University of Padova

Department of Mathematics Tullio-Levi Civita

Master Thesis in Data Science

Bi-level Optimization with Penalty Method for Binary Hyperparameter Tuning in Data Hyper-cleaning

Supervisor
prof. Francesco Rinaldi
University of Padova

Master Candidate
Dandan Zhao
2041513

Academic Year

2023-2024

I would like to express my heartfelt thanks to all the people who have helped and supported me.

Firstly, my sincere and deepest gratitude goes to my advisor, Professor Francesco Rinaldi, whose guidance and support have been invaluable throughout this journey. Despite his very busy schedule, he always took the time to carefully read my work and provide detailed feedback. Furthermore, it has been my honor to benefit from his personality. I still remember his words in the final session, where he reminded us that regardless of whether we become data scientists, we should remain humble, contribute meaningfully, and treat everyone with respect. It has been a great privilege and joy to study under his guidance, and his wisdom and dedication will leave a lasting influence on my life. My gratitude to him knows no bounds.

I also want to extend my deepest thanks to my family and my partner. When I made the decision to leave behind a 7-year career and start a study journey in a distant and unfamiliar country, they were concerned about the challenges I might face. Yet, they still supported me wholeheartedly. It was their love that gave me the encouragement and courage to successfully complete this important journey in my life.

Thanks to everyone who supported me throughout.

Abstract

Bilevel optimization presents significant challenges, particularly in large-scale problems, and binary hyperparameter optimization introduces additional complexities due to the combinatorial nature of the problem. For example, the binary constraint makes the objective function non-differentiable with respect to these binary variables, which means gradient-based methods cannot be applied directly. Additionally, the exponential increase in the number of potential configurations makes the problem computationally expensive to solve. Traditional relaxation and rounding methods often lead to inconsistent results. In this thesis, we propose a novel algorithm that combines the Relax and Penalize framework with the Zeroth-Order Frank-Wolfe algorithm, effectively handling binary variables with theoretical guarantees while maintaining computational efficiency. Our approach combines zeroth-order hypergradient estimates with a penalty approach for handling the combinatorial part, thus reducing the heavy computation typically required in gradient-based bilevel optimization methods.

In numerical experiments, we evaluate the effectiveness of our approach through tests on a data hyper-cleaning task using synthetic and MNIST datasets under varying noise conditions. In particular, we study how the maximum number of iterations in the lower-level problem affects the algorithm's efficiency and effectiveness. Looking ahead, future work will focus on exploring the impact of additional algorithmic parameters, such as the penalty parameter, decay factors, and the number of random directions used for gradient estimation. These investigations will provide further insights into how our method can be implemented more efficiently in practical applications.

Contents

Listing of figures

Listing of acronyms

BLO	Bi-Level Optimization
LL	Lower Level Subproblem
UL	Upper Level Subproblem
HO	Hyperparameter Optimization
ZOFW	Zeroth-Order Frank-Wolfe
EGBMs	Explicit Gradient-Based Methods
IGBMs	Implicit Gradient-Based Methods
LLC	Lower Level Convexity
LLS	Lower-Level Singleton
BDA	Bilevel Descent Aggregation
TRHG	Truncated Reverse Hyper-Gradient
FHG	Forward Hyper-Gradient
RHG	Reverse Hyper-Gradient
IFH	Implicit Function Theorem
CG	Conjugate Gradient
HP	Hyperparameter
LR	learning rate
SGD	Stochastic Gradient Descent
SMBO	Sequential Model-Based Optimization
KWSA	Kiefer Wolfowitz Stochastic Approximation
RDSA	Random Directions Stochastic Approximation
i-RDSA	improved Random Directions Stochastic Approximation
MNIST	Modified National Institute of Standards and Technology database

1 Introduction

1.1 Research Problem and Objectives

Bi-level optimization(BLO) refers to an optimization problem where one problem is nested within another as a constraint [2, 3]. The inner (or nested) and outer optimization tasks are often respectively referred to as the Lower-Level (LL) and Upper-Level (UL) subproblems. BLO has attracted significant research attention due to the nested problem structure in many deep learning problems, such as hyperparameter optimization [4, 5] , meta-learning [4, 6], continual learning [7], reinforcement learning [8, 9], and neural network architecture search [10]. Despite the different motivations and mechanisms, all these problems contain a series of closely related subproblems and have a natural hierarchical optimization structure.

However, bilevel optimization is much more challenging than classical optimization problems due to the complicated intertwining between the outer problem and the inner problem, especially in large-scale and high dimensional practical machine learning applications. Nowadays, as modern machine learning systems incorporate an increasing number of both parameters and hyperparameters, aiming to enhance overall performance and flexibility, placing increasingly high demands on computational resources, storage, and time. These challenges have motivated increasing efforts to develop efficient and robust algorithms that can handle the resource demands of real-world machine learning applications.

A key focus in machine learning optimization is hyperparameter optimization (HO). Unlike model parameters, hyperparameters guide the learning process however cannot be directly learned from the training data [11]. They can affect various aspects of a learning model, such as network architecture (e.g., number of layers and nodes), optimization hyperparameters (e.g., learning rate and momentum), and regularization (e.g., weight decay and dropout), which are crucial in the development of efficient and robust machine learning models, that is why hyperparameter optimization has consistently attracted significant research interest. Among the diverse approaches to hyperparameter optimization, such as the populated grid search, random search, the main challenge is hard to handle high-dimensional parameters. So the focus has shifted toward gradient-based bilevel approaches, which are considered one of the most promising methods for addressing these challenges.

There are several interesting applications of hyperparameter optimization, such as filtering a clean subset from noisy data, prune large-scale deep learning models, identify group-sparsity structures in regression problems, and learn the discrete structure of graph neural networks. In addition to being high-dimensional, these hyperparameters are also binary in nature, which makes their optimization significantly more complex. As shown in these applications, the binary hyperparameters are critical for improving model robustness and efficiency, making them valuable to be researched.

Given the increasing complexity and resource demands of modern machine learning systems, the need for efficient and robust optimization methods has become more critical than ever. Addressing these challenges is particularly important in the context of hyperparameter optimization, where resource constraints often play a significant role in determining model performance. Therefore, this study aims to develop a resource-efficient and robust bilevel optimization algorithm specifically designed for binary hyperparameter optimization, with a focus on data usage efficiency through hyper-cleaning.

1.2 Significance of the Study

As machine learning models become increasingly complex, the number of parameters involved in these models grows correspondingly, which significantly impacts the efficiency of optimization processes, particularly in bilevel optimization scenarios. In different BLO applications, the interplay between the lower-level and upper-level optimization tasks creates additional computational challenges. The complexity introduced by the large number of parameters and the nested structure of BLO increases the demand for computational resources, making the optimization process resource-intensive and time-consuming. Traditional gradient-based methods for solving BLO, which are mainstream methods, currently often require the computation of high-dimensional Jacobian, Hessians or the inverse, which further exacerbates the computational burden, particularly in large-scale settings. This creates a pressing need for more efficient optimization strategies that can reduce computational overhead while maintaining model accuracy and robustness.

To address this challenge, this study proposes a zeroth-order optimization method leveraging the Zeroth-Order Frank-Wolfe (ZOFW) algorithm [12], which inherently avoids the costly computation required in the traditional gradient methods, reducing the computational complexity significantly. By adopting this approach, we aim to develop algorithms that are not only computationally efficient but also scalable to large-scale settings, thereby making them applicable to real-world scenarios where computational resources are a critical limiting factor.

From the application perspective, the study focuses on the data hyper-cleaning task. This is motivated by two key factors. Firstly, Data is fundamental to machine learning: without data, learning is impossible. The quality and characteristics of the data used in training can dramatically influence the outcome of the model. The choice of data, whether clean, noisy, or representative of the real-world scenario, plays a crucial role in determining the performance, reliability, and generalization of the learned model. Secondly, in today's era of data explosion, while we are not lacking in data, the challenge lies in using it efficiently. Relying solely on large volumes of data, especially when much of it contains noise, can lead to wasted computational resources and unreliable learning outcomes. Noisy data can mislead the model, reducing its robustness and overall trustworthiness. Thus, efficiently selecting a smaller but cleaner subset of data is essential for robust learning, avoiding the pitfalls of over-relying on noisy datasets.

By integrating these two aspects—computational efficiency and improved data usage efficiency through a binary hyperparameter optimization problem —this study contributes to the broader goal of developing more resource-efficient machine learning models. These models are particularly valuable in the era of big data, where processing large volumes of information efficiently while maintaining high performance is critical.

1.3 Structure of the Thesis

In our study, we focus on optimizing the binary hyperparameter in the data hyper-cleaning task. The binary hyperparameter optimization problem is formulated as a bilevel optimization problem, for which we first relax the binary variables to be continuous through a penalty term [13], then use a zeroth order gradient approximation of the upper level variable w.r.t. the upper level objective to account for the interaction between the lower-level and upper-level subproblems. After obtaining a solution for the relaxed problem, we apply an iterative strategy to gradually drive the continuous upper-level variable closer to binary values (0 or 1) via the introduced penalty term.

One of the important features of our proposed method is that it is an efficient zeroth order algorithm [12] for bi-level optimization. Moreover, by iteratively solving a series of relaxed continuous and penalized problems, under suitable assumptions, the Relax and Penalize approach [13] we adopt guarantees to provide mixed-binary local solutions compared to the typical relaxation and rounding which lack theoretical guarantees.

The remainder of this thesis is structured as follows:

- **Chapter 2 Literature Review** This chapter reviews the development and evolution of bilevel optimization (BLO) techniques, covering both early classical methods and more scalable modern gradient-based methods. The chapter also explores hyperparameter optimization (HO), emphasizing its importance in machine learning and how it can be modeled as a BLO task. Finally, the chapter introduces the concept of data hyper-cleaning as an application of BLO to improve model robustness in the presence of noisy data.

- **Chapter 3 Methodological Foundations** This chapter introduces the proposed method, which integrates the Relax and Penalize approach with the Zeroth-Order Frank-Wolfe (ZOFW) algorithm to address the challenges of bilevel optimization involving binary variables. After an in-depth exploration of the theoretical foundations of both these two approaches and their suitability for handling the complexities discussed in Chapter 2, the chapter presents the proposed algorithm. It outlines the algorithm's structure, detailing how the two methods are combined and utilized effectively.

- **Chapter 4 Experiments and Analysis** This chapter presents the experimental setup, results, and analysis of the proposed bilevel optimization method applied to data hyper-cleaning. We frame the problem as a binary bilevel optimization task and evaluate the algorithm on both synthetic and MNIST [14] datasets under varying noise conditions. Key evaluation metrics include test loss, time to reach target loss, and upper-level objective function convergence. The results demonstrate that increasing inner-loop iterations (T) improves performance, though with diminishing returns at higher values. The experiments also highlight the effectiveness of data filtering, particularly in noisy environments, where models trained on filtered data outperform those trained on noisy data.

The chapter concludes by analyzing the trade-offs between computational cost and convergence speed.

- **Chapter 5 Conclusion and Future Work** The thesis concludes with a summary of the key findings, discussing the contributions of the research and potential future directions. The chapter discusses how the proposed algorithm can be refined and applied to other areas of machine learning.

2
Literature Review

2.1 Bi-level Optimization

2.1.1 Early Developments and Classical Methods

Bilevel optimization can be traced to two domains: one is from game theory where the leader and the follower compete on quantity in the Stackelberg game [15]; another one is from mathematical programming where the inner level problem serves as a constraint on the outer level problem [2]. The hierarchical structure of BLO introduces inherent complexity compared to traditional optimization problems. Early studies primarily focused on developing numerical methods to solve these challenges. The classical methods include:

- Extreme Point Methods [3] Rely on the identification of extreme points in the feasible region of the lower-level problem, mainly applied to linear BLOs problems with a well-defined polyhedral structure. but struggle with non-linear or non-convex problems.

- Branch-and-Bound Methods Used for solving mixed-integer BLOs [16] by exploring the solution space systematically by branching on decision variables and bounding the solution quality, but performance deteriorates significantly with increasing problem dimensions.

- Descent Methods Involve iterative improvements via moving in the direction of the steepest descent [17], but their applicability is limited by the requirement for specific smoothness properties of objective functions.

- Penalty Function Methods Convert BLO into a single-level problem by adding penalty terms [18, 19]. However, the challenge lies in tuning the penalty parameters and the method's success often depends on the problem's specific structure.
- Trust-Region Methods Approximate the BLO problem within a local "trust region" around the current solution [20] but solving subproblems within each region are computationally costly, especially for large-scale problems.

While these methods have laid a strong foundation in bilevel optimization, their scalability to large-scale, high-dimensional problems remains challenging, prompting the development of more practical approaches for modern machine learning tasks, which increasingly involve large numbers of parameters and vast datasets.

2.1.2 Gradient-based Methods: EGBMs and IGBMs

Compared with classical methods [16], which require strict mathematical properties or cannot scale well to large datasets, efficient gradient-based methods provide a promising solution. These methods have been widely adopted in machine learning research. These methods can be broadly categorized into Explicit Gradient-Based Methods (EGBMs) [4, 10, 21] and Implicit Gradient-Based Methods (IGBMs) [6, 22, 23], according to divergent ideas of calculating the gradient needed for implementing gradient descent.

- **Explicit Gradient-Based Methods (EGBMs)** EGBMs involve calculating gradients explicitly by differentiating through the entire optimization process. One common approach is unrolled differentiation, where the lower-level problem is iteratively solved, and the resulting gradients are used to update the upper-level variables. Under the Lower-Level Singleton(LLS) and Lower Level Convexity (LLC) the works in [21, 24] first calculate gradient representations of the LL objective and then perform either reverse or forward gradient computations (termed as Reverse Hyper-Gradient (RHG) and Forward Hyper-Gradient (FHG)) for the UL sub-problem. In order to address the LLS restriction, under the LLC, [25] proposes Bilevel Descent Aggregation (BDA) which characterizes an aggregation of both the LL and the UL descent information as a new EGBM. However, since the EGBMs method requires calculating AD for the entire trajectory of the dynamic iteration of LL, the computation load is heavy to calculate the gradient with reasonable accuracy. To reduce the computation burden, Truncated Reverse Hyper-Gradient (TRHG) [26] was proposed, which truncates the gradient trajectory. Nevertheless, TRHG's efficiency is sensitive to the truncation length, and finding an optimal truncation length remains challenging.

- **Implicit Gradient-Based Methods (IGBMs)** IGBMs or implicit differentiation, on the other hand, avoid explicit differentiation by using the Implicit Function Theorem(IFH) to obtain gradients. In situations where the lower-level problem is strongly convex and satisfies LLS conditions, the gradient of the UL objective can be calculated by implicit differential equations, however with an additional condition which requires the Hessian matrix to be invertible. Unlike EGBMs that rely on computing first-order gradients along the entire trajectory of the lower-level problem, IGBMs only use the first-order condition once. This decouples the computational burden from the lower-level solution trajectory. However, a major drawback of IGBMs is the need to repeatedly compute the inverse of the Hessian matrix, which remains computationally expensive. To alleviate this, techniques such as the Conjugate Gradient (CG) method and Neumann series approximations are often used to estimate the Hessian inverse. However the overall process still requires substantial computational resources because of computing Hessian-vector products.

As discussed above, both EGBMs and IGBMs encounter significant bottlenecks. In theory, both rely on strong assumptions such as the LLS and LLC conditions to ensure convergence and stability. These conditions are often not met in practical applications, where the LL problem in real-world learning tasks may be non-convex or contain multiple local minima, leading to serious convergence issues or failure to find a suitable solution. In terms of computational challenges, EGBMs, especially those using unrolled differentiation, require space and time complexity, as gradients must be computed along the entire trajectory of the LL optimization. On the other hand, IGBMs reduce the burden of tracking the LL trajectory but come with their own computational cost for Hessian inversion or approximation.

These theoretical and computational challenges significantly limit the practical application of gradient-based methods, particularly in large-scale bilevel optimization problems where models involve complex, high-dimensional structures and massive datasets. As a result, there has been growing interest in alternative approaches that can overcome these limitations.
Zeroth-order methods have emerged as a promising alternative by avoiding the heavy computational cost associated with calculating gradients or Hessians, zeroth-order methods offer the

potential for more efficient and robust solutions to the challenges posed by bilevel optimization in real-world applications.

2.2 Hyperparameter optimization

2.2.1 Importance of Hyperparameter Optimization

In machine learning, hyperparameters (HPs) refer to parameters external to the model, which cannot be learned from the training data alone. They can be involved in building the structure of the model, such as the number of hidden layers and the activation function, or in determining the efficiency and accuracy of model training, such as the learning rate (LR) of stochastic gradient descent (SGD), batch size. The task to identify the optimal set of hyperparameters is known as hyperparameter optimization (HO). Choosing the appropriate set of hyperparameters has often a dramatic influence on the performance of machine learning models. Optimal hyperparameters can significantly improve model training effectiveness, generalization and final performance, while poor choices can lead to underfitting, overfitting, or inefficient learning. Given the crucial role of hyperparameters in determining the success of machine learning models, hyperparameter optimization has become a critical and active area of research.

2.2.2 Strategies for Hyperparameter Optimization

Several methods have been proposed for hyperparameter optimization in the literature. We will give a brief review on representative ones to illustrate the landscape of hyperparameter optimization strategies, including both traditional techniques, such as grid and random search, as well as more advanced methods like Bayesian optimization and gradient-based approaches.

- **Grid Search** A widely used algorithm where the model is trained over a predefined range of hyperparameter values, and the best-performing configuration based on cross-validation loss is chosen [27, 28]. However, grid search scales poorly with the number of hyperparameters, requiring exhaustive exploration of all possible combinations. This method is also inefficient, as fitting the full model for each value of hyperparameters.

- **Random Search** This method improves upon grid search by randomly sampling hyperparameter values. It has been shown to explore the hyperparameter space more efficiently [29], especially in cases with multiple hyperparameters, but like grid search, it does not utilize prior evaluations to inform future iterations, resulting in slower convergence to optimal hyperparameters.

- **Hyperband Algorithm** An extension of random search that uses a multi-armed bandit strategy to allocate resources more effectively [30]. It evaluates hyperparameter configurations and dynamically allocates computational resources to the most promising ones, reducing the overall cost of hyperparameter optimization.

- **Sequential Model-Based Optimization (SMBO)** Techniques like Bayesian Optimization [31] are increasingly popular for hyperparameter optimization. SMBO builds a probabilistic model to approximate the objective function and iteratively selects the most promising hyperparameter configurations for evaluation. By leveraging past evaluations, Bayesian optimization can efficiently explore the hyperparameter space, particularly when the cost of model evaluation is high.

- **Gradient-Based Methods** In deep learning, gradient-based hyperparameter optimization methods [32] have gained traction, especially when minimizing validation loss. Techniques such as unrolled differentiation or implicit gradients are often employed. However, as discussed in Section 2.1.2, these methods face limitations in computational cost and reliance on strong assumptions like lower-level convexity.

2.2.3 Modeling Hyperparameter Optimization as a Bi-level Problem

Hyper optimization can be the most straightforward application of BLO. Before modelling the hyperparameter optimization problem as a bilevel problem, we provide the general formulation of a bilevel optimization problem, i.e., an outer optimization problem contains a nested (inner) optimization problem within it. The outer optimization problem is commonly referred to as the upper level problem and the inner optimization problem is commonly referred to as the lower level problem.

$$\min_{\theta} F(\theta, w^*(\theta)) \quad \text{subject to} \quad w^*(\theta) = \arg\min_{w} f(\theta, w) \tag{2.1}$$

where

- $F(\theta, w^*(\theta))$ is the upper-level objective function, with θ being the upper-level variable and $w^*(\theta)$ representing the solution of the lower-level problem.

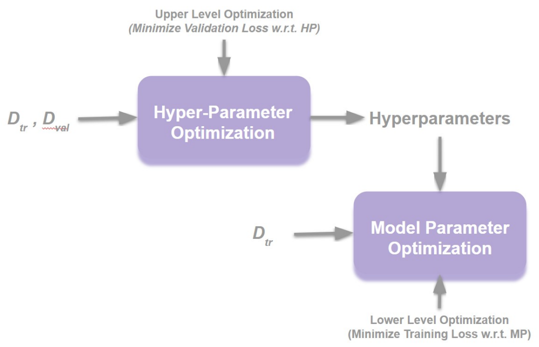
- $f(\theta, w)$ is the lower-level objective function, where w represents the lower-level variables that are optimized for a given value of θ.

In the hyperparameter optimization problem, the lower-level problem minimizes the training set loss, and the upper-level problem minimizes the validation set loss 2.1. The loss on the training set is a function of both the model parameters ω and the hyperparameters θ, while the loss on the validation set is only a function of the model parameters ω. Specifically, the upper-level objective $F(\theta, \omega(\theta); D_{val})$ aims to minimize the validation set loss with respect to the hyperparameters θ, and the lower-level objective $f(\theta, \omega; D_{train})$ seeks to output model parameters ω by minimizing the training loss with respect to ω (e.g., weights and biases).

$$\min_{\theta} F(\theta, w(\theta); D_{\text{val}}) \quad \text{subject to} \quad w^*(\theta) = \arg\min_{w} f(\theta, w; D_{\text{train}}) \tag{2.2}$$

where

- $f(\theta, w; D_{\text{train}})$ is the training loss function over the training set D_{train}.
- $F(\theta, w(\theta); D_{\text{val}})$ is the validation loss function over the validation set D_{val}.
- θ represents the hyperparameters (e.g., learning rate, regularization coefficients).
- w represents the model parameters (e.g., weights and biases).


Upper Level Optimization
(Minimize Validation Loss w.r.t. HP)
D_{tr}, D_{val}
Hyper-Parameter Optimization
Hyperparameters
D_{tr}
Model Parameter Optimization
Lower Level Optimization
(Minimize Training Loss w.r.t. MP)

The UL subproblem involves optimization of hyper-parameters (HP) based on (D_{train} , D_{val}), while the LL subproblem involves optimization of model parameters (MP), aiming to find the optimal model based on D_{train}.

Figure 2.1: Schematic diagram of Hyperparameter Optimization [1]

2.3 Application in Data Hyper-cleaning

In this work, we apply the proposed bilevel optimization method to the specific hyperparameter optimization task of data hyper-cleaning [33], which is particularly valuable in scenarios where obtaining a large, clean dataset is costly or impractical, as in the case of recommendation systems or large-scale machine learning applications.

The data hyper-cleaning task involves optimizing hyperparameters to improve the robustness of the model against noisy data. Suppose we have a dataset with label noise, and due to time or resource constraints, we can only afford to clean up a subset of the available data by manually checking and correcting the labels. In this situation, we can use the cleaned data as the validation set and the remaining noisy data as the training set. By assigning a binary hyperparameter, referred to as a 'mask,' to each training example, we can determine whether a data point should be emphasized or de-emphasized during training. A sparsity constraint is imposed on these masks, helping to focus the learning process on the most relevant data and reduce the impact of noisy examples, thereby leading to a more robust model. In the context of bilevel optimization, the upper-level objective optimizes hyperparameters to minimize the validation loss on the clean dataset, while the lower-level problem learns a model on the noisy training set, using the mask to down-weight noisy samples.

In the following chapters, we will further explore how our proposed method is applied to the data hyper-cleaning task, detailing the experimental setup, results, and analysis. Through this application, we aim to demonstrate the effectiveness of our bilevel optimization approach in handling noisy data and improving model robustness.

3
Methodological Foundations

In this chapter, we build upon established methods in bilevel optimization to address the challenges of binary hyperparameter optimization in large-scale problems. Specifically, we adopt the Relax and Penalize method proposed by Marianna De Santis, Jordan Frecon, Francesco Rinaldi, Saverio Salzo, and Martin Schmidt (2023) [13], which effectively addresses the optimization of mixed-binary hyperparameters by using a continuous reformulation with an appropriate penalty term. This method ensures mixed-binary solutions by iteratively solving a sequence of continuous and penalized problems under certain assumptions,

Additionally, for the upper-level problem, we leverage the Zeroth-Order Frank-Wolfe (ZOFW) algorithm, which was introduced by Anit Kumar Sahu, Manzil Zaheer, and Soummya Kar (2019) [12]. This method provides a gradient-free solution, reducing the computational burden associated with gradient-based methods we mentioned previously.

3.1 Relax and Penalize Method

3.1.1 Introduction and Challenges in Binary Hyperparameter Optimization

Binary hyperparameter optimization has gained increasing importance in deep learning, driven by two trends: The first trend involves the upscaling of neural networks to enhance accuracy.

Empirical studies consistently demonstrate that deeper and wider neural networks outperform simpler models in accuracy [34, 35, 36]. However, this complexity also leads to challenges such as increased inference costs and difficulties in deploying models on resource-limited devices [37]. As a result, model pruning—where redundant parameters are removed—has gained significant research attention. Pruning not only reduces model size and inference cost but also enhances model robustness and generalization, such as adversarial robustness [38], out-of-distribution generalization, and transfer learning [39]. The second trend emphasizes the efficiency of training data usage. While complex modern models have the strong capability to leverage huge amounts of data, there is a growing interest in achieving comparable training performance using smaller subsets of data. This aligns with the objectives of techniques such as one-shot learning, where the goal is to train models with minimal data while maintaining high accuracy. The ability to effectively prune the training data—selecting only the most informative examples—can significantly reduce the computational burden and improve training efficiency.

Both of these trends—model pruning and data pruning—fall under the broader scope of binary hyperparameter optimization, where binary variables, often referred to as mask variables, play a crucial role to determine whether model parameters or data points should be retained or discarded. However, optimizing binary variables introduces additional complexity compared to continuous variables, significantly complicating the optimization process.

The usual optimization approach for binary hyperparameters is that of relaxing the respective parameter over the unit interval [0, 1], solving the continuous optimization problem, and then rounding the solution so to get a binary output. This is essentially a heuristic, which overcomes the challenge of dealing with integer variables, but lacks theoretical guarantees. Our study proposes the adoption of the "Relax and Penalize" method that relies on improved mathematical grounds. This method integrates a penalty function directly into the optimization process, guiding the relaxed binary variables toward binary values throughout the optimization with an iterate style. By employing this method, we aim to study the efficiency and robustness of binary hyperparameter optimization.

3.1.2 Theoretical Foundations and Algorithmic Structure of the Relax and Penalize Method

Firstly, we define the mixed-binary optimization problem(1) as follows:

$$\begin{aligned} &\min_{\lambda,\theta} F(\lambda, \theta, w(\lambda, \theta)) \\ &\text{s.t. } \lambda \in \Lambda \subseteq \mathbb{R}^m, \theta \in \Theta_{\text{bin}} \subseteq \{0,1\}^p, \\ &w(\lambda, \theta) = \arg \min_{w \in W(\lambda,\theta)} f(\lambda, \theta, w) \end{aligned} \tag{3.1}$$

where $F, f : \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}$ represent the upper and lower-level objective functions, $W(\lambda, \theta) \subseteq \mathbb{R}^d$ is the unique solution of the lower-level problem and the hyperparameters λ and θ are continuous and binary variables, respectively. In the hyperparameter optimization problem of machine learning, F and f typically correspond to the loss over a validation set and a training set, respectively.

The Relax and Penalize method [13] tackles the mixed-binary optimization problem by relaxing the binary constraints on θ through a smooth, concave, quadratic penalty function. This function incrementally pushes the relaxed values of θ toward binary values during the optimization process. The set θ is a larger continuous one embedding the binary set Θ_{bin}, and assume that the lower-level problem admits a unique solution also when $\theta \in \Theta$. The original mixed-binary BLO problem is transformed into a continuous one as follows:

$$\min_{\lambda,\theta \in \Lambda \times \Theta} F(\lambda, \theta, w(\lambda, \theta)) + \frac{1}{\epsilon}\varphi(\theta) \tag{3.2}$$

The penalty function $\varphi(\theta)$ is defined as:

$$\varphi(\theta) = \sum_{i=1}^{p} \theta_i(1 - \theta_i) \tag{3.3}$$

This penalty function is designed to reach its minimum when θ_i is close to 0 or 1, thus driving the relaxed variables towards binary values during the optimization. The parameter ϵ controls the weight between the original objective function $F(\lambda, \theta, w(\lambda, \theta))$ and the penalty term $\varphi(\theta)$. By adjusting ϵ, the optimization process would gradually focus more on ensuring that θ approaches binary values.

To guarantee that solving the relaxed problem is equivalent to solving the original mixed-binary problem, we introduce three key theorems require that the following assumption holds [13]:

Assumption 1. *(i)* $\Lambda \subseteq \mathbb{R}^m$ *is nonempty and compact.*

(ii) $\Theta_{bin} := \Theta \cap \{0,1\}^p \neq \emptyset$*, where* $\Theta \subseteq [0,1]^p$ *is convex and compact, and* $\Theta \setminus \Theta_{bin} \neq \emptyset$*.*

(iii) $F : \Lambda \times \Theta \to \mathbb{R}$ *is continuous.*

(iv) For all $\lambda \in \Lambda$*, the map* $F(\lambda, \cdot)$ *is Lipschitz continuous with constant* $L > 0$ *on* Θ*.*

Theorem 1. *Suppose that Assumption 1 is satisfied. Then, there exists an* $\bar{\epsilon} > 0$ *such that for all* $\epsilon \in]0, \bar{\epsilon}]$*, the problem 3.1 and 3.2 have the same global minimizers, i.e.,*

$$\underset{(\lambda,\theta)\in\Lambda\times\Theta_{bin}}{\arg\min}\ F(\lambda, \theta, w(\lambda, \theta)) = \underset{(\lambda,\theta)\in\Lambda\times\Theta}{\arg\min}\left(F(\lambda, \theta, w(\lambda, \theta)) + \frac{1}{\epsilon}\varphi(\theta)\right).$$

For sufficiently small ϵ the global minimizers (i.e., the best solutions across the entire search space) of the relaxed problem match those of the original mixed-binary problem. In simpler terms, solving the relaxed problem still yields valid binary solutions when ϵ becomes close to zero.

This theorem is crucial because it provides a theoretical foundation for the relaxation process. It tells us that we can relax binary variables into continuous ones and still recover correct solutions as ϵ approaches zero. This makes the Relax and Penalize method reliable for hyperparameter optimization, ensuring the relaxed problem isn't just an approximation but leads to the same solutions as the original binary-constrained problem.

Theorem 2. *Suppose that Assumption 1 holds. Let* $c \in]0, 1/2[$ *and* $0 < \epsilon < \frac{(1-2c)}{L}$*. Moreover, let* $(\bar{\lambda}, \bar{\theta})$ *be a local minimizer of*

$$F(\lambda, \theta, w(\lambda, \theta)) + \frac{1}{\epsilon}\varphi(\theta) \quad on \quad \Lambda \times \Theta.$$

If $dist_\infty(\bar{\theta}, \Theta_{bin}) := \inf_{\theta\in\Theta_{bin}} \|\bar{\theta} - \theta\|_\infty < c$*, then* $\bar{\theta} \in \Theta_{bin}$*.*

Theorem 2 essentially says that, even if the optimization process converges to a local minimum instead of a global one, the solution is still valid for the original mixed-binary problem under specific conditions. Specifically, the local minimizers of the relaxed problem correspond to valid solutions in the original problem when ϵ is small enough.

In real-world scenarios, finding a global solution is challenging. This theorem reassures that even local minima, under certain conditions, still produce valid binary solutions. This allows the optimization process to be more flexible while maintaining the binary integrity of the solution, making it practical and applicable to real-world problems.

To addressing problem 3.1 iteratively, a sequence of problems of the form 3.2, indexed with k, with decreasing parameters ϵ^k is the problem to be solved in each iteration:

$$\min_{(\lambda,\theta)\in\Lambda\times\Theta} F(\lambda,\theta,w(\lambda,\theta)) + \frac{1}{\epsilon^k}\varphi(\theta), \quad (P^k)$$

Theorem 3. *Suppose that Assumption 1 holds. Let $(\epsilon^k)_{k\in\mathbb{N}}$ be a vanishing sequence of positive numbers and, for every $k \in \mathbb{N}$, let (λ^k,θ^k) be a local minimizer of (P^k). Then,*

$$\liminf_{k\to+\infty} dist_\infty(\theta^k, \Theta_{bin}) < 1/2 \implies \exists k \in \mathbb{N} \textit{ such that } \theta^k \in \Theta_{bin}.$$

Moreover, if $\theta^k \in \Theta_{bin}$, then we have that λ^k is a local minimizer of

$$\min_{\lambda\in\Lambda} F(\lambda,\theta^k,w(\lambda,\theta^k)).$$

This theorem introduces an iterative framework for applying the penalty method. By progressively reducing ϵ (e.g., ϵ^k for each iteration k), the penalty method incrementally tightens the binary constraints, ensuring convergence to a binary solution without requiring explicit rounding.Instead of forcing binary values at a single step, this theorem outlines a step-by-step tightening of the constraints, allowing the algorithm to systematically converge to the desired binary solution. It's useful because it avoids the need for explicit rounding and provides a more natural and efficient way to enforce the binary constraints. The iterative nature of the process ensures that solutions are improved with each step, making the optimization more robust.

Following the theoretical foundations, we now present the complete algorithmic structure of the Relax and Penalize method. This approach iteratively adjusts the penalty parameter ϵ, progressively tightening the binary constraints on θ, and ultimately converging towards a binary solution.

- **Step 1 (Initialization)** The algorithm starts with an initial penalty parameter $\epsilon^0 > 0$ and a reduction factor $\beta \in (0,1)$. The reduction factor determines how quickly the penalty parameter decreases, progressively tightening the binary constraints on θ. A larger value of β leads to slower tightening, allowing the algorithm to explore the solution space more thoroughly, whereas a smaller β quickly enforces binary constraints but may risk getting trapped in local minima.

- **Step 2 (Iteration)** At each iteration k, the algorithm solves the relaxed optimization

Algorithm 3.1 Relax and Penalize Penalty Method [13]

```
Require: Problem (3.1), $\epsilon^0 > 0, \beta \in (0,1)$
  for $k = 0, 1, 2, \ldots$
      Let $(\lambda^k, \theta^k)$ be a solution (either local or global) of problem $P^k$
      if $\theta^k \notin \{0,1\}^p$
          Update $\epsilon^{k+1} = \beta\epsilon^k$
      else
          Return $(\lambda^k, \theta^k)$
  end for
```

problem for the current value of ϵ^k, obtaining a candidate solution (λ^k, θ^k).

- **Step 3 (Binary Constraint Check)** The algorithm checks whether the solution θ^k satisfies the binary constraints, i.e., whether $\theta^k \in \{0,1\}^p$. If not, it updates the penalty parameter ϵ^k by multiplying it by β, making the penalty stronger in the next iteration.

- **Step 4 (Termination)** If the solution θ^k satisfies the binary constraints, the algorithm terminates and returns the final solution (λ^k, θ^k).

This algorithmic structure, combined with the theoretical guarantees provided by the theorems, ensures that the Relax and Penalize method effectively solves mixed-binary bilevel optimization problems under certain assumptions.

3.1.3 Key Considerations for the Relax and Penalize Method

When applying the Relax and Penalize method to mixed-binary bilevel optimization problems, several critical factors influence the effectiveness and efficiency of the algorithm. These factors include the choice of the penalty parameter ϵ and the decay factor β the initialization of hyperparameters, and the overall computational complexity. Each of these elements plays a pivotal role in ensuring that the algorithm converges to a good solution while maintaining computational feasibility.

Penalty Parameter ϵ and decay factor β The penalty parameter θ and decay factor β are crucial in controlling the trade-off between the original objective function and the penalty term introduced to enforce the binary constraints on β. The choice of these parameters directly influences the convergence behavior of the algorithm:

- Magnitude of θ: If ϵ is too large, the penalty term will have little effect, causing the algorithm to converge slowly towards binary values. Conversely, if ϵ is too small, the penalty term might dominate the optimization process prematurely, potentially leading to suboptimal solutions that overly prioritize satisfying the binary constraints at the expense of minimizing the original objective function.

- Decay factor β: The decay factor β determines how rapidly ϵ decreases during the iterative process. A smaller β leads to a faster tightening of the binary constraints but can also increase the risk of the algorithm converging too quickly, potentially trapping it in local minima. On the other hand, a larger β results in a slower reduction of ϵ, allowing the algorithm to explore the solution space more thoroughly but at the cost of increased computational time.

- Initialization of ϵ and β: Starting with a moderate ϵ allows the algorithm to explore feasible solutions before gradually enforcing binary constraints. Similarly, the choice of β should balance the need for rapid convergence with the necessity of avoiding premature optimization.

Thus, the careful tuning of ϵ and β is essential for balancing convergence speed with solution quality. The iterative process of adjusting these parameters ensures that the algorithm gradually enforces the binary constraints while still allowing sufficient exploration of the solution space.

Initialization of Hyperparameters The initialization of the hyperparameters λ and θ significantly impacts the algorithm's convergence and the quality of the final solution:

- Initial Values of λ and θ: The initial values set for λ and θ dictate the starting point of the optimization process. Poor initialization might lead to slow convergence or entrapment in local minima, while good initialization can expedite convergence and improve the likelihood of reaching a global optimum.

- Initialization Strategies: Employing adaptive strategies for initializing these variables based on the problem's characteristics can enhance the efficiency and robustness of the optimization process. For instance, choosing initial values for θ closer to 0 or 1, depending on prior knowledge or heuristic methods, can help guide the optimization towards a feasible binary solution more effectively.

Proper initialization of λ and θ sets the foundation for the optimization process, making it a crucial factor in the overall success of the Relax and Penalize method.

Computational Cost The computational cost of the Relax and Penalize method is a critical consideration, particularly for large-scale optimization problems:

- Iteration Count and Convergence Speed: The number of iterations required to achieve convergence directly affects the computational burden. While a smaller β may reduce the number of iterations, it could increase the burden of each iteration, as the algorithm might struggle with tighter binary constraints.

- Cost of Function Evaluations and Matrix Operations: Each iteration involves solving a continuous bilevel optimization problem, which requires evaluating the objective function and penalty term. This process can involve computationally expensive operations, such as computing or approximating Hessians, Jacobians, their inverses, and performing matrix operations. These computations grow in cost as the dimensionality of the problem increases.

- Scalability Considerations: As the dimensionality of θ and λ increases, the computational demands grow exponentially. Ensuring that the method scales efficiently with problem size is essential for its application to real-world problems. Parallel computing and efficient algorithms for matrix operations can mitigate some of the computational burdens.

Managing the computational cost requires a careful balance between the speed of convergence and the available resources. Optimizing the algorithm's implementation and leveraging computational resources effectively can help address these challenges.

In summary, the effectiveness of the Relax and Penalize method in mixed-binary bilevel optimization hinges on several key factors. The careful tuning of the penalty parameter ϵ and decay factor β, thoughtful initialization of hyperparameters, and a keen awareness of computational cost all contribute to the success of the method. By understanding and managing these aspects, the Relax and Penalize method can be effectively applied to complex optimization tasks, ensuring both convergence to a high-quality solution and efficient use of computational resources.

3.2 Zeroth-Order Frank-Wolfe Algorithm

3.2.1 Motivation for Using Zeroth-Order Methods in Binary Hyperparameter Optimization

In previous sections, we discussed the challenges associated with bilevel optimization, particularly when using the mainstreamed gradient-based methods for hyperparameter optimization.

These methods face significant challenges when applied to high-dimensional and large-scale optimization tasks. The computation of high-dimensional gradients, Jacobians, Hessians, and complex matrix operators often results in significant computational overhead. As such, they become computationally prohibitive, especially in complex machine learning applications involving millions of parameters. The computational burden is exacerbated in large-scale applications, where solving these problems involves managing millions of variables. As outlined in Section 2.1, there is a critical need to explore more efficient optimization strategies that can scale with increasing problem size and complexity.

In light of these challenges, particularly in the context of binary variable optimization where each iteration involves solving a continuous bilevel optimization problem after relaxation (as discussed in Section 3.1 with the relax and penalize method), the cost of each iteration—both in terms of time and computational resources—highlights the necessity for a more efficient optimization algorithm. This sets the stage for exploring zeroth-order methods, which bypass the need for explicit gradient computations or hessian inversion by using function evaluations to estimate necessary gradients, and that are more computationally feasible for high-dimensional and large-scale optimization scenarios.

Thus, the adoption of the Zeroth-Order Frank-Wolfe algorithm is motivated not only by the computational challenges inherent to gradient-based methods but also by the practical need for efficient employing the relax and penalty strategy. Integrating ZOFW with the Relax and Penalize method aims to enhance the efficiency and scalability of the optimization process, making it more suitable for high-dimensional and large-scale problems.

3.2.2 Overview of Zeroth-Order Frank-Wolfe Algorithm

The Zeroth-Order Frank-Wolfe (ZOFW) algorithm is an extension of the classical Frank-Wolfe method, tailored for optimization tasks where gradient information is unavailable or too expensive to compute. While the traditional Frank-Wolfe algorithm relies on gradients to determine search directions in constrained optimization problems, ZOFW circumvents this reliance by approximating gradients through function evaluations, This gradient-free nature makes ZOFW highly suitable for large-scale and high-dimensional optimization problems.

Here are the core elements of the Zeroth-Order Frank-Wolfe Algorithm:

I. Gradient Estimation via Function Values: ZOFW approximates gradients [12] using finite differences, where function values are evaluated at carefully selected points within the feasible space. This allows for the estimation of a descent direction without needing explicit gradient information, significantly reducing computational complexity.

II. Zeroth-Order Gradient Approximation: For stochastic optimization tasks, ZOFW leverages randomized gradient estimation techniques, including Kiefer Wolfowitz stochastic approximation (KWSA) [40], Random Directions Stochastic Approximation (RDSA) [41, 42] and improved variant, i-RDSA, which samples multiple directional derivatives at each iteration and averages them to achieve more accurate gradient estimates. We adopt i-RDSA which reduces variance and stabilizes the optimization process.

III. Gradient Averaging: To avoid potential divergence caused by non-decaying gradient noise and bias, ZOFW implements a gradient averaging technique [43, 44, 45]. Gradient averaging reduces noise and bias, yielding a more reliable surrogate gradient estimate.

IV. Linear Optimization for Direction Update: In each iteration, ZOFW solves a linear optimization problem over the feasible set to identify the optimal update direction. This step simplifies the optimization process by avoiding the need for complex projection steps, which are typically computationally intensive in gradient-based methods. The efficient linear optimization contributes significantly to the computational efficiency of ZOFW.

These key components enable ZOFW to strike a balance between accuracy and computational efficiency, making it well-suited for bilevel optimization tasks where computational costs can escalate due to the iterative nature of solving both upper- and lower-level problems. The gradient-free nature of the algorithm, combined with the ability to handle large-scale problems, provides a robust framework for optimizing binary hyperparameters in complex settings.

In the next section, we will formally introduce our proposed algorithm, which combines the Relax and Penalize method (from Section 3.1) with ZOFW to tackle binary hyperparameter optimization efficiently in bilevel optimization problems.

3.3 Proposed Algorithm for Binary Hyperparameter Optimization

In this section, we introduce our proposed bilevel optimization algorithm for binary hyperparameter tuning, which integrates the Relax and Penalize method (as outlined in Section 3.1) with the Zeroth-Order Frank-Wolfe optimization (discussed in Section 3.2). This hybrid approach efficiently handles binary constraints through continuous relaxation while addressing the computational challenges of bilevel optimization, particularly in high-dimensional and large-scale problems. Before presenting the algorithm, recall that the optimization process relies on the relaxed formulation of the mixed-binary bilevel problem, as introduced in Section 3.1.2. The relaxed problem is governed by the penalty term $\varphi(\theta)$, driving the relaxed variables towards binary values, as defined in Equation 3.3. The following algorithm outlines the iterative optimization process based on this relaxed and penalized formulation.

The proposed algorithm is structured around two nested loops:

- **Bilevel Optimization Loop (indexed by j)**: This loop is nested within each outer iteration k. It iterates over the bilevel optimization process, where the lower-level and upper-level optimization steps are executed alternately.
  - **Lower-Level Optimization**: At each iteration j, the lower-level problem is solved to find the optimal lower-level variable $w_j^{k*}(\theta_j^k, \lambda_j^k)$ by T inner loops given the current upper-level parameters θ_t^k and λ_j^k.
  - **Upper-Level Optimization**: The upper-level variables θ_j^k are updated through the Zeroth-Order Frank-Wolfe method via gradient approximation and optimization of linear programming. The updated θ_{j+1}^k is for the next step [*].
- **Outer Loop (indexed by k)**: It governs the overall penalty method iterations, adjusting ϵ^k and determining whether the current θ_J^k has converged to a binary solution or requires further refinement.

[*]Multiple steps of i-RDSA for a fixed penalty parameter to guarantee convergence would make the algorithm to computationally heavy in the big data regime.

Algorithm 3.2 Relax and Penalize with Zeroth-Order Frank-Wolfe

```
**Input:** Problem 3.1, $\varepsilon^0 > 0, \beta \in ]0,1[$
**for** $k = 0, 1, 2, \dots$
    Bilevel Optimization Loop
    **for** $j = 0, 1, \dots, J-1$
        **Lower-Level Optimization:**
        Solve the lower-level problem to obtain the optimal solution of the lower-level
        variable with T inner iterations
        $w_j^{k*}(\theta_j^k, \lambda_j^k)$ given the current upper-level variables $\theta_j^k, \lambda_j^k$
        **Upper-Level Optimization:**
        Update the upper-level variables $\theta_j^k$ based on the computed optimal direction
        employing Zeroth-Order Frank-Wolfe as the following steps:
            *Gradient Approximation (i-RDSA):*
            Sample multiple directions $\{z_{i,j}\}_{i=1}^m \sim \mathcal{N}(0, I_d)$
            Estimate the gradient of the upper-level objective
            $F(\lambda_j^k, \theta_j^k, w_j^{k*}) + \frac{1}{\varepsilon^k}\varphi(\theta_j^k)$ using finite differences:

            $$g(\theta_j^k) = \frac{1}{m}\sum_{i=1}^{m} \frac{F(\lambda_j^k, \theta_j^k + c_j z_{i,j}, w_j^{k*}) - F(\lambda_j^k, \theta_j^k, w_j^{k*})}{c_j} z_{i,j} + \frac{2}{\varepsilon}\theta_j^k(1-\theta_j^k)$$

            *Averaging the gradient approximation:*
            Update $d_j$ using a moving average to smooth the gradient:

            $$d_j = (1-\rho_j)d_{j-1} + \rho_j g(\theta_j^k)$$

            *Solve Linear Subproblem:*
            Solve the linear program to find the update direction $v_j$:

            $$v_j = \arg\min_{v\in\Theta}\langle v, d_j\rangle$$

            *Update $\theta_j^k$:*
            Update the upper-level variable $\theta_j$ using the direction $v_j$:

            $$\theta_{j+1}^k = (1-\gamma_j)\theta_j^k + \gamma_j v_j$$

    **end for**
    **if** $\theta_J^k \notin \{0,1\}^p$
        Update $\varepsilon^{k+1} = \beta\varepsilon^k$
    **else**
        Return $(\lambda_J^k, \theta_J^k)$
**end for**
```

4 Experiments and Analysis

4.1 Problem Formulation

In this experiment, we approach the data hyper-cleaning problem as a binary bilevel optimization task, where the goal is to optimize a noisy training dataset by learning a binary mask that determines which data points should be emphasized or discarded during training. The objective is to minimize the validation loss on a clean validation set. The bilevel optimization formulation is expressed as follows:

$$\min_{\theta} \ell_{\text{val}}(w(\theta)), \quad \text{s.t.} \quad w(\theta) = \arg\min_{w'} \left(\ell_{\text{train}}(w', \theta) + c\|w'\|^2\right)$$

Where:

- $\ell_{\text{val}}(w(\theta))$ represents the validation loss on a clean validation set $\mathcal{D}_{\text{val}}$, with the model parameters w being a function of the mask θ.
- $\ell_{\text{train}}(w', \theta)$ is the training loss on the noisy training dataset $\mathcal{D}_{\text{train}}$, weighted by the mask θ.
- $\theta \in [0, 1]^p$ is the binary mask variable, representing the importance of each data point during training, where $\theta_i \approx 0$ excludes the data point from training and $\theta_i \approx 1$ includes it.

- c is a regularization term to prevent overfitting by penalizing large model parameters.

Relaxed Problem with Penalty Term:

$$\min_{\theta} \ell_{\text{val}}(w(\theta)) + \frac{1}{\varepsilon}\varphi(\theta), \quad \text{s.t.} \quad w(\theta) = \arg\min_{w'} \left(\ell_{\text{train}}(w', \theta) + c\|w'\|^2\right)$$

$$\varphi(\theta) = \sum_{i=1}^{p} \theta_i(1 - \theta_i)$$

Where:

- $\varphi(\theta)$ is a penalty function that encourages θ to take binary values (0 or 1), as discussed in Section 3.1.

- ℓ_{train} is a weighted sum over the noisy training set. The weighting function $\sigma(\theta_i)$, implemented as Clip$(\theta_i, [0, 1])$, ensures that each mask variable θ_i is restricted to the range $[0, 1]$. This clipping operation helps prevent extreme values that could overly amplify or suppress the contribution of certain data points, thereby maintaining stability in the training process.

4.2 Numerical Experiments

4.2.1 Experiment Setup and Evaluation Metrics

In this study, we evaluate the performance of the proposed bilevel optimization algorithm using two datasets following the guidelines of [33]: a synthetic binary classification dataset and the widely recognized MNIST dataset [14], and the lower-level training use the general SGD optimization. The synthetic dataset, randomly generated, is employed to observe how the algorithm behaves under different noise conditions and to examine its convergence patterns. The MNIST dataset, known for its use in digit classification tasks, is modified by introducing noise into the labels to simulate real-world noisy data scenarios. In both datasets, we introduce noise by randomly flipping a specific percentage of the labels. Specifically, noise ratios of 20% and 50% are used to simulate low- and high-noise environments, respectively. The goal is to optimize a binary mask, θ, which filters out clean data points, mitigating the adverse effects of noisy data during training.

In addition to examining different noise levels, we also explore the influence of varying inner-loop iteration counts in the lower-level optimization process, denoted as T. Since the lower-level optimization is executed multiple times during the upper-level optimization, and each random direction in the Zeroth-Order Frank-Wolfe method (i-RDSA) involves a lower-level optimization step, T becomes a crucial factor affecting the algorithm's overall performance. We experiment with T values of 1, 2, 5, and 10 across different noise levels to assess how the number of inner-loop iterations impacts the results.

To comprehensively assess the algorithm's performance, we employ the following key evaluation metrics:

- **Test loss over iteration**: Tracks how quickly the test loss decreases for different T values, showing the convergence speed and stability of the algorithm under different settings.
- **Time to achieve target loss**: Measures the time taken for each T value to reach a predefined loss target (e.g., 0.02), highlighting the trade-off between computational cost and convergence speed.
- **Validation loss over iteration**: Tracks validation loss over time to evaluate the generalization ability of the model during optimization.
- **Comparison of test loss between the model trained by filtered and noisy data**: Evaluates the impact of data filtering on model performance by comparing the test loss from models trained on filtered versus noisy data, providing evidence of the algorithm's effectiveness in reducing the negative impact of noise.

By using these evaluation metrics, we can thoroughly analyze the algorithm's performance in terms of convergence speed, generalization ability, computational efficiency, and the quality of the filtered data. In the following section, we present the results of these experiments, analyzing the effects of noise levels and inner-loop iteration counts on both datasets.

4.2.2 Experiment Results and Analysis

In this section, we present the experimental results of our proposed bi-level optimization approach for data hyper-cleaning. The experiments were conducted using both synthetic and MNIST datasets under varying noise conditions. The focus of the analysis is on the impact of different noise levels and the effect of varying the maximum number of lower-level iterations (denoted as T) on model performance, as described in Section 4.2.1.

Test Loss vs. Iteration

The test loss progression for different T values is illustrated in Figure 4.1. The results show that increasing T generally leads to faster and more stable convergence.

For the synthetic dataset, as shown in Figure 4.1 (top row), the test loss consistently decreases across all T values under both noise conditions (0.2 and 0.5) in most cases. When the noise ratio is 0.2, increasing T results in faster convergence. Specifically, $T = 20$ achieves the lowest final test loss. At a noise ratio of 0.5, the impact of increasing T remains the same, but the performance differences between $T = 10$ and $T = 20$ are minimal. $T = 20$ again reaches the lowest final test loss, but the computational time required is the most. When $T = 1$, the test loss increases slowly and and keep the lowest test loss. This suggests that in different noise settings, the proper values of T would be different. For the MNIST dataset (bottom row),

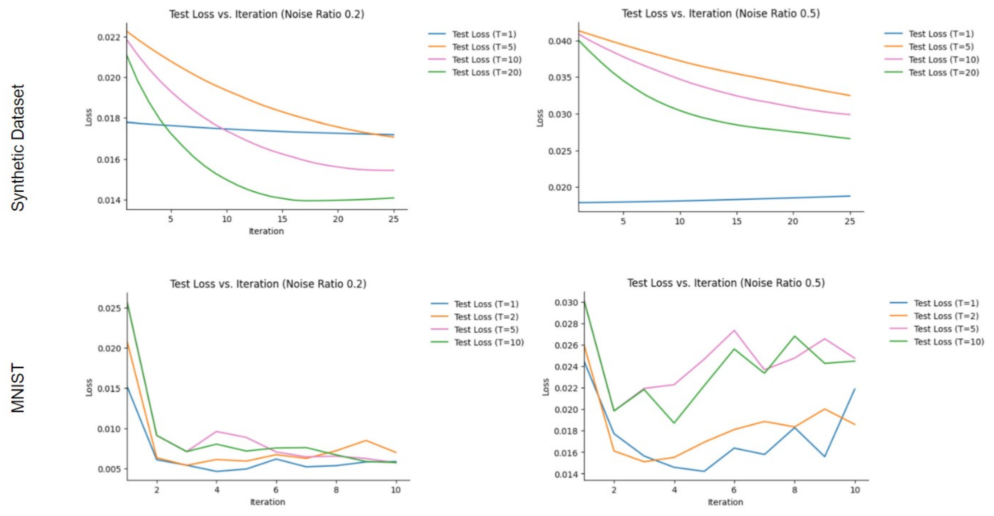


Figure 4.1: Test Loss vs. Iteration under Different Noise Ratios.

the general trends is the same from those observed in the synthetic dataset. The difference is in MNIST test loss decreases more rapidly at the early stage under a noise ratio of 0.2. At a higher noise ratio of 0.5, the results are more fluctuating. Overall test loss decrease rapidly at the beginning however while in the later stages it fluctuates and slightly increases back, even though still lower than the beginning, this implies a concern of instability in the optimization in the high-ratio noise scenario.

Time to Achieve Loss Target The time required to achieve a predefined target loss for different T values is shown in Figure 4.2. The results indicate that increasing T generally leads to slower convergence in terms of reaching the target loss.

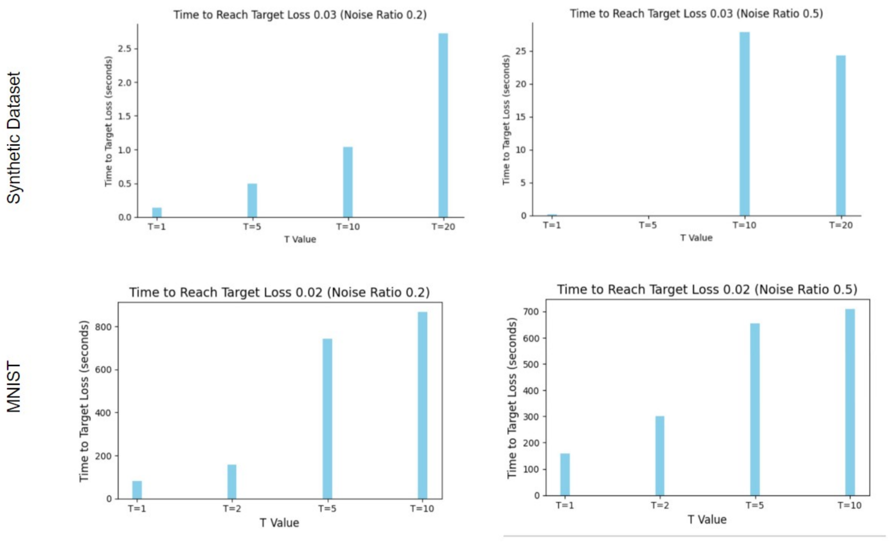


Figure 4.2: Time to Target Loss under Different Noise Ratios

For the synthetic dataset (top row), at a noise ratio of 0.2, $T = 1$ achieves the target loss of 0.03 almost immediately, showing its advantage in time efficiency for low-noise scenarios. However, as T increases to 5, 10, and 20, the time to achieve the target loss also increases, with $T = 20$ taking the longest. At a noise ratio of 0.5, a similar pattern is observed: $T = 1$ still achieves the target loss fastest, while $T = 10$ and $T = 20$ require substantially more time. This suggests that for the synthetic dataset, especially under low noise conditions, smaller T values are more efficient in terms of time to reach the target loss.

In the MNIST dataset (bottom row), the pattern is similar but with more pronounced differences between T values. At a noise ratio of 0.2, $T = 1$ and $T = 2$ reach the target loss of 0.02 quickly, with $T = 1$ being the fastest. $T = 5$ and $T = 10$, however, take significantly longer, especially $T = 10$, which nearly doubles the time compared to $T = 5$. When the noise ratio increases to 0.5, $T = 1$ continues to outperform the other values, achieving the target loss fastest. $T = 5$ and $T = 10$ show a much slower convergence, with both taking almost the same amount of time to reach the target.

In summary, smaller T values consistently reach the target loss more quickly, especially in low-noise environments. While larger T values may lead to better long-term reductions in test loss, they require significantly more time to reach the target loss. For practical applications where time efficiency is a priority, smaller T values may be preferable, particularly in lower-noise settings.

Validation Loss vs. Iteration

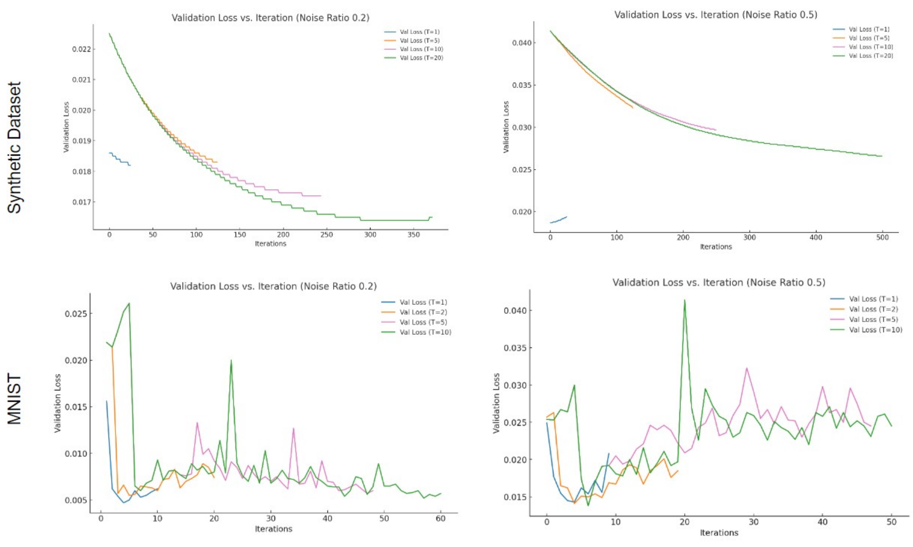


Figure 4.3: Validation Loss vs. Iteration under Different Noise

As shown in Figure 4.3 top row, for the synthetic dataset, the validation loss decreases consistently across all T values for both noise ratios (0.2 and 0.5). At a noise ratio of 0.2, we observe that larger T values (such as $T = 10$ and $T = 20$) lead to faster reductions in validation loss. For smaller T values like $T = 1$, the reduction is slower. At a noise ratio of 0.5, the trend remains similar but with less differences among T values. While $T = 20$ still achieves the lowest final validation loss, the gains compared to $T = 10$ and $T = 5$ are relatively small. This gives a insight in practical applications, we should consider how to reach a good trade-off between time required of training and the best performance.

For the MNIST dataset (Figure 4.3, bottom row), the behavior of validation loss is more unstable compared to the synthetic dataset. At a noise ratio of 0.2, we see that smaller T values (e.g., $T = 1$ and $T = 2$) show sharp fluctuations in the early iterations, though they eventually stabilize around a relatively low validation loss. In contrast, $T = 5$ and $T = 10$ shows some instability initially but ultimately achieves the lower final validation loss. At a noise ratio of 0.5, the validation loss fluctuates significantly, especially for larger T values like $T = 10$. The instability in $T = 10$ suggests that under noisier conditions, the advantage of larger T values is limited, with smaller T values (e.g., $T = 5$) providing a more stable convergence path.

Comparison of Model Performance: Filtered vs. Noisy Data

Training the model with the small set of data which is filtered by the optimized binary mask variable improves the model performance compared to the model trained with the entire noisy dataset. Here is an example of the MNIST dataset, at a noise ratio of 0.2 (Figure 4.4), models trained by filtered data lead to lower test losses for all T values except T=5, especially when T=1 and T=2.

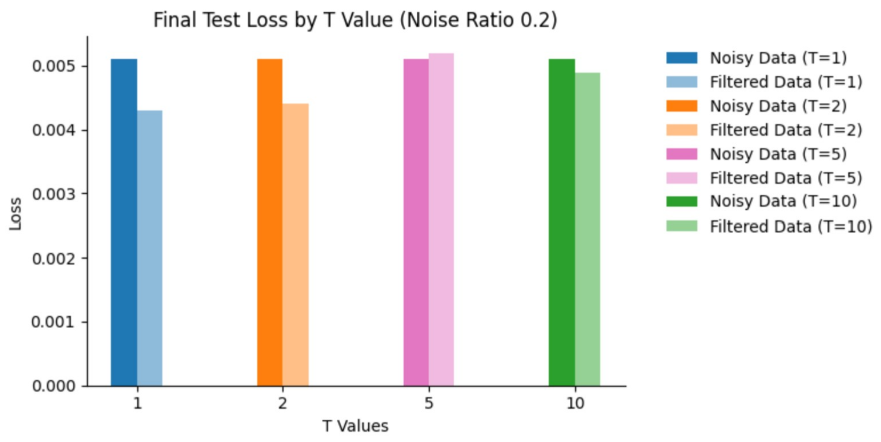


Figure 4.4: Comparison of Model Performance

5 Conclusion and Future Work

5.1 Conclusions

This thesis introduced a bilevel optimization framework designed for binary hyperparameter optimization through the integration of the Relax and Penalize method with the Zeroth-Order Frank-Wolfe algorithm. The numerical experiment is applied to the task of data hyper-cleaning. In our experiments, we specifically focused on the max number of inner-loop iterations (denoted as T) of lower-level (LL) optimization, considering that LL optimization subproblem is essential in any bilevel optimization problem, additional in our proposed method it is involved in the gradient estimation during the upper-level (UL) optimization. Then it is the most computationally demanding of our algorithm. Therefore, we chose the maximum number of LL iterations as the primary subject of our investigation, aiming to study its impact on our proposed algorithm's performance.

Our experiments demonstrated that while increasing the inner-loop iterations (T) generally improves performance, the benefits tend to diminish as noise levels increase. Notably, we observed that the optimal number of iterations differs based on the noise ratio in the data, with lower noise levels requiring fewer iterations compared to higher noise levels. This highlights the importance of selecting an appropriate value of T that balances algorithm effectiveness and computational resource efficiency. In practical applications, choosing a proper T value is crucial for optimizing performance without unnecessary computational overhead. Moreover, with a suitable choice of T, the models trained on filtered data consistently outperformed those trained on unfiltered, noisy data. This outcome demonstrates that, with fewer data points and in less time, we can achieve better training results, which aligns perfectly with our goal of efficient bi level optimization method, efficiency data utilization in data hyper-cleaning task. By selecting the proper number of inner-loop iterations and applying effective data filtering, we not only enhance the model's performance but also reduce the computational cost—achieving the desired balance between model robustness and resource efficiency.

5.2 Future Work

Apart from the number of inner-loop iterations, several other key parameters in our algorithm, such as the penalty parameter, decay factor, the number of random directions used for gradient estimation, the maximum number of outer-loop iterations, and the iteration times of penalty parameter updates—also have a significant impact on the algorithm's performance. Therefore, future work should involve more comprehensive numerical experiments to explore the effects of these parameters in various practical scenarios and to investigate how to select their optimal values. For example, one potential direction for future work is to refine the adjustment of the penalty term's weight during the optimization process. This could help to balance optimization between validation loss and promoting better binarization of the binary variables during the different optimization stages. A promising exploration is dynamically adjusting the decay factor, allowing the penalty term's influence on the objective function to adjust according to the current situation. These studies provide valuable insights that offer practical guidance for the effective implementation of the algorithm. This aligns with the ongoing trend in machine learning of shifting from using large datasets and complex models to more efficient and precise utilization of smaller datasets and lighter models.

Moreover, all the bilevel optimization methods aim to effectively guide the update of optimized variables, which is the core computational complexity. In our algorithm, we employ the Zeroth-Order Frank-Wolfe method for gradient estimation, which is essentially an approximation. Although we use i-RDSA to improve accuracy by averaging multiple estimates, achieving higher precision requires more estimates. Therefore, a key area of interest is how to efficiently obtain these approximate estimates to guide machine learning.

References

[1] R. Liu, J. Gao, J. Zhang, D. Meng, and Z. Lin, "Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond," *IEEE Journal of LaTeX Class Files*, vol. 14, no. 8, pp. 1–20, Aug. 2021.

[2] J. Bracken and J. McGill, "Mathematical programs with optimization problems in the constraints," *Operations Research*, vol. 21, pp. 37–44, 1973.

[3] S. Dempe, *Foundations of bilevel programming*. Springer, 2002.

[4] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, July 2018.

[5] *Hyperparameter Optimization*, Springer Std., 2019, in: Automated Machine Learning: Methods, Systems, Challenges, pp. 3–33.

[6] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, December 2019, pp. 113–124.

[7] Z. Borsos, F. Locatello, O.-E. Ganea, M. Tschannen, G. Rätsch, and F. Hutter, "Coreset: Continual learning along the parameter space," in *International Conference on Learning Representations*, Addis Ababa, Ethiopia, April 2020.

[8] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, Denver, USA, November 1999.

[9] M. Hong, H.-T. Wai, Z. Wang, and Z. Yang, "A two-timescale stochastic algorithm framework for bilevel optimization: Complexity analysis and application to actor-critic," *SIAM Journal on Optimization*, vol. 33, no. 1, pp. 147–180, 2023.

[10] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *Proceedings of the International Conference on Learning Representations*, New Orleans, LA, May 2019.

[11] A. Sinha, T. Khandait, and R. Mohanty, "A gradient-based bilevel optimization approach for tuning regularization hyperparameters," *Optimization Letters*, vol. 18, no. 4, pp. 1383–1404, 2023.

[12] A. K. Sahu, M. Zaheer, and S. Kar, "Towards gradient-free and projection-free stochastic optimization," in *Proceedings of the 2019 Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, Dec. 2019.

[13] M. de Santis, J. Frecon, F. Rinaldi, S. Salzo, and M. Schmidt, "Relax and penalize: a new bilevel approach to mixed-binary hyperparameter optimization," 2023. [Online]. Available: https://arxiv.org/abs/2308.10711

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] H. Von Stackelberg, *Market Structure and Equilibrium*, 1st ed. Springer Science & Business Media, 2010.

[16] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 276–295, 2017.

[17] C. Kolstad and L. Lasdon, "Derivative evaluation and computational experience with large bilevel mathematical programs," *Journal of Optimization Theory and Applications*, vol. 65, pp. 485–499, 1990.

[18] E. Aiyoshi and K. Shimizu, "Hierarchical decentralized systems and its new solution by a barrier method," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, pp. 444–449, 1981.

[19] Y. Ishizuka and E. Aiyoshi, "Double penalty method for bilevel optimization problems," *Annals of Operations Research*, vol. 34, pp. 73–88, 1992.

[20] P. Marcotte, G. Savard, and D. L. Zhu, "A trust region algorithm for nonlinear bilevel programming," *Operations Research Letters*, vol. 29, no. 4, pp. 171–179, 2001.

[21] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, *Forward and reverse gradient-based hyperparameter optimization*, arXiv Std., 2017, arXiv:1703.01785.

[22] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in *Proceedings of the International Conference on Machine Learning*, New York, NY, June 2016, pp. 737–746.

[23] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, Palermo, Sicily, Italy, April 2020.

[24] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, July 2018.

[25] R. Liu, P. Mu, X. Yuan, S. Zeng, and J. Zhang, *A General Descent Aggregation Framework for Gradient-based Bi-level Optimization*, arXiv Std., 2020, arXiv:2102.07976.

[26] A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots, "Truncated back-propagation for bilevel optimization," in *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, Naha, Okinawa, Japan, April 2019.

[27] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, Oregon, June 2007.

[28] G. E. Hinton, "A practical guide to training restricted boltzmann machines," in *Neural Networks: Tricks of the Trade*, Berlin, Germany, June 2012.

[29] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," in *Proceedings of the International Conference on Neural Information Processing Systems*, Granada, Spain, Dec. 2011.

[30] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2018.

[31] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," in *Proceedings of the International Conference on Neural Information Processing Systems*, Vancouver, Canada, Dec. 2010.

[32] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, Lille, France, July 2015.

[33] K. Levy and F. Lieder, "Bome! bilevel optimization made easy: A simple first-order approach," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, Virtual, July 2021.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, June 2016.

[35] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, York, UK, Sept. 2016.

[36] Y. Huang, V. Vasudevan, R. Monga, G. Sun, and Q. V. Le, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, Dec. 2019.

[37] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, CA, USA, June 2019.

[38] V. Sehwag, S. Wang, P. Mittal, and S. Jana, "Hydra: Pruning adversarially robust neural networks," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, New York, NY, USA, Feb. 2020.

[39] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, June 2018.

[40] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *Annals of Mathematical Statistics*, vol. 23, pp. 462–466, 1952.

[41] Y. Nesterov and V. Spokoiny, "Random gradient-free minimization of convex functions," *Foundations of Computational Mathematics*, vol. 2, pp. 157–183, 2011.

[42] J. Duchi, E. Hazan, and Y. Singer, "Stochastic gradient methods for distributionally robust optimization," *Foundations of Computational Mathematics*, vol. 15, no. 3, pp. 715–732, 2015.

[43] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, California, USA, June 2016, pp. 1480–1489.

[44] A. Ruszczyński, *Nonlinear Optimization*. Princeton University Press, 2008.

[45] A. Mokhtari and A. Ribeiro, "A surrogate gradient-based algorithm for stochastic optimization," *IEEE Transactions on Signal Processing*, vol. 66, no. 12, pp. 3270–3282, 2018.