcal{F}_{t-1}] + 4\epsilon(m) + \frac{2B+1}{t^2}$$

where C_1, C_2 are absolute constants.

Proof of Lemma A.1.8. This proof inherits the proof of Lemma 4.6 Zhang et al., 2021, and by using the result of unsaturated point $\mathbf{x}_t$ in Lemma A.1.7, along with $|f(x)| = |\langle f, k(x,\cdot)\rangle| \leq B$ instead of $|f(x)| \leq 1$ as in Zhang et al., 2021, we have the following result:

$$\begin{aligned}&\mathbb{E}[f([\mathbf{x}^*]_t) - f(\mathbf{x}_t) \mid \mathcal{F}_{t-1}]\\ \leq&44e\sqrt{\pi}(1 + c_t)\nu_t C_1\sqrt{L}\mathbb{E}[\min\{\sigma_t(\mathbf{x}_t), B\} \mid \mathcal{F}_{t-1}] + 4\epsilon(m) + \frac{2B}{t^2}\end{aligned}$$

Now using Eqn 3.1, we have the instantaneous regret at round t

$$r_t = f(\mathbf{x}^*) - f([\mathbf{x}^*]_t) + f([\mathbf{x}^*]_t) - f(\mathbf{x}_t) \leq \frac{1}{t^2} + f([\mathbf{x}^*]_t) - f(\mathbf{x}_t)$$

Taking conditional expectation, we have the result as stated in Lemma A.1.8.

□

The next lemma bounds the cumulative regret $\sum_{t=1}^{T} r_t$ after T iterations.

Lemma A.1.9. *Suppose the width of the neural network m satisfies Condition A.1.1. Then set $\eta = C_1(m\lambda + mLT))^{-1}$, we have, with probability at least $1-\alpha$, that*

$$\sum_{t=1}^{T} f(\mathbf{x}^*) - f(\mathbf{x}_t) \leq 4T\epsilon(m) + \frac{(2B+1)\pi^2}{6} + C_2(1+c_T)\nu_t \sum_{t=1}^{T} \min(\sigma_t(\mathbf{x}_t), B) + (4B + C_3(1+c_T)\nu_t L + 4\epsilon(m))\sqrt{2\log(1/\alpha)T}$$

where C_1, C_2, C_3 are absolute constants.

Proof of Lemma A.1.9. Similar to Lemma A.1.8, we utilize the proof of Lemma 4.7 in Zhang et al. (2021) or equivalent proof of Lemma 13 in Chowdhury and Gopalan (2017). Then with $f(\mathbf{x}) \leq B$, we have

$$\sum_{i=1}^{T} r_t \leq 4T\epsilon(m) + (2B+1)\sum_{i=1}^{T} t^{-2} + C_1(1+c_T)\nu_t \sum_{i=1}^{T} \min(\sigma_t(\mathbf{x}_t), B) + (4B + C_1C_2(1+c_T)\nu_t L + 4\epsilon(m))\sqrt{2\log(1/\alpha)T}$$

□

The next lemma gives a bound on the sum of variance $\sum_{i=1}^{T} \min(\sigma_t(\mathbf{x}_t), B)$ which appears in Lemma A.1.9.

Lemma A.1.10. *Suppose the width of the neural network m satisfies Condition A.1.1. Then set $\eta = C_1(m\lambda + mLT))^{-1}$, we have, with probability at least $1-\alpha$, that*

$$\sum_{i=1}^{T} \min(\sigma_t(\mathbf{x}_t), B) \leq \sqrt{\frac{\lambda BT}{\log(B+1)}(2\gamma_T + 1)}$$

To prove lemma A.1.10, we first need to utilize a technical lemma:

Lemma A.1.10.1. Let $\{\mathbf{v}_t\}_{t=1}^{\infty}$ be a sequence in $\mathbb{R}^p$, and define $\mathbf{V}_t = \lambda\mathbf{I} + \sum_{i=1}^{t} \mathbf{v}_i\mathbf{v}_i^\top$ and B is a positive constant. If $\lambda \geq 1$, then

$$\sum_{i=1}^{T} \min\{\mathbf{v}_t^\top \mathbf{V}_{t-1}^{-1}\mathbf{v}_{t-1}, B\} \leq \frac{B}{\log(B+1)} \log\det\left(\mathbf{I} + \lambda^{-1}\sum_{i=1}^{T} \mathbf{v}_i\mathbf{v}_i^\top\right)$$

A.2.4 provided the proof for Lemma A.1.10.1. Now, we start to prove Lemma A.1.10

Proof of Lemma A.1.10. From Cauchy-Schwartz inequality, we have

$$\sum_{i=1}^{T} \min\{\sigma_t(\mathbf{x}_t), B\} \leq \sqrt{T\sum_{i=1}^{T} \min\{\sigma_t^2(\mathbf{x}_t), B\}}$$

We also have,

$$\begin{aligned}
\sum_{i=1}^{T} \min\{\sigma_t^2(\mathbf{x}_t), B\} &\leq \lambda \sum_{i=1}^{T} \min\{\mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1} \mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0)/m, B\} \\
&\leq \frac{\lambda B}{\log(B+1)} \log\det\left(\mathbf{I} + \lambda^{-1} \sum_{i=1}^{T} \mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0)\mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0)^\top/m\right) \\
&= \frac{\lambda B}{\log(B+1)} \log\det\left(\mathbf{I} + \lambda^{-1}\mathbf{G}_T\mathbf{G}_T^\top/m\right) \\
&= \frac{\lambda B}{\log(B+1)} \log\det\left(\mathbf{I} + \lambda^{-1}\mathbf{G}_T^\top\mathbf{G}_T/m\right) \\
&= \frac{\lambda B}{\log(B+1)} \log\det\left(\mathbf{I} + \lambda^{-1}\mathbf{K}_T\right) \\
&\leq \frac{\lambda B}{\log(B+1)} (2\gamma_T + 1)
\end{aligned}$$

where the first inequality moves the positive parameter λ outside the min operator. Then the second inequality utilizes Lemma A.1.10.1, the first equality use the expression of $\mathbf{G}_t$ in 1, the second equality is from the fact that $\det(\mathbf{I} + \mathbf{A}\mathbf{A}^\top) = \det(\mathbf{I} + \mathbf{A}^\top\mathbf{A})$, and the last equality uses the definition of $\mathbf{K}_T$ in Definition A.1.5 and the last inequality uses Lemma A.1.5.3. Finally, we have,

$$\sum_{i=1}^{T} \min\{\sigma_t(\mathbf{x}_t), B\} \leq \sqrt{\frac{\lambda BT}{\log(B+1)}(2\gamma_T + 1)}.$$

□

Finally, we repeat to emphasize the proof of Theorem 4.2.2 given in Section 3.3.

Proof of Theorem 4.2.2. With probability at least $1-\alpha$, we have

$$\begin{aligned}
R_T &= \sum_{t=1}^{T} f(\mathbf{x}^*) - f(\mathbf{x}_t) \\
&= \sum_{t=1}^{T} [f(\mathbf{x}^*) - f([\mathbf{x}^*]_t)] + [f([\mathbf{x}^*]_t) - f(\mathbf{x}_t)] \\
&\leq 4T\epsilon(m) + \frac{(2B+1)\pi^2}{6} + \bar{C}_1(1+c_T)\nu_T\sqrt{L}\sum_{i=1}^{T}\min(\sigma_t(\mathbf{x}_t), B) \\
&\quad + (4 + \bar{C}_2(1+c_T)\nu_T L + 4\epsilon(m))\sqrt{2\log(1/\alpha)T} \\
&\leq \bar{C}_1(1+c_T)\nu_T\sqrt{L}\sqrt{\frac{\lambda BT}{\log(B+1)}(2\gamma_T+1)} + \frac{(2B+1)\pi^2}{6} + 4T\epsilon(m) \\
&\quad + 4\epsilon(m)\sqrt{2\log(1/\alpha)T} + (4 + \bar{C}_2(1+c_T)\nu_T L)\sqrt{2\log(1/\alpha)T} \\
&= \bar{C}_1(1+c_T)\nu_t\sqrt{L}\sqrt{\frac{\lambda BT}{\log(B+1)}(2\gamma_T+1)} + \frac{(2B+1)\pi^2}{6} \\
&\quad + \epsilon(m)(4T + \sqrt{2\log(1/\alpha)T}) + (4 + \bar{C}_2(1+c_T)\nu_t L)\sqrt{2\log(1/\alpha)T}
\end{aligned}$$

The first inequality is due to Lemma A.1.9, which provides the bound for cumulative regret R_T in terms of $\sum_{t=1}^{T}\min(\sigma_t(\mathbf{x}_t), B)$. The second inequality further provides the bound of term $\sum_{t=1}^{T}\min(\sigma_t(\mathbf{x}_t), B)$ due to Lemma A.1.10, while the last equality rearranges addition. Picking $\eta = (m\lambda + mLT)^{-1}$ and $J = (1 + LT/\lambda)\left(\log(C_{\epsilon,2}) + \log\left(T^3L\lambda^{-1}\log(1/\alpha)\right)\right)$, we have

$$\begin{aligned}
&\frac{b}{a}C_{\epsilon,2}(1-\eta m\lambda)^J\sqrt{TL/\lambda}\left(4T+\sqrt{2\log(1/\alpha)T}\right)\\
=&\frac{b}{a}C_{\epsilon,2}\left(1-\frac{1}{1+LT/\lambda}\right)^J\left(4T+\sqrt{2\log(1/\alpha)T}\right)\\
=&\frac{b}{a}C_{\epsilon,2}e^{-\left(\log(C_{\epsilon,2})+\log\left(T^3L\lambda^{-1}\log(1/\alpha)\right)\right)}\left(4T+\sqrt{2\log(1/\alpha)T}\right)\\
=&\frac{b}{a}\frac{1}{C_{\epsilon,2}}.T^{-3}L^{-1}\lambda\log^{-1}(1/\alpha)\left(4T+\sqrt{2\log(1/\alpha)T}\right)\leq\frac{b}{a}
\end{aligned}$$

Then choosing m that satisfies:

$$\begin{aligned}
\left(\frac{b}{a}C_{\epsilon,1}m^{-1/6}\lambda^{-2/3}L^3\sqrt{\log m}+\left(\frac{b}{a}\right)^3C_{\epsilon,3}m^{-1/6}\sqrt{\log m}L^4T^{5/3}\lambda^{-5/3}(1+\sqrt{T/\lambda})\right)\\
\left(4T+\sqrt{2\log(1/\alpha)T}\right)\leq\left(\frac{b}{a}\right)^3
\end{aligned}$$

We finally achieve the bound of R_T as:

$$\begin{aligned}
R_T\leq&\bar{C}_1(1+c_T)\nu_T\sqrt{L}\sqrt{\frac{\lambda BT}{\log(B+1)}(2\gamma_T+1)}\\
&+(4+\bar{C}_2(1+c_T)\nu_TL)\sqrt{2\log(1/\alpha)T}+\frac{(2B+1)\pi^2}{6}+\frac{b(a^2+b^2)}{a^3}
\end{aligned}$$

□

A.2 Proof of Auxiliary Lemmas

A.2.1 Proof of Lemma A.1.5.1

We strictly inherit the technique used in Lemma 4.1 Cao and Gu, 2019, Lemma B.3 Cao and Gu, 2019 and Theorem 5 Allen-Zhu, Li, and Song, 2019, combine with Lemma A.1.2 to achieve following results:

Lemma A.2.0.1. There exists positive constants $\{C_i\}_{i=1}^2$ and $\bar{C}_{\epsilon,1}$ such that for any $\alpha\in(0,1]$, if τ satisfies that:

$$C_1m^{-3/2}L^{-3/2}\left[\log\left(TL^2/\alpha\right)\right]^{3/2}\leq\tau\leq C_2L^{-6}[\log m]^{-3/2},$$

then with probability at least $1-\alpha$ over the randomness of $\boldsymbol{\theta}_0$ and $\mathbf{x}\in\{\mathbf{x}_1,\dots\mathbf{x}_T\}$ with $0<a\leq\|\mathbf{x}\|_2\leq b$:

1. For all $\hat{\boldsymbol{\theta}},\widetilde{\boldsymbol{\theta}}$ satisfying $\left\|\boldsymbol{\theta}_0-\hat{\boldsymbol{\theta}}\right\|_2\leq\tau$, $\left\|\boldsymbol{\theta}_0-\widetilde{\boldsymbol{\theta}}\right\|_2\leq\tau$,

$$\left|h(\mathbf{x};\hat{\boldsymbol{\theta}})-h(\mathbf{x};\widetilde{\boldsymbol{\theta}})-\langle\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0),\hat{\boldsymbol{\theta}}-\widetilde{\boldsymbol{\theta}}\rangle\right|\leq\frac{b}{a}\bar{C}_{\epsilon,1}\tau^{4/3}L^3\sqrt{m\log m}$$

2. For all $\boldsymbol{\theta}$ satisfying $\|\boldsymbol{\theta}_0 - \boldsymbol{\theta}\|_2 \leq \tau$,

$$\|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}) - \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)\|_2 \leq \frac{b}{a}\bar{C}_{\epsilon,1}\sqrt{\log m}\tau^{1/3}L^3\|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)\|_2$$

3. For all $\boldsymbol{\theta}$ satisfying $\|\boldsymbol{\theta}_0 - \boldsymbol{\theta}\|_2 \leq \tau$,

$$\|\mathbf{g}(\mathbf{x};\boldsymbol{\theta})\|_F \leq \frac{b}{a}\bar{C}_{\epsilon,1}\sqrt{mL}$$

The next lemma controls the difference between the parameter of neural network learned by gradient descent and the theoretical optimal solution of the linearized network.

Lemma A.2.0.2. There exists constants $\{C_i\}_{i=1}^4$ and $\bar{C}_{\epsilon,2}$ such that for any $\alpha \in (0,1]$, if η, m satisfy that for all $t \in [T]$

$$\begin{aligned}
&2\sqrt{t/\lambda} \geq C_1 m^{-1}L^{-3/2}\left[\log\left(TL^2/\alpha\right)\right]^{3/2},\\
&2\sqrt{t/\lambda} \leq C_2 \min\left\{m^{1/2}L^{-6}[\log m]^{-3/2}, m^{7/8}(\lambda^2\eta^2L^{-6}t^{-1}(\log m)^{-1}\right\},\\
&\eta \leq C_3(m\lambda + tmL)^{-1},\\
&m^{1/6} \geq C_4\sqrt{\log m}L^{7/2}t^{7/6}\lambda^{-7/6}(1 + \sqrt{t/\lambda}),
\end{aligned}$$

then with probability at least $1-\alpha$ over the randomness of $\boldsymbol{\theta}_0$, $\mathbf{x} \in \{\mathbf{x}_1, \ldots \mathbf{x}_T\}$ with $0 < a \leq \|\mathbf{x}\|_2 \leq b$, we have $\|\boldsymbol{\theta}_0 - \boldsymbol{\theta}\|_2 \leq 2\sqrt{t/(m\lambda)}$ and

$$\begin{aligned}
&\left\|\boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}_0 - \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{r}_{t-1}/m\right\|_2\\
&\leq (1-\eta m\lambda)^J\sqrt{t/(m\lambda)} + \left(\frac{b}{a}\right)^2\bar{C}_{\epsilon,2}m^{-2/3}\sqrt{\log m}L^{7/2}t^{5/3}\lambda^{-5/3}(1+\sqrt{t/\lambda})
\end{aligned}$$

Proof of Lemma A.1.5.1. Set $\tau = 2\sqrt{t/m\lambda}$, it can be verified that τ satisfies condition in A.2.0.1. Therefore, by inequality 1 in Lemma A.2.0.1, and with the initialization of the network $f(\mathbf{x};\boldsymbol{\theta}_0) = 0$, there exists a constant $C_{\epsilon,1}$ such that

$$|h(\mathbf{x};\boldsymbol{\theta}_{t-1}) - \langle\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0), \boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}_0\rangle| \leq \frac{b}{a}C_{\epsilon,1}t^{2/3}m^{-1/6}\lambda^{-2/3}L^3\sqrt{m\log m}$$

Then, we have the difference between the linearized model and the theoretical optimal solution of kernelized regression:

$$\begin{aligned}
&|\langle\mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0), \boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}_0\rangle - \langle\mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0), \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{r}_{t-1}/m\rangle|\\
&\leq \|\mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0)\|_2\left\|\boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}_0 - \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{r}_{t-1}/m\right\|_2\\
&\leq \frac{b}{a}\bar{C}_{\epsilon,1}\sqrt{mL}\Bigg((1-\eta m\lambda)^J\sqrt{t/(m\lambda)}\\
&\quad + \left(\frac{b}{a}\right)^2\bar{C}_{\epsilon,2}m^{-2/3}\sqrt{\log m}L^{7/2}t^{5/3}\lambda^{-5/3}(1+\sqrt{t/\lambda})\Bigg)\\
&\leq \frac{b}{a}C_{\epsilon,2}(1-\eta m\lambda)^J\sqrt{tL/(\lambda)} + \left(\frac{b}{a}\right)^3C_{\epsilon,3}m^{-1/6}\sqrt{\log m}L^4t^{5/3}\lambda^{-5/3}\left(1+\sqrt{t/\lambda}\right),
\end{aligned}$$

where the first inequality is from (3) and Lemma A.2.0.2, with $C_{\epsilon,2} = \bar{C}_{\epsilon,1}$ and $C_{\epsilon,3} = \bar{C}_{\epsilon,1}\bar{C}_{\epsilon,2}$. Finally, we have

$$\begin{aligned}
&\left|h(\mathbf{x};\boldsymbol{\theta}_{t-1}) - \langle \mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0), \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{r}_{t-1}/m\rangle\right| \\
&\leq |h(\mathbf{x};\boldsymbol{\theta}_{t-1}) - \langle \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0), \boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}_0\rangle| \\
&+ |\langle \mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0), \boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}_0\rangle - \langle \mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0), \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{r}_{t-1}/m\rangle| \\
&\leq \frac{b}{a}C_{\epsilon,1}t^{2/3}m^{-1/6}\lambda^{-2/3}L^3\sqrt{m\log m} + \frac{b}{a}C_{\epsilon,2}(1-\eta m\lambda)^J\sqrt{tL/(\lambda)} \\
&+ \left(\frac{b}{a}\right)^3 C_{\epsilon,3}m^{-1/6}\sqrt{\log m}L^4t^{5/3}\lambda^{-5/3}\left(1+\sqrt{t/\lambda}\right)
\end{aligned}$$

as stated in Lemma A.1.5.1. □

A.2.2 Proof of Lemma A.1.5.2

Proof of Lemma A.1.5.2. We bound the noise-affected term using the sub-Gaussianity assumption. Let $\mathbf{Z}_{t-1}^\top(\mathbf{x}) = \mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top\mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}/m$. We will show that $\mathbf{Z}_{t-1}^\top(\mathbf{x})\boldsymbol{\epsilon}_{t-1}$ is a sub-Gaussian random variable whose moment generating function is bounded by that of a Gaussian random variable with variance $\frac{R^2\sigma_i^2(\mathbf{x})}{\lambda}$.

Following the proof style in Vakili et al., 2021, we bound the noise-affected term $\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top\mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\boldsymbol{\epsilon}_{t-1}/m = \mathbf{Z}_{t-1}^\top(\mathbf{x})\boldsymbol{\epsilon}_{t-1}$. Let $\zeta_i(\mathbf{x}) = [\mathbf{Z}_{t-1}(\mathbf{x})]_i$, then we have:

$$\begin{aligned}
\mathbb{E}\left[\exp\left(\mathbf{Z}_{t-1}^\top(\mathbf{x})\boldsymbol{\epsilon}_{t-1}\right)\right] &= \mathbb{E}\left[\exp\sum_{i=1}^{t-1}\zeta_i(\mathbf{x})\boldsymbol{\epsilon}_i\right] \\
&= \prod_{i=1}^{t-1}\mathbb{E}[\exp\zeta_i(\mathbf{x})\boldsymbol{\epsilon}_i] \\
&\leq \prod_{i=1}^{t-1}\exp\left(\frac{R^2\zeta_i^2(\mathbf{x})}{2}\right) \\
&= \exp\left(\frac{R^2\sum_{i=1}^{t-1}\zeta_i^2(\mathbf{x})}{2}\right) \\
&= \exp\left(\frac{R^2\|\mathbf{Z}_{t-1}(\mathbf{x})\|_2^2}{2}\right),
\end{aligned} \tag{A.3}$$

where the second equation is the consequence of independence of $\zeta_i(\mathbf{x})\boldsymbol{\epsilon}_i$, which directly utilizes our i.i.d noise assumption which is mentioned in Assumption 3.3.4. The first inequality holds by the concentration property of the sub-Gaussian noise

random variable with parameter R. Then we bound:

$$\begin{aligned}
\|\mathbf{Z}_{t-1}(\mathbf{x})\|_2^2 &= \mathbf{Z}_{t-1}(\mathbf{x})\mathbf{Z}_{t-1}^\top(\mathbf{x}) \\
&= \frac{1}{m^2}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{G}_{t-1}\mathbf{G}_{t-1}^\top \mathbf{U}_{t-1}^{-1}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0) \\
&= \frac{1}{m}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}(\mathbf{U}_{t-1} - \lambda\mathbf{I})\mathbf{U}_{t-1}^{-1}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0) \\
&= \frac{1}{m}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top (\mathbf{I} - \lambda\mathbf{U}_{t-1}^{-1})\mathbf{U}_{t-1}^{-1}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0) \\
&= \frac{1}{m}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \left[\mathbf{U}_{t-1}^{-1}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0) - \lambda\mathbf{U}_{t-1}^{-2}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)\right] \\
&= \frac{1}{m}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-1}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0) - \frac{\lambda}{m}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-2}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0) \\
&= \frac{1}{\lambda}\sigma_t^2(\mathbf{x}) - \frac{\lambda}{m}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)^\top \mathbf{U}_{t-1}^{-2}\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0) \\
&= \frac{1}{\lambda}\sigma_t^2(\mathbf{x}) - \frac{\lambda}{m}\left\|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)\mathbf{U}_{t-1}^{-2}\right\|_2^2 \\
&\leq \frac{1}{\lambda}\sigma_t^2(\mathbf{x}),
\end{aligned} \tag{A.4}$$

where the second equality is from definition of $\mathbf{Z}_{t-1}$ and the third inequality is from the note of Definition A.1.5. The inequality is from the fact that $\frac{\lambda}{m}\left\|\mathbf{g}(\mathbf{x};\boldsymbol{\theta}_0)\mathbf{U}_{t-1}^{-2}\right\|^2 \geq 0, \forall \mathbf{x}, t$. Replace inequality A.4 to equation A.3 then we have

$$\mathbb{E}\left[\exp\left(\mathbf{Z}_{t-1}^\top(\mathbf{x})\boldsymbol{\epsilon}_{t-1}\right)\right] \leq \exp\left(\frac{R^2\sigma_t^2(\mathbf{x})}{2\lambda}\right)$$

Thus, using Chernoff-Hoeffding inequality Antonini, Kozachenko, and Volodin, 2008, we have with probabilty at least $1-\alpha$:

$$\left|\mathbf{Z}_{t-1}^\top(\mathbf{x})\boldsymbol{\epsilon}_{t-1}\right| \leq \frac{R\sigma_t(\mathbf{x})}{\sqrt{\lambda}}\sqrt{2\log\left(\frac{1}{\alpha}\right)}$$

□

A.2.3 Proof of Lemma A.1.5.3

Proof of Lemma A.1.5.3. From the definition of $\mathbf{K}_t$ and Lemma B.7 Zhou, Li, and Gu, 2020, we have that

$$\begin{aligned}
\log\det\Big(\mathbf{I}+\lambda^{-1}\mathbf{K}_t\Big) &= \log\det\left(\mathbf{I}+\sum_{i=1}^{t}\mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0)\mathbf{g}(\mathbf{x}_t;\boldsymbol{\theta}_0)^\top/(m\lambda)\right)\\
&= \log\det\Big(\mathbf{I}+\lambda^{-1}\mathbf{H}_t+\lambda^{-1}(\mathbf{K}_t-\mathbf{H}_t)\Big)\\
&\le \log\det\Big(\mathbf{I}+\lambda^{-1}\mathbf{H}_t\Big)+\langle(\mathbf{I}+\lambda^{-1}\mathbf{H}_t)^{-1},\lambda^{-1}(\mathbf{K}_t-\mathbf{H}_t)\rangle\\
&\le \log\det\Big(\mathbf{I}+\lambda^{-1}\mathbf{H}_t\Big)+\Big\|(\mathbf{I}+\lambda^{-1}\mathbf{H}_t)^{-1}\Big\|_F\Big\|\lambda^{-1}(\mathbf{K}_t-\mathbf{H}_t)\Big\|_F\\
&\le \log\det\Big(\mathbf{I}+\lambda^{-1}\mathbf{H}_t\Big)+t\sqrt{t}\epsilon\\
&\le \log\det\Big(\mathbf{I}+\lambda^{-1}\mathbf{H}_t\Big)+1\\
&\le 2\gamma_t+1,
\end{aligned}$$

where the first equality is from the definition of $\mathbf{K}_t$ in Definition A.1.5, the first inequality is from the convexity of $\log\det(\cdot)$ function, and the second inequality is from the fact that $\langle\mathbf{A},\mathbf{B}\rangle \le \|\mathbf{A}\|_F\|\mathbf{B}\|_F$. The third inequality is from the fact that $\|\mathbf{A}\|_F \le \sqrt{t}\|\mathbf{A}\|_2$ if $\mathbf{A}\in\mathbb{R}^{t\times t}$. The fourth inequality utilizes the choice of $\epsilon = T^{-3/2}$ and the last inequality inherits Lemma 3 in Chowdhury and Gopalan, 2017. □

A.2.4 Proof of Lemma A.1.10.1

Proof of Lemma A.1.10.1. By basic linear algebra, we have

$$\begin{aligned}
\det(\mathbf{V}_t) &= \det\Big(\mathbf{V}_{t-1}+\mathbf{v}_n\mathbf{v}_n^\top\Big) = \det(\mathbf{V}_t)\det\Big(\mathbf{I}+\mathbf{V}_t^{-1/2}\mathbf{v}_n(\mathbf{V}_t^{-1/2}\mathbf{v}_n)^\top\Big)\\
&= \det(\mathbf{V}_{t-1})(1+\|\mathbf{v}_{t-1}\|^2_{\mathbf{V}_{t-1}^{-1}})\\
&= \det(\lambda\mathbf{I})\prod_{t=1}^{T}(1+\|\mathbf{v}_{t-1}\|^2_{\mathbf{V}_{t-1}^{-1}})
\end{aligned}$$

where we used that all the eigenvalues of a matrix of the form $\mathbf{I}+\mathbf{x}\mathbf{x}^\top$ are 1, except one eigenvalue, which is $1+\|\mathbf{x}\|^2$ and which corresponds to the eigenvector $\mathbf{x}$. Using $\min\{u,B\} \le \frac{B}{\log(B+1)}\log(1+u), \forall u\in[0,B]$, we get

$$\begin{aligned}
\sum_{t=1}^{T}\min\{B,\|\mathbf{v}_{t-1}\|^2_{\mathbf{V}_{t-1}^{-1}}\} &\le \frac{B}{\log(B+1)}\sum_{t=1}^{T}\log\Big(1+\|\mathbf{v}_{t-1}\|^2_{\mathbf{V}_{t-1}^{-1}}\Big)\\
&\le \frac{B}{\log(B+1)}\log\det\left(\mathbf{I}+\lambda^{-1}\sum_{i=1}^{T}\mathbf{v}_i\mathbf{v}_i^\top\right)
\end{aligned}$$

□

Appendix B

Supplementary Material of Chapter 4

B.1 Proof of Theoretical Analysis in Chapter 3

In this section, we will provide the proof of Lemma 4.1.5 and Theorem 4.2.2. Before going to the proof, we briefly remind existing terms and introduce new notations for convenience. Remind that $\mathbf{g}_a(\mathbf{x};\mathbf{W}) = \nabla_{\mathbf{W}} a(\mathbf{x};\mathbf{W})$. Therefore, $\mathbf{g}_a(\mathbf{x};\mathbf{W}_0)$ and $\mathbf{g}_a(\mathbf{x};\mathbf{W}_t)$ will be the gradients of the neural network $a(\mathbf{x};\mathbf{W})$ (using to model a *arbitrary* function f_a, defined in Eqn. 4.1) at initialization and at iteration t, respectively. Further, let us define terms as follows:

$$\begin{aligned}
\mathbf{G}_{a,t-1} &= [\mathbf{g}_a(\mathbf{x}_1;\mathbf{W}_0),\ldots,\mathbf{g}_a(\mathbf{x}_{t-1};\mathbf{W}_0)] \\
\bar{\mathbf{G}}_{a,t-1} &= [\mathbf{g}_a(\mathbf{x}_1;\mathbf{W}_{t-1}),\ldots,\mathbf{g}_a(\mathbf{x}_{t-1};\mathbf{W}_{t-1})] \\
\mathbf{U}_{a,t-1} &= \mathbf{I} + \mathbf{G}_{a,t-1}\mathbf{G}_{a,t-1}^\top \\
\mathbf{F}_{a,t-1} &= [f_a(\mathbf{x}_1),\ldots,f_a(\mathbf{x}_{t-1})]
\end{aligned} \tag{B.1}$$

Further, it can be verify that $\mathbf{H}_0 = \mathbf{G}_{a,t-1}^\top \mathbf{G}_{a,t-1}$, where $\mathbf{H}_0$ is the NTK matrix at initialization defined in Definition 4.1.1. Now we are ready to bound Lemma 4.1.5.

B.1.1 Proof of Lemma 4.1.5

Lemma 4.1.5. *Let Assumptions 4.1.2 and 4.1.3 hold. Using neural network $a(\mathbf{x};\mathbf{W})$ satisfied Condition 4.1.4 to model an arbitrary function f_a. Setting the step size at training step t as $\alpha_t \leq \frac{\nu}{(T+1)^2}$, then for any $\delta \in (0,1)$, with probability at least $1-\delta\exp\big(\Omega(C^{-L}m^{1/36})\big)$, the following holds for all $\mathbf{x} \in \mathcal{D}$ and $1 \leq t \leq T$:*

$$\begin{aligned}
&|f_a(\mathbf{x}) - a(\mathbf{x};\mathbf{W}_{t-1})| \leq \beta_{a,t}\sigma_{a,t-1}(\mathbf{x}) + \frac{\mathcal{E}(m)}{T+1}, \\
&\beta_{a,t} = \left(B_a + R_a\sqrt{\gamma_{t,a} + 2 + 2\log(1/\delta)}\right), \\
&\mathcal{E}(m) = \mathcal{O}(C^{2L}L^{3/2}m^{11/36}).
\end{aligned}$$

Proof. To prove Lemma 4.1.5, we analyse the left-hand side as follow:

$$\begin{aligned}
&|f_a(\mathbf{x}) - a(\mathbf{x};\mathbf{W}_{t-1})| \\
&\leq \underbrace{|f_a(\mathbf{x}) - \langle \mathbf{g}_a(\mathbf{x}_t;\mathbf{W}_0), \mathbf{U}_{a,t-1}^{-1}\mathbf{G}_{a,t-1}\mathbf{y}_{a,t-1}\rangle|}_{T_1} + \underbrace{|a(\mathbf{x};\mathbf{W}_{t-1}) - \langle \mathbf{g}_a(\mathbf{x}_t;\mathbf{W}_0), \mathbf{U}_{a,t-1}^{-1}\mathbf{G}_{a,t-1}\mathbf{y}_{a,t-1}\rangle|}_{T_2}
\end{aligned}$$

Here, T_1 represents the difference between the actual function value and the theoretical optimal solution for a linearized network. Meanwhile, T_2 refers to the gap between the neural network's output $a(\mathbf{x}; \boldsymbol{W}_{t-1})$ at iteration $t-1$ and the theoretical optimal solution for the same linearized network.

Bound term T_1 :

First of all, following Assumption 4.1.2 in the main paper, we assume that f_a is in RKHS $\mathcal{H}_{k_a}$ with NTK kernel k_a, and $\mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)$ can be considered as finite approximation of $\varphi(\cdot)$, the feature map of the NTK from $\mathbb{R}^d \to \mathcal{H}_{k_a}$. From Lemma A.1.5.4 , there exists $f_a^* \in \mathbb{R}^p$ such that $f_a(\mathbf{x}) = \langle \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0), f_a^* \rangle = \mathbf{g_a}(\mathbf{x}; \mathbf{W}_0)^\top f_a^*$. Then the term T_1 can be bounded as:

$$
\begin{aligned}
T_1 &= \left| f(\mathbf{x}) - \langle \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0); \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \mathbf{y}_{a,t-1} \rangle \right| \\
&= \left| f(\mathbf{x}) - \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \mathbf{y}_{a,t-1} \right| \\
&\leq \left| f(\mathbf{x}) - \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \mathbf{f}_{a,t-1} \right| + \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \boldsymbol{\epsilon}_{a,t-1} \right| \\
&= \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top f_a^* - \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \mathbf{G}_{a,t-1}^\top f_a^* \rangle \right| + \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \boldsymbol{\epsilon}_{a,t-1} \right| \\
&= \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \left(\mathbf{I} - \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \mathbf{G}_{a,t-1}^\top \right) f_a^* \right| + \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \boldsymbol{\epsilon}_{a,t-1} \right| \\
&= \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \left(\mathbf{I} - \mathbf{U}_{a,t-1}^{-1} \left(\mathbf{U}_{t-1} - \mathbf{I} \right) \right) f_a^* \right| + \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \boldsymbol{\epsilon}_{a,t-1} \right| \\
&= \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{w} \right| + \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \boldsymbol{\epsilon}_{a,t-1} \right| \\
&\leq \| f_a^* \|_{k_a} \left\| \mathbf{U}_{a,t-1}^{-1} \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0) \right\|_{k_a} + \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \boldsymbol{\epsilon}_{a,t-1} \right| \\
&\leq \| f_a^* \|_{k_a} \sqrt{\mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)} + \left| \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \boldsymbol{\epsilon}_{a,t-1} \right| \\
&\leq \sqrt{2} B_a \sigma_{a,t}(\mathbf{x}) + \sigma_{a,t}(\mathbf{x}) R \sqrt{\log \det(\mathbf{I} + \mathbf{H}_0) + 2 \log(1/\delta)} \\
&\leq \sqrt{2} B_a \sigma_{a,t}(\mathbf{x}) + R \sqrt{\gamma_{a,t} + 2 + 2 \log(1/\delta)} \sigma_{a,t}(\mathbf{x}) \\
&= \left(\sqrt{2} B_a + R_a \sqrt{\gamma_{a,t} + 2 + 2 \log(1/\delta)} \right) \sigma_{a,t}(\mathbf{x})
\end{aligned}
$$

where the first inequality uses triangle inequality and the fact that $\mathbf{y}_{a,t-1} = \mathbf{f}_{a,t-1} + \boldsymbol{\epsilon}_{a,t-1}$. The second inequality is from the reproducing property of function relying on RKHS, and the fourth equality is from the verification noted in Eqn. A.1.5. The last inequality directly uses the results from Lemma A.1.5.2 and Lemma A.1.5.3.

Bound term T_2 To bound term T_2, we again divide $T2$ into two terms:

$$
\begin{aligned}
T_2 &= |a(\mathbf{x}; \mathbf{W}_{t-1}) - \langle \mathbf{g}_a(\mathbf{x}_t; \mathbf{W}_0), \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \mathbf{y}_{a,t-1} \rangle| \\
&= \underbrace{|a(\mathbf{x}; \mathbf{W}_{t-1}) - \langle \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0), \mathbf{W}_{t-1} - \mathbf{W}_0 \rangle|}_{T_2'} \\
&\quad + \underbrace{|\langle \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0), \mathbf{W}_{t-1} - \mathbf{W}_0 \rangle - \langle \mathbf{g}_a(\mathbf{x}_t; \mathbf{W}_0), \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \mathbf{y}_{a,t-1} \rangle|}_{T_2''} \\
&\leq C^{2L} L^{3/2} m^{11/36} / (T+1) + C_1^{2L} L^{1/2} m^{-1/36}
\end{aligned}
$$

Here, T_2' is the difference between the network output and its linear approximation, while T_2'' indicates the gap between the network's linear approximation and the theoretical optimal solution for a linearized network. The first inequality uses lemma B.1.8 and Lemma B.1.9. Combining the bound of term T_1 and T_2, then given any $\delta \in (0,1)$, with probability at least $1-\delta\exp\big(\Omega(C^{-L}m^{1/36})\big)$, we have:

$$|f_a(\mathbf{x}) - a(\mathbf{x};\mathbf{W}_{t-1})| \leq \left(\sqrt{2}B + R\sqrt{\gamma_{t,a} + 2 + 2\log(1/\delta)}\right)\sigma_{a,t-1}(\mathbf{x}) + \frac{\mathcal{E}(m)}{T+1},$$

where $\mathcal{E}(m) = \mathcal{O}(C^{2L}L^{3/2}m^{11/36})$. □

B.1.2 Proof of Theorem 4.2.2

Theorem 4.2.2. *Under Assumption 4.1.2, Assumption 4.1.3 and Condition 4.1.4, set the step size used to train the neural networks in Alg. 3 as $\alpha_t \leq \frac{\nu}{(T+1)^2}$, then for any $\delta \in (0,1)$, with probability at least $1-\delta\exp\big(\Omega(C^{-L}m^{1/36})\big)$, the Cumulative Regret R_T and Cumulative Violation $V_{c_i,T}$ after T iterations are bounded as:*

$$V_{c_i,T} \leq 2\beta_{c_i,T}\sqrt{\frac{S_i T}{\log(S_i+1)}(2\gamma_{c_i,T}+1)} + 2\mathcal{E}(m),$$
$$R_T \leq R_T^+ \leq 2\beta_{f,T}\sqrt{\frac{BT}{\log(B+1)}(2\gamma_{f,T}+1)} + 2\mathcal{E}(m),$$

where $\mathcal{E}(m) = \mathcal{O}(C^{2L}L^{3/2}m^{11/36})$. Especially, by choosing $m = \Omega(d^9\exp\big(\nu LC^L\log T\big)))$, the Cumulative Regret and Cumulative Violation enjoy the following results:

$$V_{c_i,T} = \mathcal{O}(\gamma_{c_i,T}\sqrt{T}), \quad R_T = \mathcal{O}(\gamma_{f,T}\sqrt{T}).$$

Proof. We gradually provide the upper bound of the cumulative regret R_T and cumulative violation of each constraints $V_{c_i,T}$ as:

Bound Cumulative Regret R_T: We utilize some results from (Tran-The et al., 2022) to bound our cumulative regret:

Lemma B.1.1 (Lemma 10, (Tran-The et al., 2022)). *There exist constant $C > 0$ such that*

$$r_t \leq r_t^+ \leq \left(C + \sqrt{2}\pi(B+\sqrt{2})\right)\left(Im_t(\mathbf{x}) + \beta_{f,t}\sigma_{f,t-1}(\mathbf{x}_t)\right),$$

where $Im_t(\mathbf{x}) = \max(0, f(\mathbf{x}_t) - \mu_t^+ + \mathcal{E}(m))$, and $\mu_t^+ = \max_{\mathbf{x}_k\in\mathcal{D}_{t-1}} v(\mathbf{x}_k;\boldsymbol{\theta}_{t-1})$ is the best value of the mean objective function so far.

Lemma B.1.2 (Lemma 14, (Tran-The et al., 2022)). *Pick $\delta \in (0,1)$. Then with probability at least $1-\delta$ we have that:*

$$\sum_{t=1}^{T} Im_t(\mathbf{x}_t) = \mathcal{O}(\beta_T\sqrt{T\gamma_T}) + \mathcal{E}(m).$$

We obtain an upper bound for the Cumulative Regret R_T as follows:

$$
\begin{aligned}
R_T \leq R_T^+ &= \sum_{t=1}^{T} r_t^+ \\
&\leq \sum_{t=1}^{T} \left(C + \sqrt{2}\pi(B+\sqrt{2})\right) \left(\mathrm{Im}_t(\mathbf{x}) + \beta_{f,t}\sigma_{f,t-1}(\mathbf{x}_t)\right) \\
&\leq \sum_{t=1}^{T} \left(C + \sqrt{2}\pi(B+\sqrt{2})\right) \mathrm{Im}_t(\mathbf{x}) + \sum_{t=1}^{T} \left(C + \sqrt{2}\pi(B+\sqrt{2})\right) \beta_{f,t}\sigma_{f,t-1}(\mathbf{x}_t) \\
&\leq \left(C + \sqrt{2}\pi(B+\sqrt{2})\right) \sum_{t=1}^{T} \mathrm{Im}_t(\mathbf{x}) + \left(C + \sqrt{2}\pi(B+\sqrt{2})\right) \beta_{f,T} \sum_{t=1}^{T} \sigma_{f,t-1}(\mathbf{x}) \\
&\leq \left(C + \sqrt{2}\pi(B+\sqrt{2})\right) \sum_{t=1}^{T} \mathrm{Im}_t(\mathbf{x}) + \left(C + \sqrt{2}\pi(B+\sqrt{2})\right) \beta_{f,T} \max\left(\sum_{t=1}^{T} \sigma_{f,t-1}(\mathbf{x}), B\right) \\
&\leq 2\beta_{f,T}\sqrt{\frac{BT}{\log(B+1)}(2\gamma_{f,T}+1)} + 2\mathcal{E}(m)
\end{aligned}
$$

The first inequality is from Lemma B.1.1 and the last inequality is due to Lemma B.1.2 and Lemma B.1.12.

Bound Cumulative Violation $V_{c_i,T}$:

$$
\begin{aligned}
V_{c_i,T} &= \sum_{t=1}^{T} [c_i(\mathbf{x}_t)]^+ \\
&= \sum_{t=1}^{T} [c_i(\mathbf{x}_t) - \mathrm{LCB}_{c_i,t}(\mathbf{x}, \boldsymbol{\omega}_{c_i,t}) + \mathrm{LCB}_{c_i,t}(\mathbf{x}, \boldsymbol{\omega}_{c_i,t})]^+ \\
&\leq \sum_{t=1}^{T} [c_i(\mathbf{x}_t) - \mathrm{LCB}_{c_i,t}(\mathbf{x}, \boldsymbol{\omega}_{c_i,t})]^+ + \sum_{t=1}^{T} [\mathrm{LCB}_{c_i,t}(\mathbf{x}, \boldsymbol{\omega}_{c_i,t})]^+ \\
&= \sum_{t=1}^{T} [c_i(\mathbf{x}_t) - \mathrm{LCB}_{c_i,t}(\mathbf{x}, \boldsymbol{\omega}_{c_i,t})]^+ \\
&\leq \sum_{t=1}^{T} [\mathrm{UCB}_{c_i,t}(\mathbf{x}, \boldsymbol{\omega}_{c_i,t}) - \mathrm{LCB}_{c_i,t}(\mathbf{x}, \boldsymbol{\omega}_{c_i,t})]^+ \\
&\leq 2\beta_{c_i,t} \sum_{t=1}^{T} \sigma_{c_i,t}(\mathbf{x}) + \frac{2\mathcal{E}(m)}{T+1} \\
&\leq 2\beta_{c_i,T} \max\left(\sum_{t=1}^{T} (\sigma_{c_i,t}(\mathbf{x}), S_i) + \frac{2\mathcal{E}(m)}{T+1}\right) \\
&\leq 2\left(S_i + R_a\sqrt{\gamma_{a,T} + 2 + 2\log(1/\delta)}\right) \sqrt{\frac{S_i T}{\log(S_i+1)}(2\gamma_{c_i,T}+1)} + 2\mathcal{E}(m)
\end{aligned}
$$

□

The first inequality follows by the fact that $[a+b]^+ \leq [a]^+ + [b]^+, \forall a, b \in \mathbb{R}$. The second equality is from the feasibility condition in Alg 3. The second inequality is from Corollary 4.1.6 and the last inequality is from Lemma 4.1.5.

B.1.3 Technical Lemmas

The following lemma gives the concentration of NTK at the initialization of the neural network introduced in Eqn. 4.1.

Lemma B.1.3 (Theorem 1, (Xu and Zhu, 2024)). *Under Gaussian initialization, for* $m \geq Cd^2 \exp(L^2)$ *for some constant* C*, there exist constants* C_1, C_2 *and* C_3 *such that, with probability at least* $1 - \exp(-C_1 m^{1/3})$*,*

$$\left\|\mathbf{H}_0^{(l)} - \mathbf{\Phi}^{(l)}\right\|_\infty \leq C_2 \left(\frac{C_3^L}{m^{1/6}} + \sqrt{\frac{dL\log m}{m}}\right), \forall 1 \leq l \leq L,$$

where $\mathbf{\Phi}^{(l)}$ *is a deterministic kernel matrix. For more details about the recursive definition of* $\mathbf{\Phi}^{(l)}$*, see Section 4.1 of Xu and Zhu (2024).*

The next lemma shows the bound of difference between the gradient of neural network $a(\mathbf{x}; \mathbf{W})$ at step t and initialization.

Lemma B.1.4 (Proposition 9, (Xu and Zhu, 2024)). *Consider the neural network introduced in Eqn. 4.1 and assume that the condition 4.1.4 holds. With probability* $1 - \exp(\Omega(C^{-L} m^{1/36}))$*, for any sample path* $\{\mathbf{x}_s, y_s\}_{s=0}^{T-1}$*, all* $t \leq T$*, we have*

$$\sup_{\mathbf{x}} \|\mathbf{g}_a(\mathbf{x}, \mathbf{W}_t) - \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)\|_2 \leq C_1^{2L} L^{1/2} m^{-1/36}$$
$$\|\mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)\|_2 \leq C_2^L L^{1/2},$$

for some constants C_1 *and* C_2*.*

The next lemma shows the reproducing property of function f_a being assumed to belong to RKHS induced by NTK kernel k_a of the neural network $a(\mathbf{x}; \mathbf{W})$ introduced in Eqn. 4.1.

Lemma B.1.5. *Let* f_a *be a member of* $\mathcal{H}_{k_a}$ *with bounded RKHS norm* $\|f_a\|_{\mathcal{H}_{k_a}} \leq B_a$*. Assume that the network width of the model used to estimate function* $f_a(\cdot)$ *satisfies the Condition 4.1.4, then* $\forall \mathbf{x} \in D$*, there exists* $f_a^* \in \mathbb{R}^p$*, where* $p = md + m^2(L-2) + m$ *such that:*

$$f_a(\mathbf{x}) = \langle \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0), f_a^* \rangle = \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)^\top f_a^*$$

Proof of Lemma A.1.5.4. Due to Lemma B.1.3, with probability at least $1 - \exp(-C_1 m^{1/3} \log L)$, we have:

$$\|\mathbf{H}_0 - \mathbf{\Phi}\|_\infty \leq C_2 L \left(\frac{C_3^L}{m^{1/6}} + \sqrt{\frac{dL\log m}{m}}\right).$$

It is noted that following our definition, $\mathbf{H}_0 = \mathbf{G}_{a,t}^\top \mathbf{G}_{a,t}$. That leads to:

$$\begin{aligned}\frac{1}{\sqrt{m}}\left\|\mathbf{G}_{a,t}^\top \mathbf{G}_{a,t} - \mathbf{\Phi}\right\|_F &\leq \frac{t}{\sqrt{m}}\left\|\mathbf{G}_{a,t-1}^\top \mathbf{G}_{a,t-1} - \mathbf{\Phi}\right\|_\infty \\ &\leq \frac{t}{C_2\sqrt{m}L}\left(\frac{C_3^L}{m^{1/6}} + \sqrt{\frac{dL\log m}{m}}\right) \leq \lambda_0,\end{aligned}$$

Where λ_0 is a constant which is independent of m. The second inequality is from the choice of m in Condition 4.1.4. Then, we have:

$$\begin{aligned}\frac{1}{\sqrt{m}}\mathbf{G}_{a,t}^\top\mathbf{G}_{a,t} &\succcurlyeq \frac{1}{\sqrt{m}}\left(\mathbf{\Phi} - \left\|\mathbf{G}_{a,t}^\top\mathbf{G}_{a,t} - \mathbf{\Phi}\right\|_F\mathbf{I}\right)\\ &\succcurlyeq \frac{1}{\sqrt{m}}\left(\mathbf{\Phi} - \lambda_0\mathbf{I}\right) \succ 0,\end{aligned}$$

suggests that $\mathbf{G}_{a,t}^\top\mathbf{G}_{a,t}$ is positive definite. Thus, suppose the singular value decomposition of $\mathbf{F}_{a,t-1}$ is $\mathbf{G}_{a,t} = \mathbf{P}_{a,t}\mathbf{A}_{a,t}\mathbf{Q}_{a,t}^\top$, then by choosing $f_a^* = \mathbf{P}_{a,t}\mathbf{A}_{a,t}\mathbf{Q}_{a,t}^\top\mathbf{F}_{a,t}$, we have

$$\mathbf{G}_{a,t-1}^\top f_a^* = \mathbf{Q}_{a,t}\mathbf{A}_{a,t}\mathbf{P}_{a,t}^\top\mathbf{P}_{a,t}\mathbf{A}_{a,t}\mathbf{Q}_{a,t}^\top\mathbf{F}_{a,t} = \mathbf{F}_{a,t},$$

which indicates that for any $\mathbf{x}$, $\langle g(\mathbf{x};\mathbf{W}_0), f_a^*\rangle = f_a(\mathbf{x})$. □

Let $\mathbf{z}_t^{(l)}(\mathbf{x})$ measure the sensitivity of the output from the l-th hidden layer and defined as:

$$\begin{aligned}[\mathbf{z}_t^{(l)}(\mathbf{x})]^\top &= \left[\frac{\partial a(\mathbf{x};\mathbf{W}_t)}{\partial \mathbf{h}_t^{(l)}(\mathbf{x})}\right]^\top\\ &= \mathbf{q}^\top\frac{1}{\sqrt{m}}\mathbf{D}_t^{(L)}(\mathbf{x})\mathbf{W}_t^{(L)}\cdots\frac{1}{\sqrt{m}}\mathbf{D}_t^{(l+1)}(\mathbf{x})\mathbf{W}_t^{(l+1)},\end{aligned}$$

Then the following lemma provides the bound of the difference between $\mathbf{z}_t^{(l)}(\mathbf{x})$ and $\mathbf{z}_0^{(l)}(\mathbf{x})$:

Lemma B.1.6 (Lemma 12, (Xu and Zhu, 2024)). *Consider the neural network introduced in Eqn. 4.1 and assume that the condition 4.1.4 holds. With probability* $1 - \exp\left(\Omega(C_1^{-L+l}m^{1/36})\right)$, *for layer* l *and any sample path* $\{\mathbf{x}_s, y_s\}_{s=0}^{T-1}$, *with all* $t \le T$, *we have:*

$$\begin{aligned}\sup_{\mathbf{x}}\left\|\mathbf{z}_t^{(l)}(\mathbf{x}) - \mathbf{z}_0^{(l)}(\mathbf{x})\right\|_2 &\le \mathcal{O}(C_1^{2L-l}m^{17/36})\\ \sup_{\mathbf{x}}\left\|\mathbf{z}_0^{(l)}(\mathbf{x})\right\|_2 &\le C_2^{L-l}\sqrt{m}\\ \sup_{\mathbf{x}}\left\|\mathbf{z}_t^{(l)}(\mathbf{x})\right\|_2 &\le C_3^{2L-l-1}\sqrt{m}\end{aligned}$$

for some absolute constant C_1, C_2, C_3.

The following lemma provides the bound on the difference between neural network weights and output at initialization and at step t:

Lemma B.1.7 (Lemma 10, (Xu and Zhu, 2024)). *Consider the neural network introduced in Eqn. 4.1 and assume that the condition 4.1.4 holds. Setting the step size at training step* t *as* $\alpha_t \le \frac{\nu}{(T+1)^2}$, *then with probability* $1 - \exp\left(\Omega(C^{-L}m^{1/36})\right)$, *for any sample path*

$\{\mathbf{x}_s, y_s\}_{s=0}^{T-1}$, *all* $t \leq T$, *we have:*

$$\begin{aligned}\left\|\mathbf{W}_t^{(l)} - \mathbf{W}_0^{(l)}\right\|_2 &\leq \frac{m^{1/3}L^{1/2}}{T+1}\\ \left\|\mathbf{W}_0^{(l)}\right\|_2, \left\|\mathbf{W}_t^{(l)}\right\|_2 &\leq \mathcal{O}(\sqrt{m})\\ \sup_{\mathbf{x}}\left\|\mathbf{h}_t^{(l)}(\mathbf{x}) - \mathbf{h}_0^{(l)}(\mathbf{x})\right\|_2 &\leq \frac{C_3^l}{m^{1/6}},\end{aligned}$$

for some absolute constant C_3.

The following lemmas provide bound on the technical terms used in Lemma 4.1.5.

Lemma B.1.8. *Let* $a(\mathbf{x};\mathbf{W})$ *is the neural network defined in Eqn. 4.1. Then, with probability* $1-\exp\big(\Omega(C^{-L}m^{1/36})\big)$, *we have:*

$$|a(\mathbf{x},\mathbf{W}_{t-1}) - a(\mathbf{x},\mathbf{W}_0) - \langle \mathbf{g}_a(\mathbf{x},\mathbf{W}_0), \mathbf{W}_{t-1}-\mathbf{W}_0\rangle| \leq \mathcal{O}(C^{2L}L^{3/2}m^{11/36})$$

Proof of Lemma B.1.8. Remind that

$$\mathbf{h}^{(l)}(\mathbf{x}) = \frac{1}{\sqrt{m}}\mathbf{D}^{(l)}(\mathbf{x})\mathbf{W}^{(l)}\ldots\frac{1}{\sqrt{m}}\mathbf{D}^{(1)}(\mathbf{x})\mathbf{W}^{(1)}\mathbf{x},$$

Then, by direct calculation, we have

$$\begin{aligned}\frac{\partial a(\mathbf{x};\mathbf{W}_0)}{\partial \mathbf{W}^{(l)}} &= \frac{\mathbf{q}^\top}{\sqrt{m}}\mathbf{D}_0^{(L)}(\mathbf{x})\mathbf{W}_0^{(L)}\ldots\frac{1}{\sqrt{m}}\mathbf{D}_0^{(l)}(\mathbf{x})\left[\mathbf{h}_0^{(l-1)}(\mathbf{x})\right]^\top\\ &= \frac{1}{\sqrt{m}}[\mathbf{z}_0^{(l)}(\mathbf{x})]^\top\mathbf{D}_0^{(l)}(\mathbf{x})\left[\mathbf{h}_0^{(l-1)}(\mathbf{x})\right]^\top,\end{aligned}$$

and

$$\mathbf{g}_a(\mathbf{x};\mathbf{W}_0) = \left[\frac{\partial a(\mathbf{x};\mathbf{W}_0)}{\partial \mathbf{W}^{(1)}}, \frac{\partial a(\mathbf{x};\mathbf{W}_0)}{\partial \mathbf{W}^{(2)}}, \ldots, \frac{\partial a(\mathbf{x};\mathbf{W}_0)}{\partial \mathbf{W}^{(L)}}\right]$$

We also rewrite $a(\mathbf{x};\mathbf{W}_{t-1})$ and $a(\mathbf{x};\mathbf{W}_0)$ as:

$$\begin{aligned}a(\mathbf{x};\mathbf{W}_{t-1}) &= \frac{\mathbf{q}^\top}{\sqrt{m}}\mathbf{D}_{t-1}^{(L)}(\mathbf{x})\mathbf{W}_{t-1}^{(L)}\ldots\frac{1}{\sqrt{m}}\mathbf{D}_{t-1}^{(1)}(\mathbf{x})\mathbf{W}_{t-1}^{(1)}(\mathbf{x})\\ &= \frac{1}{\sqrt{m}}\mathbf{z}_{t-1}^{(l)}(\mathbf{x})\mathbf{D}_{t-1}^{(l)}(\mathbf{x})\mathbf{W}_{t-1}\mathbf{h}_{t-1}^{(l-1)}(\mathbf{x}),\\ a(\mathbf{x};\mathbf{W}_0) &= \frac{\mathbf{q}^\top}{\sqrt{m}}\mathbf{D}_0^{(L)}(\mathbf{x})\mathbf{W}_0^{(L)}\ldots\frac{1}{\sqrt{m}}\mathbf{D}_0^{(1)}(\mathbf{x})\mathbf{W}_0^{(1)}(\mathbf{x})\\ &= \frac{1}{\sqrt{m}}\mathbf{z}_0^{(l)}(\mathbf{x})\mathbf{D}_0^{(l)}(\mathbf{x})\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l-1)}(\mathbf{x})\end{aligned}$$

Using the technique in the proof of Lemma 8.2 in (Allen-Zhu, Li, and Song, 2019), there exist diagonal matrices $\widehat{\mathbf{D}}^{(l)}(\mathbf{x}) = \mathbf{D}_{t-1}^{(l)}(\mathbf{x}) - \mathbf{D}_0^{(l)}(\mathbf{x}) \in \mathbb{R}^{m\times m}, \forall 1 \leq l \leq L$ with

entries in $[-1, 1]$ such that:

$$
\begin{aligned}
&a(\mathbf{x}, \mathbf{W}_{t-1}) - a(\mathbf{x}, \mathbf{W}_0) \\
&= \frac{1}{\sqrt{m}} \sum_{l=1}^{L} \left[\prod_{r=l+1}^{L} \left(\widehat{\mathbf{D}}^{(r)}(\mathbf{x}) + \mathbf{D}_{t-1}^{(r)}(\mathbf{x}) \right) \mathbf{W}_{t-1}^{(r)} \right] \left(\widehat{\mathbf{D}}^{(l)}(\mathbf{x}) + \mathbf{D}_{t-1}^{(l)}(\mathbf{x}) \right) (\mathbf{W}_{t-1}^{(l)} - \mathbf{W}_0^{(l)}) \mathbf{h}_0^{(l-1)}(\mathbf{x}) \\
&= \frac{1}{\sqrt{m}} \sum_{l=1}^{L} \widehat{\mathbf{z}}_{t-1}^{(l)} \left(\widehat{\mathbf{D}}^{(l)}(\mathbf{x}) + \mathbf{D}_{t-1}^{(l)}(\mathbf{x}) \right) \left(\mathbf{W}_{t-1}^{(l)} - \mathbf{W}_0^{(l)} \right) \mathbf{h}_0^{(l-1)}(\mathbf{x})
\end{aligned}
$$

Furthermore, we have

$$
\langle \mathbf{g}_a(\mathbf{x}, \mathbf{W}_0), \mathbf{W}_{t-1} - \mathbf{W}_0 \rangle = \frac{1}{\sqrt{m}} \sum_{l=1}^{L} \mathbf{z}_0^{(l)}(\mathbf{x}) \mathbf{D}_0^{(l)}(\mathbf{x}) \left(\mathbf{W}_{t-1}^{(l)} - \mathbf{W}_0^{(l)} \right) \mathbf{h}_0^{(l-1)}(\mathbf{x})
$$

Replacing all below expressions, we get

$$
\begin{aligned}
&|a(\mathbf{x}, \mathbf{W}_{t-1}) - a(\mathbf{x}, \mathbf{W}_0) - \langle \mathbf{g}_a(\mathbf{x}, \mathbf{W}_0), \mathbf{W}_{t-1} - \mathbf{W}_0 \rangle| \\
&= \frac{1}{\sqrt{m}} \sum_{l=1}^{L} \widehat{\mathbf{z}}_{t-1}^{(l)}(\mathbf{x}) \left(\widehat{\mathbf{D}}^{(l)}(\mathbf{x}) + \mathbf{D}_{t-1}^{(l)}(\mathbf{x}) \right) \left(\mathbf{W}_{t-1}^{(l)} - \mathbf{W}_0^{(l)} \right) \mathbf{h}_0^{(l-1)}(\mathbf{x}) \\
&- \frac{1}{\sqrt{m}} \sum_{l=1}^{L} \mathbf{z}_0^{(l)}(\mathbf{x}) \mathbf{D}_0^{(l)}(\mathbf{x}) \left(\mathbf{W}_{t-1}^{(l)} - \mathbf{W}_0^{(l)} \right) \mathbf{h}_0^{(l-1)}(\mathbf{x}) \\
&\leq \frac{1}{\sqrt{m}} \sum_{l=1}^{L} \left\| \widehat{\mathbf{z}}_{t-1}^{(l)} - \mathbf{z}_0^{(l)}(\mathbf{x}) \right\|_2 \left\| \mathbf{W}_{t-1}^{(l)} - \mathbf{W}_0^{(l)} \right\|_2 \left\| \mathbf{h}_0^{(l-1)} \right\| \\
&\leq L m^{-1/2} L^{1/2} m^{1/3} C^{2L} m^{17/36} / (T+1) \\
&\leq (C^{2L} L^{3/2} m^{11/36}) / (T+1).
\end{aligned}
$$

□

The first inequality uses triangle inequality. The second inequality is from Lemma B.1.6 and Lemma B.1.7.

Lemma B.1.9. *Let $a(\mathbf{x}; \mathbf{W})$ is the neural network defined in Eqn. 4.1. Then, with probability $1 - \exp(\Omega(C^{-L} m^{1/36}))$, we have:*

$$
|\langle \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0), \mathbf{W}_{t-1} - \mathbf{W}_0 - \mathbf{U}_{a,t-1}^{-1} \mathbf{G}_{a,t-1} \mathbf{y}_{t-1} \rangle| \leq C_1^{2L} L^{1/2} m^{-1/36}.
$$

Proof of Lemma B.1.9. Using the model update formula given in Eqn. 4.2, we have

$$
\begin{aligned}
\mathbf{W}_{t-1} - \mathbf{W}_0 &= (\mathbf{W}_{t-1} - \mathbf{W}_{t-2}) + (\mathbf{W}_{t-2} - \mathbf{W}_{t-3}) + \cdots + (\mathbf{W}_1 - \mathbf{W}_0) \\
&= \sum_{i=1}^{t-1} (\mathbf{W}_i - \mathbf{W}_{i-1}) \\
&= \sum_{i=1}^{t-1} \alpha_i \left(y_i - a(\mathbf{x}_i, \mathbf{W}_{i-1}) \right) \nabla_{\mathbf{W}} a(\mathbf{x}_i, \mathbf{W}_{i-1}) \\
&= \sum_{i=1}^{t-1} \alpha_i \left(y_i - a(\mathbf{x}_i, \mathbf{W}_{i-1}) \right) \mathbf{g}_{a,i-1}(\mathbf{x}_i, \mathbf{W}_{i-1}) \\
&= \alpha \tilde{\mathbf{G}}_{a,t-1} (\mathbf{y}_{t-1} - \mathbf{A}_{t-1}),
\end{aligned}
$$

where $\mathbf{A}_{t-1} = [a(\mathbf{x}_1, \mathbf{W}_1), \ldots, a(\mathbf{x}_{t-1}, \mathbf{W}_{t-1})] \in \mathbb{R}^{t-1}$. Then we have:

$$
\begin{aligned}
&|\mathbf{W}_{t-1} - \mathbf{W}_0 - \mathbf{U}_{a,t-1}^{-1}\mathbf{G}_{a,t-1}\mathbf{y}_{t-1}| \\
&= |\alpha\bar{\mathbf{G}}_{a,t-1}(\mathbf{y}_{t-1} - \mathbf{A}_{t-1}) - \mathbf{U}_{a,t-1}^{-1}\mathbf{G}_{a,t-1}\mathbf{y}_{t-1}| \\
&= |\alpha(\bar{\mathbf{G}}_{a,t-1} - \mathbf{G}_{a,t-1})(\mathbf{y}_{t-1} - \mathbf{A}_{t-1}) + \alpha\mathbf{G}_{a,t-1}(\mathbf{y}_{t-1} - \mathbf{A}_{t-1}) - (\mathbf{I} + \mathbf{G}_{a,t-1}\mathbf{G}_{a,t-1}^\top)^{-1}\mathbf{G}_{a,t-1}\mathbf{y}_{t-1}| \\
&= |\alpha(\bar{\mathbf{G}}_{a,t-1} - \mathbf{G}_{a,t-1})(\mathbf{y}_{t-1} - \mathbf{A}_{t-1}) + \alpha\mathbf{G}_{a,t-1}(\mathbf{y}_{t-1} - \mathbf{A}_{t-1}) - \mathbf{G}_{a,t-1}(\mathbf{I} + \mathbf{G}_{a,t-1}^\top\mathbf{G}_{a,t-1})^{-1}\mathbf{y}_{t-1}| \\
&\leq \left\|\alpha(\bar{\mathbf{G}}_{a,t-1} - \mathbf{G}_{a,t-1})(\mathbf{y}_{t-1} - \mathbf{A}_{t-1})\right\|_2 + \alpha\|\mathbf{G}_{a,t-1}\|_2\left\|\mathbf{y}_{t-1}\left[\mathbf{I} - (\alpha\mathbf{I} + \alpha\mathbf{G}_{a,t-1}^\top\mathbf{G}_{a,t-1})^{-1}\right] - \mathbf{A}_{t-1}\right\|_2 \\
&\leq |\alpha|\sqrt{t}\left\|(\bar{\mathbf{G}}_{a,t-1} - \mathbf{G}_{a,t-1})\right\|_2 + |\alpha|\sqrt{t}\|(\mathbf{G}_{a,t-1})\|_2 \\
&\leq C_1^{2L}L^{1/2}m^{-1/36}
\end{aligned}
$$

The first inequality is from the triangle inequality and the last inequality is due to the choice of $\alpha = \frac{\nu}{(T+1)^2}$, where ν is a parameter and independent of dimension d. Therefore, we have:

$$
\begin{aligned}
&|\langle \mathbf{g}_a(\mathbf{x}; \mathbf{W}_0), \mathbf{W}_{t-1} - \mathbf{W}_0 - \mathbf{U}_{a,t-1}^{-1}\bar{\mathbf{G}}_{a,t-1}\mathbf{y}_{t-1}\rangle| \\
&\leq \|\mathbf{g}_a(\mathbf{x}; \mathbf{W}_0)\|_2\left\|\mathbf{W}_{t-1} - \mathbf{W}_0 - \mathbf{U}_{a,t-1}^{-1}\bar{\mathbf{G}}_{a,t-1}\mathbf{y}_{t-1}\right\|_2 \\
&\leq C_1^{2L}L^{1/2}m^{-1/36}
\end{aligned}
$$

□

Lemma B.1.10 (Theorem 1, Chowdhury and Gopalan (2017)). *Let $\{\epsilon_{a,t}\}_{t=1}^{\infty}$ be a real-valued stochastic process such that for some $R \geq 0$ and for all $t \geq 1$, $\epsilon_{a,t}$ is $\mathcal{F}_{a,t-1}$-measurable and R-sub-Gaussian conditioned on $\mathcal{F}_{a,t-1}$. Recall $\mathbf{H}_0$ defined in Eqn. A.1.5. For a given $\eta > 0$, with probability $1 - \delta$, the following holds for all t:*

$$\boldsymbol{\epsilon}_{a,t}^\top((\mathbf{H}_0 + \eta\mathbf{I})^{-1} + \mathbf{I})^{-1}\boldsymbol{\epsilon}_{a,t} \leq R_a^2 \log\det((1+\eta)\mathbf{I} + \mathbf{H}_0) + 2R_a^2\log(1/\delta).$$

Lemma B.1.11. *Let $\delta \in (0,1)$. If the network width m satisfies Condition 4.1.4, then with probability at least $1 - \delta$, the following holds for every $t \in [T]$:*

$$\log\det(\mathbf{I} + \mathbf{H}_0) \leq 2\gamma_{a,t} + 1,$$

where $\gamma_{a,t}$ is the maximum information gain associated with the NTK kernel k_a.

Proof of Lemma A.1.5.3. From the definition of $\mathbf{H}_0$ and Lemma B.7 Zhou, Li, and Gu, 2020, we have that

$$
\begin{aligned}
\log\det(\mathbf{I} + \mathbf{H}_0) &= \log\det\left(\mathbf{I} + \sum_{i=1}^{T}\mathbf{g}(\mathbf{x}_t; \boldsymbol{\theta}_0)\mathbf{g}(\mathbf{x}_t; \boldsymbol{\theta}_0)^\top\right) \\
&= \log\det(\mathbf{I} + \mathbf{H}_0 + (\boldsymbol{\Phi} - \mathbf{H}_0)) \\
&\leq \log\det(\mathbf{I} + \mathbf{H}_0) + \langle(\mathbf{I} + \mathbf{H}_0)^{-1}, (\boldsymbol{\Phi} - \mathbf{H}_0)\rangle \\
&\leq \log\det(\mathbf{I} + \mathbf{H}_0) + \left\|(\mathbf{I} + \mathbf{H}_0)^{-1}\right\|_F\|(\boldsymbol{\Phi} - \mathbf{H}_0)\|_F \\
&\leq 2\gamma_{a,t} + 1,
\end{aligned}
$$

where the first equality is from the definition of $\mathbf{K}_t$ in Definition A.1.5, the first inequality is from the convexity of $\log\det(\cdot)$ function, and the second inequality is from the fact that $\langle \mathbf{A}, \mathbf{B}\rangle \leq \|\mathbf{A}\|_F\|\mathbf{B}\|_F$. The third inequality is from the choice of m

in Condition 4.1.4, combined with Lemma B.1.3 and Lemma 3 in Chowdhury and Gopalan, 2017. □

Lemma B.1.12 (Lemma 8, Phan-Trong, Tran-The, and Gupta (2023)). *Suppose the width of the neural network m satisfies Condition 4.1.4. Then*

$$\sum_{i=1}^{T} \min(\sigma_t(\mathbf{x}_t), B) \leq \sqrt{\frac{BT}{\log(B+1)}(2\gamma_T + 1)}.$$

Appendix C

Supplementary Material of Chapter 5

C.1 Additional Experimental Results

C.1.1 Synthetic Benchmark Functions

We present the mathematical expressions of five synthetic objective functions and their accompanied PDEs used in Section 5 of the main paper as follows:

Drop-Wave:

$$f(\mathbf{x}) = -\frac{1+\cos\left(12\sqrt{\mathbf{x}_1^2+\mathbf{x}_2^2}\right)}{0.5(\mathbf{x}_1^2+\mathbf{x}_1^2)+2} \text{ s.t. } \mathbf{x}_1\frac{\partial f}{\partial \mathbf{x}_2} - \mathbf{x}_2\frac{\partial f}{\partial \mathbf{x}_1} = 0$$

Styblinski-Tang:

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{d}(\mathbf{x}_i^4 - 16\mathbf{x}_i^2 + 5\mathbf{x}_i) \text{ s.t. } \sum_{i=1}^{d}\frac{\partial f}{\partial \mathbf{x}_i} = \sum_{i=1}^{d}(2\mathbf{x}_i^3 - 16\mathbf{x}_i + \frac{5}{2})$$

Rastrigin:

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

$$\text{s.t. } \mathbf{x}^\top\nabla f(\mathbf{x}) - f(\mathbf{x}) = 10\sum_{i=1}^{d}\left[(\cos(2\pi\mathbf{x}_i)) + \pi\mathbf{x}_i(\sin(2\pi\mathbf{x}_i)) - 1\right]$$

Michalewics:

$$f(\mathbf{x}) = -\sum_{i=1}^{d}\sin(\mathbf{x}_i)\sin^{2m}\left(\frac{i\mathbf{x}_i^2}{\pi}\right) \text{ s.t. } \mathbf{h}^\top\nabla f(\mathbf{x}) - f(\mathbf{x}) = 0,$$

$$\text{where } \mathbf{h} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_d]^\top, \mathbf{h}_i = \left[\frac{\cos(\mathbf{x}_i)}{\sin(\mathbf{x}_i)} + \frac{2\mathbf{x}_i(2m-1)}{\tan\left(\frac{i\mathbf{x}_i^2}{\pi}\right)}\right]^{-1}$$

Cosine Mixture:

$$0.1\sum_{i=1}^{d}\cos(5\pi \mathbf{x}_i)+\sum_{i=1}^{d}\mathbf{x}_i^2 \text{ s.t } \sum_{i=1}^{d}\left(\frac{\partial f}{\partial \mathbf{x}_i}-2\mathbf{x}_i+0.5\pi\sin(5\pi\mathbf{x}_i)\right)^2=0$$

C.2 Proof of Theoretical Analysis in Chapter 5

C.2.1 Proof of Lemma 5.3.4

In this section, we present the detailed proof of Lemma 5.3.4 in the main paper. Before going to the proof, we repeat Lemma 5.3.4 here:

Lemma 5.3.4. *Conditioned on* $\mathcal{D}_t=\{\mathbf{x}_i,y_i\}_{i=1}^t, \mathcal{R}=\{\mathbf{z}_j,u_j\}_{j=1}^{N_r}$, *the acquisition function* $\widetilde{f}_t(\mathbf{x})=h(\mathbf{x};\boldsymbol{\theta}_{t-1})$ *can be viewed as a random draw from a* GP $\left(\mu_t^f(\mathbf{x}),\nu_t^2\left(\sigma_t^f\right)^2(\mathbf{x})\right)$ *with the following mean and covariance functions:*

$$\mu_t^f(\mathbf{x})=\phi(\mathbf{x})^\top\boldsymbol{\xi}_t^\top\widehat{\mathbf{K}}_{\text{PINN}}^{-1}\begin{bmatrix}\mathbf{Y}_t\\ \mathbf{U}_r\end{bmatrix}$$
$$\left(\sigma_t^f\right)^2(\mathbf{x})=\langle\phi(\mathbf{x}),\phi(\mathbf{x})\rangle-\phi(\mathbf{x})^\top\boldsymbol{\xi}_t^\top\widehat{\mathbf{K}}_{\text{PINN}}^{-1}\boldsymbol{\xi}_t\phi(\mathbf{x}),$$

where

$$\widehat{\mathbf{K}}_{\text{PINN}}^{-1}=\begin{bmatrix}\mathbf{K}_{uu}+\lambda_1\mathbf{I} & \mathbf{K}_{ur}\\ \mathbf{K}_{ru} & \mathbf{K}_{rr}+\lambda_2\mathbf{I}\end{bmatrix}^{-1}=\begin{bmatrix}\widetilde{\mathbf{A}} & \widetilde{\mathbf{B}}\\ \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}}\end{bmatrix}$$
$$\boldsymbol{\Phi}_t=[\phi(\mathbf{x}_1)^\top,\ldots,\phi(\mathbf{x}_t)^\top]^\top$$
$$\boldsymbol{\Omega}_r=[\omega(\mathbf{z}_1)^\top,\ldots,\omega(\mathbf{z}_{N_r})^\top]^\top,\ \boldsymbol{\xi}_t=\begin{bmatrix}\boldsymbol{\Phi}_t^\top & \boldsymbol{\Omega}_r^\top\end{bmatrix}^\top$$
$$\mathbf{K}_{uu}=\boldsymbol{\Phi}_t\boldsymbol{\Phi}_t^\top,\ \mathbf{K}_{ur}=\boldsymbol{\Phi}_t\boldsymbol{\Omega}_r^\top,\mathbf{K}_{ru}=\mathbf{K}_{ur}^\top,\mathbf{K}_{rr}=\boldsymbol{\Omega}_r\boldsymbol{\Omega}_r^\top$$
$$\mathbf{Y}_t=[y_1,y_2,\ldots,y_t]^\top,\mathbf{U}_r=[u_1,u_2,\ldots,u_{N_r}]^\top$$

To prove Lemma 5.3.4, we need the following lemma:

Lemma C.2.1. *(Theorem 4.1 Wang, Yu, and Perdikaris (2022)) A sufficiently wide physics-informed neural network for modeling the problem defined in Section 4 induces a joint multivariate Gaussian distribution between the network outputs and its "derivatives" after applying the differential operator to this network*

$$\begin{bmatrix}\mathbf{h}_\theta\\ \mathbf{g}_\theta\end{bmatrix}\sim N(\mathbf{0},\mathbf{K}_{\text{NTK}-\text{PINN}}),\tag{C.1}$$

where $\mathbf{K}_{\text{NTK}-\text{PINN}} = \begin{bmatrix} \mathbf{K}_{uu} & \mathbf{K}_{ur} \\ \mathbf{K}_{ru} & \mathbf{K}_{rr} \end{bmatrix}$ *is the NTK matrix of PINN, with*

$$(\mathbf{K}_{uu})_{ij} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \tag{C.2}$$
$$(\mathbf{K}_{ur})_{ij} = \langle \phi(\mathbf{x}_i), \omega(\mathbf{z}_j) \rangle \tag{C.3}$$
$$(\mathbf{K}_{uu})_{ij} = \langle \omega(\mathbf{z}_i), \omega(\mathbf{z}_j) \rangle \tag{C.4}$$
$$\mathbf{K}_{ru} = \mathbf{K}_{ur}^{\top}, \tag{C.5}$$
$$\mathbf{h}_{\theta} = [h(\mathbf{x}_1, \theta), \ldots, h(\mathbf{x}_t, \theta)]^{\top} \tag{C.6}$$
$$\mathbf{g}_{\theta} = [\mathcal{N}[h](\mathbf{z}_1, \theta), \ldots, \mathcal{N}[h](\mathbf{z}_{N_r}, \theta)]^{\top} \tag{C.7}$$

where $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_t$ *and* $\mathbf{z}_i, \mathbf{z}_j \in \mathcal{R}$ *are two arbitrary points belonging to the set* $\mathcal{R}$ *defined in Algorithm 1 and* $\mathcal{N}[h]$ *denotes a differential operator of the neural network* $h(\cdot, \theta)$ *with respect to the input* $\mathbf{x}$*.*

Corollary C.2.1.1. The unknown reward function values $f_{1:t}$ and the PDE observations $g_{1:N_r}$ are jointly Gaussian with an initial prior distribution with zero mean and the covariance $\nu_t^2 \mathbf{K}_{\text{NTK}-\text{PINN}}$, where ν_t is the exploration coefficient introduced in Section 4.

$$\begin{bmatrix} f_{1:t} \\ g_{1:N_r} \end{bmatrix} \sim N(\mathbf{0}, \nu_t^2 \mathbf{K}_{\text{NTK}-\text{PINN}}), \tag{C.8}$$

Proof of Corollary C.2.1.1. The algorithm updates the neural network by minimizing the loss function:

$$\mathcal{L}(t) = \sum_{i=1}^{t-1} [y_i - \nu_t h(\mathbf{x}_i; \theta_{t-1})]^2 + \sum_{j=1}^{N_r} [u_j - \nu_t \mathcal{N}[h](\mathbf{z}_j; \theta_{t-1})]^2 \tag{C.9}$$

As the function prediction and its "derivatives" prediction of each observation $\mathbf{x}_i$ and $\mathbf{z}_j$ at iteration t is modeled by $\nu_t h(\mathbf{x}_i; \theta_{t-1})$ and $\nu_t \mathcal{N}[h](\mathbf{z}_j; \theta_{t-1})$, it is clear to see, from Lemma C.2.1, that the function values of the unknown function f can be assumed to follow a joint Gaussian prior with zero means and covariance matrix $\nu_t^2 \mathbf{K}_{\text{NTK}-\text{PINN}}$. A similar argument can be applied to the function g, where $g(\cdot) = \mathcal{N}[f](\cdot)$ with $\mathcal{N}[f]$ is a differential operator. We are now ready to prove 5.3.4. □

Proof of Lemma 5.3.4. From Corollary C.2.1.1, the priors for values of both f and g follow a joint Gaussian distribution with kernel $\mathbf{K}_{\text{NTK}-\text{PINN}}$. Let $\mathbf{K}_{\mathbf{x}}$ be the NTK matrix between point $\mathbf{x}$ and all training data: $\mathbf{K}_x = \begin{bmatrix} \Sigma_a & \Sigma_b \\ \Sigma_c & \Sigma_d \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}_t \phi(\mathbf{x}) & \mathbf{\Phi}_t \omega(\mathbf{x}) \\ \mathbf{\Omega}_r \phi(\mathbf{x}) & \mathbf{\Omega}_r \omega(\mathbf{x}) \end{bmatrix}$. Then the the posterior of f and g evaluated at an input $\mathbf{x}$ will be a Gaussian distribution with mean and variance functions:

Mean function:

$$
\begin{aligned}
&\begin{bmatrix} \mu_t^f(\mathbf{x}) \\ \mu_t^g(\mathbf{x}) \end{bmatrix} = \mathbf{K}_{\mathbf{x}}^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \begin{bmatrix} \mathbf{Y}_t \\ \mathbf{U}_r \end{bmatrix} \\
&= \begin{bmatrix} \Sigma_a^\top & \Sigma_c^\top \\ \Sigma_b^\top & \Sigma_d^\top \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} \\ \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} \end{bmatrix} \begin{bmatrix} \mathbf{Y}_t \\ \mathbf{U}_r \end{bmatrix} \\
&= \begin{bmatrix} \phi(\mathbf{x})^\top \mathbf{\Phi}_t^\top & \phi(\mathbf{x})^\top \mathbf{\Omega}_r^\top \\ \omega(\mathbf{x})^\top \mathbf{\Phi}_t^\top & \omega(\mathbf{x})^\top \mathbf{\Omega}_r^\top \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} \\ \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} \end{bmatrix} \begin{bmatrix} \mathbf{Y}_t \\ \mathbf{U}_r \end{bmatrix} \\
&= \begin{bmatrix} \phi(\mathbf{x})^\top \mathbf{\Phi}_t^\top \widetilde{\mathbf{A}} \mathbf{Y}_t + \phi(\mathbf{x})^\top \mathbf{\Omega}_r^\top \widetilde{\mathbf{C}} \mathbf{Y}_t + \phi(\mathbf{x})^\top \mathbf{\Phi}_t^\top \widetilde{\mathbf{B}} \mathbf{U}_r + \phi(\mathbf{x})^\top \mathbf{\Omega}_r^\top \widetilde{\mathbf{D}} \mathbf{U}_r \\ \omega(\mathbf{x})^\top \mathbf{\Phi}_t^\top \widetilde{\mathbf{A}} \mathbf{Y}_t + \omega(\mathbf{x})^\top \mathbf{\Omega}_r^\top \widetilde{\mathbf{C}} \mathbf{Y}_t + \omega(\mathbf{x})^\top \mathbf{\Phi}_t^\top \widetilde{\mathbf{B}} \mathbf{U}_r + \omega(\mathbf{x})^\top \mathbf{\Omega}_r^\top \widetilde{\mathbf{D}} \mathbf{U}_r \end{bmatrix} \\
&= \phi(\mathbf{x})^\top \boldsymbol{\xi}_t^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \begin{bmatrix} \mathbf{Y}_t \\ \mathbf{U}_r \end{bmatrix}
\end{aligned}
$$

Variance function: Let $\mathbf{K}_{\mathbf{xx}} = \begin{bmatrix} \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle & \langle \phi(\mathbf{x}), \omega(\mathbf{x}) \rangle \\ \langle \omega(\mathbf{x}), \phi(\mathbf{x}) \rangle & \langle \omega(\mathbf{x}), \omega(\mathbf{x}) \rangle \end{bmatrix}$, then we have the posterior covariance matrix of f and g at input $\mathbf{x}$ is:

$$
\begin{bmatrix} \text{Cov}_f(\mathbf{x}) & \text{Cov}_{fg}(\mathbf{x}) \\ \text{Cov}_{gf}(\mathbf{x}) & \text{Cov}_g(\mathbf{x}) \end{bmatrix} \tag{C.10}
$$

$$
= \nu_t^2 \left(\mathbf{K}_{\mathbf{xx}} - \mathbf{K}_{\mathbf{x}}{}^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \mathbf{K}_{\mathbf{x}} \right) \tag{C.11}
$$

$$
= \nu_t^2 \begin{bmatrix} \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle & \langle \phi(\mathbf{x}), \omega(\mathbf{x}) \rangle \\ \langle \omega(\mathbf{x}), \phi(\mathbf{x}) \rangle & \langle \omega(\mathbf{x}), \omega(\mathbf{x}) \rangle \end{bmatrix} - \nu_t^2 \begin{bmatrix} \Sigma_a^\top & \Sigma_c^\top \\ \Sigma_b^\top & \Sigma_d^\top \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} \\ \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} \end{bmatrix} \begin{bmatrix} \Sigma_a & \Sigma_b \\ \Sigma_c & \Sigma_d \end{bmatrix} \tag{C.12}
$$

Therefore,

$$
\text{Cov}_f(\mathbf{x}) = \nu_t^2 \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - \nu_t^2 (\Sigma_a^\top \widetilde{\mathbf{A}} \Sigma_a + \Sigma_c^\top \widetilde{\mathbf{C}} \Sigma_c + \Sigma_a^\top \widetilde{\mathbf{B}} \Sigma_a + \Sigma_c^\top \widetilde{\mathbf{D}} \Sigma_c) \tag{C.13}
$$

$$
= \nu_t^2 \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - \nu_t^2 \begin{bmatrix} \Sigma_a^\top & \Sigma_c^\top \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} \\ \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} \end{bmatrix} \begin{bmatrix} \Sigma_a \\ \Sigma_c \end{bmatrix} \tag{C.14}
$$

$$
= \nu_t^2 \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - \nu_t^2 \begin{bmatrix} \phi(\mathbf{x})^\top \mathbf{\Phi}_t^\top & \phi(\mathbf{x})^\top \mathbf{\Omega}_r^\top \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} \\ \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_t \phi(\mathbf{x}) \\ \mathbf{\Omega}_r \phi(\mathbf{x}) \end{bmatrix} \tag{C.15}
$$

$$
= \nu_t^2 \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - \nu_t^2 \phi(\mathbf{x})^\top \boldsymbol{\xi}_t^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \boldsymbol{\xi}_t \phi(\mathbf{x}) \tag{C.16}
$$

$$
= \nu_t^2 (\sigma_t^f)^2(\mathbf{x}) \tag{C.17}
$$

□

C.2.2 Proof of Lemma C.2.2

Lemma C.2.2. *The interaction information between f and observation $\mathbf{Y}_t$ and PDE data $\mathbf{U}_r$, for the points chosen from Algorithm 1 can be calculated as:*

$$I(f;\mathbf{Y}_t;\mathbf{U}_r) = \frac{1}{2}\log\left(\frac{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)\det\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}\right)$$

To prove Lemma C.2.2, we need to prove the following technical lemma:

Lemma C.2.2.1. Let $\mathbf{u}\in\mathrm{R}^{n\times p}$ and $\mathbf{K}\in\mathrm{R}^{p\times p}$ is a positive semi-definite matrix and $p\geq n$. Then

$$\frac{\det\left[\mathbf{u}\left(\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}+\mathbf{I}\right)^{-1}\mathbf{u}^\top\right]}{\left[\mathbf{u}\left(\mathbf{K}+\mathbf{I}\right)^{-1}\mathbf{u}^\top\right]} = \frac{\det\left(\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}+\mathbf{I}\right)^{-1}}{\det\left(\mathbf{K}+\mathbf{I}\right)^{-1}} \tag{C.18}$$

Proof of Lemma C.2.2.1. We start the proof by gradually calculating the denominator and numerator.

Denominator

$$\begin{aligned}
&\det\left[\mathbf{u}(\mathbf{I}+\mathbf{K})^{-1}\mathbf{u}^\top\right]\\
&=\det\left[\mathbf{u}\left[(\mathbf{I}-\mathbf{K}(\mathbf{I}+\mathbf{K})^{-1}\right]\mathbf{u}^\top\right]\\
&=\det\left[\mathbf{u}\mathbf{u}^\top-\mathbf{u}\mathbf{K}(\mathbf{I}+\mathbf{K})^{-1}\mathbf{u}^\top\right]\\
&=\det\left[(\mathbf{u}\mathbf{u}^\top)\left(\mathbf{K}(\mathbf{I}+\mathbf{K})^{-1}\right)\left((\mathbf{I}+\mathbf{K})\mathbf{K}^{-1}-\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right)\right]\\
&=\det\left(\mathbf{u}\mathbf{u}^\top\right)\det\left(\mathbf{K}(\mathbf{I}+\mathbf{K})^{-1}\right)\det\left((\mathbf{I}+\mathbf{K})\mathbf{K}^{-1}-\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right)\\
&=\det\left(\mathbf{u}\mathbf{u}^\top\right)\det\left((\mathbf{I}+\mathbf{K})^{-1}\right)\det\mathbf{K}\det\left((\mathbf{I}+\mathbf{K})\mathbf{K}^{-1}-\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right)\\
&=\det\left(\mathbf{u}\mathbf{u}^\top\right)\det\left((\mathbf{I}+\mathbf{K})^{-1}\right)\det\left(\mathbf{K}(\mathbf{I}+\mathbf{K})\mathbf{K}^{-1}-\mathbf{K}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right)\\
&=\det\left(\mathbf{u}\mathbf{u}^\top\right)\det\left((\mathbf{I}+\mathbf{K})^{-1}\right)\det\left(\mathbf{I}+\mathbf{K}-\mathbf{K}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right)
\end{aligned}$$

The first equation utilizes the Woodbury matrix inversion formula while the third equation uses generalized matrix determinant lemma [1].

Numerator Let $\widetilde{\mathbf{K}}=\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}$, then we have

$$\det\left[\mathbf{u}\left(\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}+\mathbf{I}\right)^{-1}\mathbf{u}^\top\right] \tag{C.19}$$

$$=\det\left[\mathbf{u}\left(\mathbf{I}+\widetilde{\mathbf{K}}\right)^{-1}\mathbf{u}^\top\right] \tag{C.20}$$

$$=\det\left(\mathbf{u}\mathbf{u}^\top\right)\det\left((\mathbf{I}+\widetilde{\mathbf{K}})^{-1}\right)\det\left(\mathbf{I}+\widetilde{\mathbf{K}}-\widetilde{\mathbf{K}}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right), \tag{C.21}$$

[1] Suppose $\mathbf{A}$ is an invertible n-by-n matrix and $\mathbf{U},\mathbf{V}$ are n-by-m matrices, $m\leq n$. Then $\det\left(\mathbf{A}+\mathbf{U}\mathbf{W}\mathbf{V}^\top\right)=\det(\mathbf{A})\det(\mathbf{W})\det\left(\mathbf{W}^{-1}+\mathbf{V}^\top\mathbf{A}^{-1}\mathbf{U}\right)$.

where we use the result at line (C.19) and replace $\mathbf{K}$ by $\widetilde{\mathbf{K}}$. We also have:

$$\det\left((\mathbf{I}+\widetilde{\mathbf{K}})-\widetilde{\mathbf{K}}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right) \tag{C.22}$$
$$=\det\left(\mathbf{I}+\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}-\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right) \tag{C.23}$$
$$=\det\left[\mathbf{I}+\mathbf{K}\left(\mathbf{I}-\mathbf{u}^\top\mathbf{u}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}\right)-\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right] \tag{C.24}$$
$$=\det\left[\mathbf{I}+\mathbf{K}-\mathbf{K}\mathbf{u}^\top\mathbf{u}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}-\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right] \tag{C.25}$$
$$=\det\left[\mathbf{I}+\mathbf{K}-\mathbf{K}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top+\mathbf{I})^{-1}\mathbf{u}-\mathbf{K}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top+\mathbf{I})^{-1}(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right] \tag{C.26}$$
$$=\det\left[\mathbf{I}+\mathbf{K}-\mathbf{K}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top+\mathbf{I})^{-1}\left(\mathbf{I}+(\mathbf{u}\mathbf{u}^\top)^{-1}\right)\mathbf{u}\right] \tag{C.27}$$
$$=\det\left(\mathbf{I}+\mathbf{K}-\mathbf{K}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right) \tag{C.28}$$

Therefore, we have the final expression of the numerator

$$\det\left((\mathbf{I}+\widetilde{\mathbf{K}})-\widetilde{\mathbf{K}}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right) \tag{C.29}$$
$$=\det\left(\mathbf{u}\mathbf{u}^\top\right)\det\left((\mathbf{I}+\widetilde{\mathbf{K}})^{-1}\right)\det\left(\mathbf{I}+\mathbf{K}-\mathbf{K}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right) \tag{C.30}$$

Using the derived numerator and denominator, we have

$$\frac{\det\left[\mathbf{u}\left(\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}+\mathbf{I}\right)^{-1}\mathbf{u}^\top\right]}{\left[\mathbf{u}\left(\mathbf{K}+\mathbf{I}\right)^{-1}\mathbf{u}^\top\right]} \tag{C.31}$$
$$=\frac{\det(\mathbf{u}\mathbf{u}^\top)\det\left((\mathbf{I}+\widetilde{\mathbf{K}})^{-1}\right)\det\left(\mathbf{I}+\mathbf{K}-\mathbf{K}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right)}{\det(\mathbf{u}\mathbf{u}^\top)\det\left((\mathbf{I}+\mathbf{K})^{-1}\right)\det\left(\mathbf{I}+\mathbf{K}-\mathbf{K}\mathbf{u}^\top(\mathbf{u}\mathbf{u}^\top)^{-1}\mathbf{u}\right)} \tag{C.32}$$
$$=\frac{\det\left(\mathbf{I}+\widetilde{\mathbf{K}}\right)^{-1}}{\det(\mathbf{I}+\mathbf{K})^{-1}} \tag{C.33}$$
$$=\frac{\det\left(\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}+\mathbf{I}\right)^{-1}}{\det\left(\mathbf{K}+\mathbf{I}\right)^{-1}} \tag{C.34}$$

□

Corollary C.2.2.2. Let $\mathbf{u}\in\mathrm{R}^{n\times p}$ and $\mathbf{K}\in\mathrm{R}^{p\times p}$ is a positive semi-definite matrix and $p\geq n$. Then

$$\frac{\det\left[\mathbf{u}(\mathbf{K}+\mathbf{I})^{-1}\mathbf{u}^\top\right]}{\det\left[\mathbf{u}(\mathbf{K}+\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}\mathbf{u}^\top\right]}=\frac{\det(\mathbf{K}+\mathbf{I})^{-1}}{\det(\mathbf{K}+\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}} \tag{C.35}$$

Proof of Corollary C.2.2.2.

$$\frac{\det\left[\mathbf{u}(\mathbf{K}+\mathbf{I})^{-1}\mathbf{u}^\top\right]}{\det[\mathbf{u}(\mathbf{K}+\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}\mathbf{u}^\top]}$$

$$= \frac{\det\left[\mathbf{u}(\mathbf{K}+\mathbf{I})^{-1}\mathbf{u}^\top\right]}{\det[\mathbf{u}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}(\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}+\mathbf{I})^{-1}\mathbf{u}^\top]} \tag{C.36}$$

$$= \frac{\det\left[\mathbf{u}(\mathbf{K}+\mathbf{I})^{-1}\mathbf{u}^\top\right]}{\det[(\mathbf{u}\mathbf{u}^\top+\mathbf{I})^{-1}\mathbf{u}(\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}+\mathbf{I})^{-1}\mathbf{u}^\top]} \tag{C.37}$$

$$= \frac{1}{\det(\mathbf{u}\mathbf{u}^\top+\mathbf{I})^{-1}} \frac{\det\left[\mathbf{u}(\mathbf{K}+\mathbf{I})^{-1}\mathbf{u}^\top\right]}{\det[\mathbf{u}(\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}+\mathbf{I})^{-1}\mathbf{u}^\top]} \tag{C.38}$$

$$= \frac{1}{\det(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}} \frac{\det\left[(\mathbf{K}+\mathbf{I})^{-1}\right]}{\det[(\mathbf{K}(\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}+\mathbf{I})^{-1}]} \tag{C.39}$$

$$= \frac{\det(\mathbf{K}+\mathbf{I})^{-1}}{\det(\mathbf{K}+\mathbf{u}^\top\mathbf{u}+\mathbf{I})^{-1}}, \tag{C.40}$$

The second equation uses the matrix inversion identity of two non-singular matrices $\mathbf{A}$ and $\mathbf{B}$, i.e., $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$ while the fourth equation directly utilizes Lemma C.2.2.1. □

Proof of Lemma C.2.2. By definition, we have $I(f;\mathbf{Y}_t;\mathbf{U}_r) = I(f;\mathbf{Y}_t) - I(f;\mathbf{Y}_t|\mathbf{U}_r)$. We start by proof by calculating $I(f;\mathbf{Y}_t|\mathbf{U}_r)$.

By the properties of GPs, given a set of sampling points $\mathcal{D}_t \subset \mathcal{D}$, we have that $f, \mathbf{Y}_t, \mathbf{U}_r$ are jointly Gaussian:

$$\begin{pmatrix} f \\ \mathbf{Y}_t \\ \mathbf{U}_r \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \nu_t^2 \begin{bmatrix} \mathbf{K}_{uu} & \mathbf{K}_{uu} & \mathbf{K}_{ur} \\ \mathbf{K}_{uu} & \mathbf{K}_{uu}+\lambda_1\mathbf{I} & \mathbf{K}_{ur} \\ \mathbf{K}_{ru} & \mathbf{K}_{uu} & \mathbf{K}_{rr}+\lambda_2\mathbf{I} \end{bmatrix}\right) \tag{C.41}$$

Then, we have

$$\begin{aligned}
\mathrm{Cov}(f|\mathbf{U}_r) &= \nu_t^2\left[\mathbf{K}_{uu} - \mathbf{K}_{ur}(\mathbf{K}_{rr}+\lambda_2\mathbf{I})^{-1}\mathbf{K}_{ru}\right] && \text{(C.42)}\\
&= \nu_t^2\left[\mathbf{\Phi}_t\mathbf{\Phi}_t^\top - \mathbf{\Phi}_t\mathbf{\Omega}_r^\top(\mathbf{\Omega}_r\mathbf{\Omega}_r^\top+\lambda_2\mathbf{I})^{-1}\mathbf{\Omega}_r\mathbf{\Phi}_t^\top\right] && \text{(C.43)}\\
&= \nu_t^2\left[\mathbf{\Phi}_t\mathbf{\Phi}_t^\top - \mathbf{\Phi}_t\left[\mathbf{I}-\lambda_2(\mathbf{\Omega}_r\mathbf{\Omega}_r^\top+\lambda_2\mathbf{I})^{-1}\right]\mathbf{\Phi}_t^\top\right] && \text{(C.44)}\\
&= \nu_t^2\lambda_2\mathbf{\Phi}_t(\mathbf{\Omega}_r^\top\mathbf{\Omega}_r+\lambda_2\mathbf{I})^{-1}\mathbf{\Phi}_t^\top && \text{(C.45)}\\
&= \nu_t^2\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top, && \text{(C.46)}
\end{aligned}$$

and

$$\mathrm{Cov}(f|\mathbf{Y}_t;\mathbf{U}_r) = \nu_t^2\left(\mathbf{K}_{uu} - \begin{bmatrix}\mathbf{K}_{uu} & \mathbf{K}_{ur}\end{bmatrix}\begin{bmatrix}\mathbf{K}_{uu}+\lambda_1\mathbf{I} & \mathbf{K}_{ur} \\ \mathbf{K}_{ru} & \mathbf{K}_{rr}+\lambda_2\mathbf{I}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{K}_{uu} \\ \mathbf{K}_{ru}\end{bmatrix}\right) \tag{C.47}$$

$$= \nu_t^2(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top - \mathbf{\Phi}_t\boldsymbol{\xi}_t^\top\widehat{\mathbf{K}}_{\mathrm{PINN}}^{-1}\boldsymbol{\xi}_t\mathbf{\Phi}_t^\top) \tag{C.48}$$

Let $\mathbf{V} = \boldsymbol{\xi}_t^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \boldsymbol{\xi}_t$, now we need to calculate $\mathbf{V}$. We have

$$
\begin{aligned}
\mathbf{V} &= \boldsymbol{\xi}_t{}^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \boldsymbol{\xi}_t && \text{(C.49)}\\
&= \boldsymbol{\Phi}_t^\top \widetilde{\mathbf{A}} \boldsymbol{\Phi}_t + \boldsymbol{\Omega}_r^\top \widetilde{\mathbf{C}} \boldsymbol{\Phi}_t + \boldsymbol{\Phi}_t^\top \widetilde{\mathbf{B}} \boldsymbol{\Omega}_r + \boldsymbol{\Omega}_r^\top \widetilde{\mathbf{D}} \boldsymbol{\Omega}_r && \text{(C.50)}\\
&= \boldsymbol{\Phi}_t^\top (\mathbf{P}^{-1} - \mathbf{P}^{-1}\mathbf{Q}\widetilde{\mathbf{C}}) \boldsymbol{\Phi}_t + \boldsymbol{\Omega}_r^\top \widetilde{\mathbf{C}} \boldsymbol{\Phi}_t - \boldsymbol{\Phi}_t^\top \mathbf{P}^{-1}\mathbf{Q}\boldsymbol{\Omega}_r + \boldsymbol{\Omega}_r^\top \widetilde{\mathbf{D}} \boldsymbol{\Omega}_r && \text{(C.51)}\\
&= \boldsymbol{\Phi}_t^\top \mathbf{P}^{-1} \boldsymbol{\Phi}_t - \boldsymbol{\Phi}_t^\top \mathbf{P}^{-1}\mathbf{Q}\widetilde{\mathbf{C}} \boldsymbol{\Phi}_t + \boldsymbol{\Omega}_r^\top \widetilde{\mathbf{C}} \boldsymbol{\Phi}_t - \boldsymbol{\Phi}_t^\top \mathbf{P}^{-1}\mathbf{Q}\boldsymbol{\Omega}_r + \boldsymbol{\Omega}_r^\top \widetilde{\mathbf{D}} \boldsymbol{\Omega}_r && \text{(C.52)}\\
&= \boldsymbol{\Phi}_t^\top \mathbf{P}^{-1} \boldsymbol{\Phi}_t + (\underbrace{\boldsymbol{\Omega}_r - \boldsymbol{\Phi}_t^\top \mathbf{P}^{-1}\mathbf{Q}}_{U_1})(\widetilde{\mathbf{C}} \boldsymbol{\Phi}_t + \widetilde{\mathbf{D}} \boldsymbol{\Omega}_r) && \text{(C.53)}\\
&= \boldsymbol{\Phi}_t^\top \mathbf{P}^{-1} \boldsymbol{\Phi}_t + (\underbrace{\boldsymbol{\Omega}_r - \boldsymbol{\Phi}_t^\top \mathbf{P}^{-1}\mathbf{Q}}_{V_1})(\underbrace{\widetilde{\mathbf{C}} \boldsymbol{\Phi}_t + \widetilde{\mathbf{D}} \boldsymbol{\Omega}_r}_{V_2}) && \text{(C.54)}\\
&= \boldsymbol{\Phi}_t^\top (\boldsymbol{\Phi}_t \boldsymbol{\Phi}_t^\top + \lambda_1 \mathbf{I})^{-1} \boldsymbol{\Phi}_t + V_1 V_2 && \text{(C.55)}\\
&= (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + V_1 V_2 && \text{(C.56)}
\end{aligned}
$$

where

$$
\begin{bmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{uu} + \lambda_1 \mathbf{I} & \mathbf{K}_{ur} \\ \mathbf{K}_{ru} & \mathbf{K}_{rr} + \lambda_2 \mathbf{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}_t \boldsymbol{\Phi}_t^\top + \lambda_1 \mathbf{I} & \boldsymbol{\Phi}_t \boldsymbol{\Omega}_r^\top \\ \boldsymbol{\Omega}_r \boldsymbol{\Phi}_t^\top & \boldsymbol{\Omega}_r \boldsymbol{\Omega}_r^\top + \lambda_2 \mathbf{I} \end{bmatrix} \quad \text{(C.57)}
$$

$$
\text{and} \quad \begin{bmatrix} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}} \\ \widetilde{\mathbf{C}} & \widetilde{\mathbf{D}} \end{bmatrix} = \begin{bmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{bmatrix}^{-1} \quad \text{(C.58)}
$$

The second equality applied the formula of block matrix inversion [2], while the last equality used push-through identity. Next, we have

$$\mathbf{M} = (\mathbf{S} - \mathbf{R}\mathbf{P}^{-1}\mathbf{Q})^{-1} \tag{C.59}$$
$$= \left[\mathbf{\Omega}_r\mathbf{\Omega}_r^\top + \lambda_2\mathbf{I} - \mathbf{\Omega}_r\mathbf{\Phi}_t^\top(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top + \lambda_1\mathbf{I})^{-1}\mathbf{\Phi}_t\mathbf{\Omega}_r^\top)\right]^{-1} \tag{C.60}$$
$$= \left[\mathbf{\Omega}_r\mathbf{\Omega}_r^\top + \lambda_2\mathbf{I} - \mathbf{\Omega}_r(\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1}\mathbf{\Phi}_t^\top\mathbf{\Phi}_t\right]\mathbf{\Omega}_r^\top)^{-1} \tag{C.61}$$
$$= \left[\mathbf{\Omega}_r\mathbf{\Omega}_r^\top + \lambda_2\mathbf{I} - \mathbf{\Omega}_r\left[\mathbf{I} - \lambda_1(\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1}\right]\mathbf{\Omega}_r^\top)\right]^{-1} \tag{C.62}$$
$$= \left[\lambda_2\mathbf{I} + \lambda_1\mathbf{\Omega}_r(\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1}\mathbf{\Omega}_r^\top\right]^{-1} \tag{C.63}$$
$$= \lambda_1^{-1}\left[\mathbf{\Omega}_r(\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1}\mathbf{\Omega}_r^\top + \frac{\lambda_2}{\lambda_1}\mathbf{I}\right]^{-1} \tag{C.64}$$
$$V_1 = \mathbf{\Omega}_r^\top - \mathbf{\Phi}_t^\top\mathbf{P}^{-1}\mathbf{Q} \tag{C.65}$$
$$= \mathbf{\Omega}_r^\top - \mathbf{\Phi}_t^\top(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top + \lambda_1\mathbf{I})^{-1}\mathbf{\Phi}_t\mathbf{\Omega}_r^\top \tag{C.66}$$
$$= \left[\mathbf{I} - \mathbf{\Phi}_t^\top(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top + \lambda_1\mathbf{I})^{-1}\mathbf{\Phi}_t\right]\mathbf{\Omega}_r^\top \tag{C.67}$$
$$= \left[\mathbf{I} - (\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1}\mathbf{\Phi}_t^\top\mathbf{\Phi}_t\right]\mathbf{\Omega}_r^\top \tag{C.68}$$
$$= \left[\mathbf{I} - (\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1}(\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I} - \lambda_1\mathbf{I})\right]\mathbf{\Omega}_r^\top \tag{C.69}$$
$$= \lambda_1(\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1}\mathbf{\Omega}_r^\top \tag{C.70}$$
$$V_2 = \widetilde{\mathbf{C}}\mathbf{\Phi}_t + \widetilde{\mathbf{D}}\mathbf{\Omega}_r \tag{C.71}$$
$$= -\mathbf{M}\mathbf{R}\mathbf{P}^{-1} + \mathbf{M}\mathbf{\Omega}_r \tag{C.72}$$
$$= \mathbf{M}(\mathbf{\Omega}_r - \mathbf{R}\mathbf{P}^{-1}\mathbf{\Phi}_t) \tag{C.73}$$
$$= \mathbf{M}\left[\mathbf{\Omega}_r - \mathbf{\Omega}_r\mathbf{\Phi}_t^\top(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top + \lambda_1\mathbf{I})^{-1}\mathbf{\Phi}_t\right] \tag{C.74}$$
$$= \mathbf{M}\mathbf{\Omega}_r\left[\mathbf{I} - \mathbf{\Phi}_t^\top(\mathbf{\Phi}_t\mathbf{\Phi}_t^\top + \lambda_1\mathbf{I})^{-1}\mathbf{\Phi}_t\right] \tag{C.75}$$
$$= \lambda_1\mathbf{M}\mathbf{\Omega}_r(\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1} \tag{C.76}$$
$$= \left[\mathbf{\Omega}_r(\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1}\mathbf{\Omega}_r^\top + \frac{\lambda_2}{\lambda_1}\mathbf{I}\right]^{-1}\mathbf{\Omega}_r(\mathbf{\Phi}_t^\top\mathbf{\Phi}_t + \lambda_1\mathbf{I})^{-1} \tag{C.77}$$

[2] The inversion of matrix $\mathbf{K} = \begin{bmatrix}\mathbf{P} & \mathbf{Q}\\ \mathbf{R} & \mathbf{S}\end{bmatrix}$ is given as $\mathbf{K}^{-1} = \begin{bmatrix}\mathbf{P}^{-1} + \mathbf{P}^{-1}\mathbf{Q}\mathbf{M}\mathbf{R}\mathbf{P}^{-1} & \mathbf{P}^{-1}\mathbf{Q}\mathbf{M}\\ -\mathbf{M}\mathbf{R}\mathbf{P}^{-1} & \mathbf{M}\end{bmatrix}$ with $\mathbf{M} = (\mathbf{S} - \mathbf{R}\mathbf{P}^{-1}\mathbf{Q})^{-1}$.

Then we have,

$$\mathbf{V} = \boldsymbol{\xi}_t{}^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \boldsymbol{\xi}_t \tag{C.78}$$

$$= (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + V_1 V_2 \tag{C.79}$$

$$= (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \boldsymbol{\Omega}_r^\top \cdot \left[\boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \boldsymbol{\Omega}_r^\top + \frac{\lambda_2}{\lambda_1} \mathbf{I} \right]^{-1} \boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \tag{C.80}$$

$$= \mathbf{I} - \lambda_1 (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} + \lambda_1 (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \boldsymbol{\Omega}_r^\top \cdot \left[\boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \boldsymbol{\Omega}_r^\top + \frac{\lambda_2}{\lambda_1} \mathbf{I} \right]^{-1} \boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \tag{C.81}$$

$$= \mathbf{I} - \lambda_1 (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \tag{C.82}$$

$$\cdot \left[\mathbf{I} - \boldsymbol{\Omega}_r^\top \left(\boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \boldsymbol{\Omega}_r^\top + \frac{\lambda_2}{\lambda_1} \mathbf{I} \right)^{-1} \boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \right] \tag{C.83}$$

$$= \mathbf{I} - \lambda_1 (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \tag{C.84}$$

$$\cdot \left[\mathbf{I} - \left(\boldsymbol{\Omega}_r^\top \boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} + \frac{\lambda_2}{\lambda_1} \mathbf{I} \right)^{-1} \boldsymbol{\Omega}_r^\top \boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \right] \tag{C.85}$$

$$= \mathbf{I} - \lambda_1 (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \tag{C.86}$$

$$\cdot \left[\mathbf{I} - \mathbf{I} + \frac{\lambda_2}{\lambda_1} \left(\boldsymbol{\Omega}_r^\top \boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} + \frac{\lambda_2}{\lambda_1} \mathbf{I} \right)^{-1} \right] \tag{C.87}$$

$$= \mathbf{I} - \lambda_2 (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} \left(\boldsymbol{\Omega}_r^\top \boldsymbol{\Omega}_r (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I})^{-1} + \frac{\lambda_2}{\lambda_1} \mathbf{I} \right)^{-1} \tag{C.88}$$

$$= \mathbf{I} - \lambda_2 \left(\boldsymbol{\Omega}_r^\top \boldsymbol{\Omega}_r + \frac{\lambda_2}{\lambda_1} (\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_1 \mathbf{I}) \right)^{-1} \tag{C.89}$$

$$= \mathbf{I} - \lambda_2 \left(\boldsymbol{\Omega}_r^\top \boldsymbol{\Omega}_r + \frac{\lambda_2}{\lambda_1} \boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_2 \mathbf{I} \right)^{-1} \tag{C.90}$$

In conclusion, we have

$$\boldsymbol{\xi}_t^\top \widehat{\mathbf{K}}_{\text{PINN}}^{-1} \boldsymbol{\xi}_t = \mathbf{I} - \lambda_2 \left(\boldsymbol{\Omega}_r^\top \boldsymbol{\Omega}_r + \frac{\lambda_2}{\lambda_1} \boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t + \lambda_2 \mathbf{I} \right)^{-1} \tag{C.91}$$

$$= \mathbf{I} - \left(\frac{\boldsymbol{\Omega}_r^\top \boldsymbol{\Omega}_r}{\lambda_2} + \frac{\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t}{\lambda_1} + \mathbf{I} \right)^{-1} \tag{C.92}$$

Replace Eqn. C.91 to the expression of $\text{Cov}(f_t | \mathbf{Y}_t; \mathbf{U}_r)$ in Eqn. C.47, we have

$$\text{Cov}(f | \mathbf{Y}_t; \mathbf{U}_r) = \nu_t^2 \boldsymbol{\Phi}_t \left(\frac{\boldsymbol{\Omega}_r^\top \boldsymbol{\Omega}_r}{\lambda_2} + \frac{\boldsymbol{\Phi}_t^\top \boldsymbol{\Phi}_t}{\lambda_1} + \mathbf{I} \right)^{-1} \boldsymbol{\Phi}_t^\top \tag{C.93}$$

Combining the expression of $\mathrm{Cov}(f|\mathbf{U}_r)$ in Eqn. C.46 with Eqn. C.93, with H is the entropy function, we have

$$
\begin{aligned}
& I(f;\mathbf{Y}_t|\mathbf{U}_r) \\
&= H(f|\mathbf{U}_r) - H(f|\mathbf{Y}_t;\mathbf{U}_r) && \text{(C.94)} \\
&= \frac{1}{2}\log\det(\mathrm{Cov}(f|\mathbf{U}_r)) - \frac{1}{2}\log\det(\mathrm{Cov}(f|\mathbf{Y}_t;\mathbf{U}_r)) && \text{(C.95)} \\
&= \frac{1}{2}\log\det\left(\nu_t^2\boldsymbol{\Phi}_t\left(\frac{\boldsymbol{\Omega}_r^\top\boldsymbol{\Omega}_r}{\lambda_2}+\mathbf{I}\right)^{-1}\boldsymbol{\Phi}_t^\top\right) \\
&\quad - \frac{1}{2}\log\det\left(\nu_t^2\boldsymbol{\Phi}_t\left(\frac{\boldsymbol{\Omega}_r^\top\boldsymbol{\Omega}_r}{\lambda_2}+\frac{\boldsymbol{\Phi}_t^\top\boldsymbol{\Phi}_t}{\lambda_1}+\mathbf{I}\right)^{-1}\boldsymbol{\Phi}_t^\top\right) && \text{(C.96)} \\
&= \frac{1}{2}\log\frac{\det\left(\boldsymbol{\Phi}_t\left(\frac{\boldsymbol{\Omega}_r^\top\boldsymbol{\Omega}_r}{\lambda_2}+\mathbf{I}\right)^{-1}\boldsymbol{\Phi}_t^\top\right)}{\det\left(\boldsymbol{\Phi}_t\left(\frac{\boldsymbol{\Omega}_r^\top\boldsymbol{\Omega}_r}{\lambda_2}+\frac{\boldsymbol{\Phi}_t^\top\boldsymbol{\Phi}_t}{\lambda_1}+\mathbf{I}\right)^{-1}\boldsymbol{\Phi}_t^\top\right)} && \text{(C.97)}
\end{aligned}
$$

Then we have

$$
\begin{aligned}
& I(f;\mathbf{Y}_t;\mathbf{U}_r) \\
&= I(f;\mathbf{Y}_t) - I(f;\mathbf{Y}_t|\mathbf{U}_r) && \text{(C.98)} \\
&= \frac{1}{2}\log\det\left(\frac{\mathbf{\Phi}_t\mathbf{\Phi}_t^\top}{\lambda_1}+\mathbf{I}\right) - \frac{1}{2}\log\frac{\det\left(\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right)}{\det\left(\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right)} && \text{(C.99)} \\
&= \frac{1}{2}\log\frac{\det\left(\frac{\mathbf{\Phi}_t\mathbf{\Phi}_t^\top}{\lambda_1}+\mathbf{I}\right)\det\left(\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right)}{\det\left(\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right)} && \text{(C.100)} \\
&= \frac{1}{2}\log\frac{\det\left[\left(\frac{\mathbf{\Phi}_t\mathbf{\Phi}_t^\top}{\lambda_1}+\mathbf{I}\right)\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right]}{\det\left[\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right]} && \text{(C.101)} \\
&= \frac{1}{2}\log\frac{\det\left[\mathbf{\Phi}_t\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right]}{\det\left[\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right]} && \text{(C.102)} \\
&= \frac{1}{2}\log\frac{\det\left[\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)^{-1}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right]}{\det\left[\mathbf{\Phi}_t\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)^{-1}\mathbf{\Phi}_t^\top\right]} && \text{(C.103)} \\
&= \frac{1}{2}\log\frac{\det\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)^{-1}+\mathbf{I}\right)^{-1}}{\det\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)^{-1}} && \text{(C.104)} \\
&= \frac{1}{2}\log\frac{\det\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}{\det\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)^{-1}+\mathbf{I}\right)} && \text{(C.105)} \\
&= \frac{1}{2}\log\left(\frac{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\mathbf{I}\right)\det\left(\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}{\det\left(\frac{\mathbf{\Phi}_t^\top\mathbf{\Phi}_t}{\lambda_1}+\frac{\mathbf{\Omega}_r^\top\mathbf{\Omega}_r}{\lambda_2}+\mathbf{I}\right)}\right), && \text{(C.106)}
\end{aligned}
$$

where Eqn C.104 is resulted from technical Lemma C.2.2.1. □

Bibliography

Agrawal, Shipra and Navin Goyal (2012). "Analysis of thompson sampling for the multi-armed bandit problem". In: *Conference on learning theory*. JMLR Workshop and Conference Proceedings, pp. 39–1.

Agrawal, Shipra et al. (2017). "Thompson sampling for the mnl-bandit". In: *Conference on learning theory*. PMLR, pp. 76–78.

Allen-Zhu, Zeyuan, Yuanzhi Li, and Zhao Song (2019). "A convergence theory for deep learning via over-parameterization". In: *International Conference on Machine Learning*. PMLR, pp. 242–252.

Antonini, Rita Giuliano, Yuriy Kozachenko, and Andrei Volodin (2008). "Convergence of series of dependent φ-subGaussian random variables". In: *Journal of mathematical analysis and applications* 338.2, pp. 1188–1203.

Ariafar, Setareh et al. (2019). "ADMMBO: Bayesian optimization with unknown constraints using ADMM". In: *Journal of Machine Learning Research* 20.123, pp. 1–26.

Arora, Sanjeev et al. (2019). "Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks". In: *International Conference on Machine Learning*. PMLR, pp. 322–332.

Auer, P (2002). *Finite-time Analysis of the Multiarmed Bandit Problem*.

Bergstra, James and Yoshua Bengio (2012). "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2.

Berkenkamp, Felix, Andreas Krause, and Angela P Schoellig (2023). "Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics". In: *Machine Learning* 112.10, pp. 3713–3747.

Bubeck, Sébastien, Nicolo Cesa-Bianchi, et al. (2012). "Regret analysis of stochastic and nonstochastic multi-armed bandit problems". In: *Foundations and Trends® in Machine Learning* 5.1, pp. 1–122.

Cai, Shengze et al. (2020). "Heat transfer prediction with unknown thermal boundary conditions using physics-informed neural networks". In: *Fluids Engineering Division Summer Meeting*.

Cao, Yuan and Quanquan Gu (2019). "Generalization bounds of stochastic gradient descent for wide and deep neural networks". In: *Advances in neural information processing systems* 32.

— (2020). "Generalization error bounds of gradient descent for learning over-parameterized deep relu networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34, pp. 3349–3356.

Chen, Yifan et al. (2021). "Solving and learning nonlinear PDEs with Gaussian processes". In: *Journal of Computational Physics* 447, p. 110668.

Chowdhury, Sayak Ray and Aditya Gopalan (2017). "On kernelized multi-armed bandits". In: *International Conference on Machine Learning*. PMLR, pp. 844–853.

Cover, Thomas M (1999). *Elements of information theory*. John Wiley & Sons.

Du, Simon S et al. (2018). "Gradient Descent Provably Optimizes Over-parameterized Neural Networks". In: *International Conference on Learning Representations*.

Frazier, Peter I (2018). "A tutorial on Bayesian optimization". In: *arXiv preprint arXiv:1807.02811*.

Frazier, Peter I, Warren B Powell, and Savas Dayanik (2008). "A knowledge-gradient policy for sequential information collection". In: *SIAM Journal on Control and Optimization* 47.5, pp. 2410–2439.

Gardner, Jacob R et al. (2014). "Bayesian optimization with inequality constraints." In: *ICML*. Vol. 2014, pp. 937–945.

Gelbart, Michael A, Jasper Snoek, and Ryan P Adams (2014). "Bayesian optimization with unknown constraints". In: *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pp. 250–259.

Gonzalez, Javier et al. (2015). "Bayesian optimization for synthetic gene design". In: *arXiv preprint arXiv:1505.01627*.

Gramacy, Robert B et al. (2016). "Modeling an augmented Lagrangian for blackbox constrained optimization". In: *Technometrics* 58.1, pp. 1–11.

Greenhill, Stewart et al. (2020). "Bayesian optimization for adaptive experimental design: A review". In: *IEEE access* 8, pp. 13937–13948.

Hansen, Nikolaus and Andreas Ostermeier (2001). "Completely derandomized self-adaptation in evolution strategies". In: *Evolutionary computation* 9.2, pp. 159–195.

Hansen, Nikolaus et al. (2021). "COCO: A platform for comparing continuous optimizers in a black-box setting". In: *Optimization Methods and Software* 36.1, pp. 114–144.

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

He, Zecheng, Tianwei Zhang, and Ruby B Lee (2018). "Verideep: Verifying integrity of deep neural networks through sensitive-sample fingerprinting". In: *arXiv preprint arXiv:1808.03277*.

Hennig, Philipp and Christian J Schuler (2012). "Entropy Search for Information-Efficient Global Optimization." In: *Journal of Machine Learning Research* 13.6.

Hernández-Lobato, José Miguel, Matthew W Hoffman, and Zoubin Ghahramani (2014). "Predictive entropy search for efficient global optimization of black-box functions". In: *Advances in neural information processing systems* 27.

Hernández-Lobato, José Miguel et al. (2015). "Predictive entropy search for bayesian optimization with unknown constraints". In: *International conference on machine learning*. PMLR, pp. 1699–1707.

Hestenes, Magnus Rudolph, Eduard Stiefel, et al. (1952). *Methods of conjugate gradients for solving linear systems*. Vol. 49. 1. NBS Washington, DC.

Hoffman, Matthew W, Bobak Shahriari, and Nando de Freitas (2013). "Exploiting correlation and budget constraints in bayesian multi-armed bandit optimization". In: *arXiv preprint arXiv:1303.6746*.

Hutter, Frank, Holger H Hoos, and Kevin Leyton-Brown (2011). "Sequential model-based optimization for general algorithm configuration". In: *International conference on learning and intelligent optimization*. Springer, pp. 507–523.

Jacot, Arthur, Franck Gabriel, and Clément Hongler (2018). "Neural tangent kernel: Convergence and generalization in neural networks". In: *Advances in neural information processing systems* 31.

Jidling, Carl et al. (2017). "Linearly constrained Gaussian processes". In: *Advances in Neural Information Processing Systems* 30.

Jones, Donald R, Matthias Schonlau, and William J Welch (1998). "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13.4, pp. 455–492.

Ju, Shenghong et al. (2017). "Designing nanostructures for phonon transport via Bayesian optimization". In: *Physical Review X* 7.2, p. 021024.

Kassraie, Parnian and Andreas Krause (2022). "Neural contextual bandits without regret". In: *Artificial Intelligence and Statistics*. PMLR, pp. 240–278.

Kikuchi, Shun et al. (2018). "Bayesian optimization for efficient determination of metal oxide grain boundary structures". In: *Physica B: Condensed Matter* 532, pp. 24–28.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kiritchenko, Svetlana, Stan Matwin, A Fazel Famili, et al. (2005). "Functional annotation of genes using hierarchical text categorization". In: *Proc. of the ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*.

Kirkpatrick, Scott, C Daniel Gelatt Jr, and Mario P Vecchi (1983). "Optimization by simulated annealing". In: *science* 220.4598, pp. 671–680.

Kumar, Abhishek et al. (2020). "A test-suite of non-convex constrained optimization problems from the real-world and some baseline results". In: *Swarm and Evolutionary Computation* 56, p. 100693.

Kushner, Harold J (1964). "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise". In: *Journal of Fluids Engineering* 86.1, pp. 97–106.

Lattimore, Tor and Csaba Szepesvári (2020). *Bandit algorithms*. Cambridge University Press.

LeCun, Yann and Corinna Cortes (2010). "MNIST handwritten digit database". In: URL: http://yann.lecun.com/exdb/mnist/.

Letham, Benjamin et al. (2019). "Constrained Bayesian Optimization with Noisy Experiments". In: *Bayesian Analysis* 14.2, pp. 495–519.

Li, g et al. (2017). "Rapid Bayesian optimisation for synthesis of short polymer fiber materials". In: *Scientific reports* 7.1, pp. 1–10.

Li, Lihong et al. (2010). "A contextual-bandit approach to personalized news article recommendation". In: *Proceedings of the 19th international conference on World wide web*, pp. 661–670.

Lu, Congwen and Joel A Paulson (2022). "No-regret Bayesian optimization with unknown equality and inequality constraints using exact penalty functions". In: *IFAC-PapersOnLine* 55.7, pp. 895–902.

Mockus, Jonas, Vytautas Tiesis, and Antanas Zilinskas (1978). "The application of Bayesian methods for seeking the extremum". In: *Towards global optimization* 2.117-129, p. 2.

Nelder, John A and Roger Mead (1965). "A simplex method for function minimization". In: *The computer journal* 7.4, pp. 308–313.

Nguyen, Quoc Phong et al. (2023). "Optimistic Bayesian Optimization with Unknown Constraints". In: *The Twelfth International Conference on Learning Representations*.

Nour, Majid, Zafer Cömert, and Kemal Polat (2020). "A novel medical diagnosis model for COVID-19 infection detection based on deep features and Bayesian optimization". In: *Applied Soft Computing* 97, p. 106580.

Osborne, Michael A, Roman Garnett, and Stephen J Roberts (2009). "Gaussian processes for global optimization". In: *3rd international conference on learning and intelligent optimization (LION3)*. Citeseer, pp. 1–15.

Paria, Biswajit et al. (2022). "Be Greedy–a Simple Algorithm for Blackbox Optimization using Neural Networks". In: *ICML2022 Workshop on Adaptive Experimental Design and Active Learning in the Real World*.

Phan-Trong, Dat, Hung Tran-The, and Sunil Gupta (2023). "NeuralBO: A black-box optimization algorithm using deep neural networks". In: *Neurocomputing* 559, p. 126776.

Picheny, Victor et al. (2016). "Bayesian optimization under mixed constraints with a slack-variable augmented Lagrangian". In: *Advances in neural information processing systems* 29.

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational physics* 378, pp. 686–707.

Raissi, Maziar, Paris Perdikaris, and George Em Karniadakis (2017). "Machine learning of linear differential equations using Gaussian processes". In: *Journal of Computational Physics* 348, pp. 683–693.

Rasmussen, Carl Edward, Christopher KI Williams, et al. (2006). *Gaussian processes for machine learning*. Vol. 1. Springer.

Robbins, Herbert (1952). "Some aspects of the sequential design of experiments". In.

Russo, Daniel J et al. (2018). "A tutorial on thompson sampling". In: *Foundations and Trends® in Machine Learning* 11.1, pp. 1–96.

Schiassi, Enrico et al. (2021). "Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations". In: *Neurocomputing* 457, pp. 334–356.

Schonlau, Matthias, William J Welch, and Donald R Jones (1998). "Global versus local search in constrained optimization of computer models". In: *Lecture notes-monograph series*, pp. 11–25.

Shahriari, Bobak et al. (2015). "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1, pp. 148–175.

Snelson, Edward and Zoubin Ghahramani (2007). "Local and global sparse Gaussian process approximations". In: *Artificial Intelligence and Statistics*. PMLR, pp. 524–531.

Snoek, Jasper, Hugo Larochelle, and Ryan P Adams (2012). "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems* 25.

Snoek, Jasper et al. (2015). "Scalable bayesian optimization using deep neural networks". In: *International conference on machine learning*. PMLR, pp. 2171–2180.

Springenberg, Jost Tobias et al. (2016). "Bayesian optimization with robust Bayesian neural networks". In: *NeurIPS* 29.

Srinivas, Niranjan et al. (2009). "Gaussian process optimization in the bandit setting: No regret and experimental design". In: *arXiv preprint arXiv:0912.3995*.

Streltsov, Simon and Pirooz Vakili (1999). "A non-myopic utility function for statistical global optimization algorithms". In: *Journal of Global Optimization* 14, pp. 283–298.

Swersky, Kevin, Jasper Snoek, and Ryan P Adams (2013). "Multi-task bayesian optimization". In: *Advances in neural information processing systems* 26.

Swiler, Laura P et al. (2020). "A survey of constrained Gaussian process regression: Approaches and implementation challenges". In: *Journal of Machine Learning for Modeling and Computing* 1.2.

Takeno, Shion et al. (2022). "Sequential and parallel constrained max-value entropy search via information lower bound". In: *International Conference on Machine Learning*. PMLR, pp. 20960–20986.

Thompson, William R (1933). "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika* 25.3-4, pp. 285–294.

Tieleman, Tijmen (2012). "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2, p. 26.

Tran, Dustin, Rajesh Ranganath, and David M Blei (2016). "The variational Gaussian process". In: *4th International Conference on Learning Representations, ICLR 2016*.

Tran-The, Hung et al. (2022). "Regret bounds for expected improvement algorithms in Gaussian process bandit optimization". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 8715–8737.

Turgeon, Martine, Cindy Lustig, and Warren H Meck (2016). "Cognitive aging and time perception: Roles of Bayesian optimization and degeneracy". In: *Frontiers in aging neuroscience* 8, p. 102.

Vakili, Sattar, Kia Khezeli, and Victor Picheny (2021). "On information gain and regret bounds in gaussian process bandits". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 82–90.

Vakili, Sattar et al. (2021). "Optimal order simple regret for Gaussian process bandits". In: *Advances in Neural Information Processing Systems* 34, pp. 21202–21215.

Wang, Chuwei et al. (2022). "Is L_2 Physics Informed Loss Always Suitable for Training Physics Informed Neural Network?" In: *Advances in Neural Information Processing Systems* 35, pp. 8278–8290.

Wang, Sifan, Xinling Yu, and Paris Perdikaris (2022). "When and why PINNs fail to train: A neural tangent kernel perspective". In: *Journal of Computational Physics* 449, p. 110768.

Wang, Zi and Stefanie Jegelka (2017). "Max-value entropy search for efficient Bayesian optimization". In: *International Conference on Machine Learning*. PMLR, pp. 3627–3635.

Wu, Jian and Peter Frazier (2016). "The parallel knowledge gradient method for batch Bayesian optimization". In: *Advances in neural information processing systems* 29.

Xu, Jiaming and Hanjing Zhu (2024). "Overparametrized Multi-layer Neural Networks: Uniform Concentration of Neural Tangent Kernel and Convergence of Stochastic Gradient Descent". In: *Journal of Machine Learning Research* 25.94, pp. 1–83.

Xu, Pan et al. (2020). "Neural contextual bandits with deep representation and shallow exploration". In: *preprint arXiv:2012.01780*.

Xu, Wenjie et al. (2023). "Constrained efficient global optimization of expensive black-box functions". In: *International Conference on Machine Learning*. PMLR, pp. 38485–38498.

Yang, Liu, Xuhui Meng, and George Em Karniadakis (2021). "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data". In: *Journal of Computational Physics* 425, p. 109913.

Zhang, Weitong et al. (2021). "Neural Thompson Sampling". In: *International Conference on Learning Representation (ICLR)*.

Zhou, Dongruo, Lihong Li, and Quanquan Gu (2020). "Neural contextual bandits with ucb-based exploration". In: *International Conference on Machine Learning*. PMLR, pp. 11492–11502.

Zhou, Xingyu and Bo Ji (2022). "On kernelized multi-armed bandits with constraints". In: *Advances in neural information processing systems* 35, pp. 14–26.