Computational Science and Engineering (Int. Master's Program)

Technische Universität München

Master's Thesis

Neural Network Hyperparameter Optimization using SNOWPAC

Author:	Kislaya Ravi
1st examiner:	Univ. Prof. Dr. Hans-Joachim Bungartz
2nd examiner:	Prof. Dr. Laura Leal-Taixé
Assistant advisor(s):	Friedrich Menhorn, M.Sc. (hons)
	Tim Meinhardt, M.Sc.
Thesis handed in on:	March 14, 2019

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, March 14, 2019 Kislaya Ravi

Acknowledgments

I would first like to thank my thesis supervisors Univ. Prof. Dr. Hans-Joachim Bungartz and Prof. Dr. Laura Leal-Taixé for providing and guiding this interesting topic. I would also like to express my gratitude towards my advisors Friedrich Menhorn and Tim Meinhardt. Their office door was always open whenever I ran into a trouble spot or had a question about my research or writing. They not only gave me a lot of freedom to implement my ideas but also consistently steered me in the right direction whenever they thought I needed it.

I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of studies and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Abstract

Automatic searching for the optimum set of hyperparameters is crucial in practical application of deep learning algorithms. In this work we optimize the hyperparameters using mixed integer SNOWPAC (Stochastic Nonlinear Optimization With Path-Augmented Constraints), a method for stochastic nonlinear constrained derivative free optimization using a trust region approach.

We present new addition to SNOWPAC to solve mixed integer optimization problems. We compare its performance against various existing optimizers using different benchmark problems. Then, we link the it with neural network training to optimize the hyperparameters. We create eight different neural network hyperparameter optimization problems with number of unknown parameters ranging from six to nineteen. We optimize hyperparameter for the problems using SNOWPAC and other existing methods like HORD, HORD-ISP, Spearmint, TPE and SMAC. Then we compare these tools over the different criterions. We show that, statistically SNOWPAC not only finds the set of hyperparameters with lower validation error but also take small non-evaluation time.

Contents

Acknowledgements v

Abstract vii

Outline of the Thesis xi

I. Introduction 1

1. Introduction 3
   1.1. Literature Review . . . 3
   1.2. Motivation . . . 5

II. Method and Theory 7

2. NOWPAC 9
   2.1. Trust-region derivative free unconstrained optimization . . . 9
   2.2. Constrained Optimization . . . 11
   2.3. Surrogate Model . . . 12
   2.4. Poisedness . . . 13
   2.5. Parameter Default Values . . . 15

3. SNOWPAC 17
   3.1. Robustness Measure Formulation . . . 17
      3.1.1. Robustness Measure . . . 18
      3.1.2. Statistical Estimation . . . 18
   3.2. Gaussian Process Supported Trust Region Management . . . 19
      3.2.1. Introduction to Gaussian Process . . . 19
      3.2.2. Gaussian Process Surrogates . . . 20
      3.2.3. Noise Adapted Trust Region . . . 20

4. Mixed Integers 23
   4.1. Local minima of mixed integer optimization problems . . . 23
   4.2. Modifications for Mixed Integers . . . 25
      4.2.1. Trust Region Shape . . . 25
      4.2.2. Surrogate Model Optimization . . . 27
      4.2.3. Branch and Bound . . . 28
      4.2.4. Trust Region Update . . . 30

4.3. Adapting SNOWPAC for Mixed Integer problems 32

5. Artificial Neural Network 33
5.1. Feed forward Neural Network 33
5.2. Supervised Training 36
5.3. Convolutional Neural Networks 37
5.3.1. Convolutional Layer 37
5.3.2. Pooling 38
5.3.3. Fully connected layer 38
5.4. Regularization 38
5.4.1. Weight Decay 38
5.4.2. Dropout 39
5.4.3. Batch Normalization 39
5.5. Hyperparameter Optimization 40
5.5.1. Problem Statement 40
5.5.2. Existing Methods 40

III. Experiment Setup and Results 43

6. Mixed Integer SNOWPAC Performance 45
6.1. Illustrative Two Dimensional Example 45
6.2. Performance Profiling Benchmarks 47
6.2.1. Mixed Integer Box-Constrained Problem 47
6.2.2. Mixed Integer Stochastic Box-Constrained Problem 50

7. Neural Network Hyperparameter Optimization 53
7.1. Experimental Setup 53
7.2. Results and Discussion 55
7.2.1. First Category (Unscaled Hyperparameter Space) 55
7.2.2. Second Category (Scaled Hyperparameter Space) 59
7.2.3. Non-evaluation Time 61
7.2.4. Stochastic Optimization 62

IV. Conclusion 65

8. Conclusion and Future Works 67
8.1. Outcomes 67
8.2. Future Works 68

Appendix 71

A. Benchmark Optimizations Problems 71
A.1. Box-Constrained Problems 71

B. Optimization Benchmark Comparison Graphs **73**

C. Hyperparameter List **83**

D. Hyperparameter Optimization Results **89**

Bibliography **99**

Outline of the Thesis

Part I: Introduction

CHAPTER 1: INTRODUCTION

This chapter presents an overall overview of the thesis and purpose of the work. We take a cursory look at the neural network training and importance of hyperparameters. We further give a brief description of the various techniques used till now to final optimum hyperparameters. Additionally, we include a brief history of the various optimization techniques.

Part II: Method and Theory

CHAPTER 2: NOWPAC

This chapter gives an introduction to deterministic derivative-free nonlinear optimization solver NOWPAC. NOWPAC is trust-region based optimization algorithm. We introduce the underlying idea of the method and give a short description of the NOWPAC algorithm. This is followed by an overview of various terms, definitions and algorithms related to NOWPAC.

CHAPTER 3: SNOWPAC

We provide a brief overview of Stochastic NOWPAC. There is a short description of the algorithm along with the modifications done to the existing method to handle stochastic functions. We describe Gaussian Process used to decrease noise.

CHAPTER 4: MIXED INTEGER OPTIMIZATION

Some optimization problems put additional integer constraints on some of its parameters. This chapter describes the mixed integer trust-region based optimization algorithm. We give a detailed explanation of the Branch and Bound algorithm used to solve the local surrogate optimization. Other modifications to NOWPAC is also briefly described.

CHAPTER 5: ARTIFICIAL NEURAL NETWORKS

Our intent is to apply Mixed Integer SNOWPAC to optimize the hyperparameters. This chapter gives a brief overview of the various features of a neural network and the methods to train it. We formally introduce the hyperparameter optimization problem statement and mention the existing methods used to optimize the hyperparameters.

Part III: Results

CHAPTER 6: MIXED INTEGER SNOWPAC PERFORMANCE

Before applying our solver for hyperparameter optimization, we check the correctness and performance of Mixed Integer SNOWPAC with respect to the benchmark problems. We compare various existing mixed integer optimization methods with SNOWPAC using performance profiling and statistical distribution of evaluation points.

CHAPTER 7: NEURAL NETWORK HYPERPARAMETER OPTIMIZATION

In this chapter, we present results of optimizing hyperparameter using our solver. Firstly, we describe our experimental setup. Then, we present our results to compare SNOWPAC against already existing methods.

Part IV: Conclusion

CHAPTER 8: CONCLUSION

We summarize the outcomes of this thesis. At last, we enlist possible future works.

Part I.

Introduction

1. Introduction

It has been a long desire of mankind to generate machines that can think and make decisions like humans. This leads to the birth of *Artificial Intelligence (AI)*, which is described by Pamela McCorduck as "an ancient wish to forge the gods" [78]. With the advent of computers, people began to hard-code logic that will mimic human thinking. But, it was difficult to find decent logics. This led the research work to move in direction of developing AI systems that has the ability to acquire knowledge by extracting patterns from raw data. This capability is known as *machine learning* [54]. By using machine learning and deep learning, computers started to tackle real-world problems and find patterns from the big set of data. In modern days, neural networks have touched almost every plethora of our lives.

Training a neural network involves various challenges. One of them is to set some parameters manually. They are commonly known as *hyperparameters*. Setting them is a crucial step in training a neural network because final accuracy depends upon it. We introduce a new tool to find the optimum hyperparameters. In this chapter we first present literature review of the neural network, optimization and its application in optimal hyperparameter search in Section 1.1. Then, we provide the intended aim of this work in Section 1.2.

1.1. Literature Review

Brain cells have been a source of inspiration in the field of AI. First attempt to model a brain with electric circuits was done in 1950s, when a neurophysiologist and a mathematician co-wrote a paper on working of neurons [4, 97]. The first neural network applied to a real-world problem was Stanford's MADALINE that used an adaptive filter to remove echoes over phone lines. It is still in use today[97]. Later in 1999, neural network was used in diagnosis of cancer when computer detected cancer more accurately than radiologists [63]. Nowadays, artificial neural network has been successfully used in many fields.

In various literatures we can find numerous approaches of searching the good set of hyperparameters. Earlier, grid search and manual search were the most commonly used strategies. Later it was shown by Bergstra et al. [10], that random search is a more efficient method. Another way to find optimal hyperparameters is to look at it as a mixed integer optimization problem, where one needs to maximize the validation accuracy. The biggest problem in this approach is the unknown functional dependency of the validation accuracy with respect to the hyperparameters. Therefore we do not have any information about the derivative. Moreover, a single training with a given set of hyperparameters (i.e. one black-box evaluation) is computationally expensive. So, we need to find an efficient black-box derivative free optimization method. Bergstra et. al. [13] and Hutter et. al. [60] used bayesian optimization to search optimal hyperparameters. Ilievski et. al. [61] used deterministic trust-region based optimization method.

Optimization problems also have a huge history by itself. Mitri Kitti [68] lists breakthroughs in optimization over last 2300 years. The first written record of an optimization problem appears in Euclid's work Elements [46]. Then, we see mention of a maximization problem of packing of balls in a unit cube by Kepler [42]. This problem has also been dealt by Carl Friedrich Gauss [50]. Other notable scientists who contributed to the field optimization were Galilei, who studied the shape of a hanging chain [55]; Newton, who dealt in minimum resistance against body [102]; Lagrange, who examined the minimal surface problem [101] and Legendre, who came up with the famous least-square problem [79].

To optimize the hyperparameters, we need a derivative free optimization method. There are many real life applications where we need optimization of the design parameters when the functional dependency between the output and the input is unknown. Such functions are knows as *black-box functions*. Many derivative free methods have been developed to tackle such problems. The simplest derivative free methods were the directional grid search method where we sample the objective function at finite points for every optimization iteration. Based on the function values, we decide the next step of optimization. These methods and its variations are discussed in Fermi and Metropolis [6], Davidson [41] and Box et al. [25]. Another approach is the multi-directional search method like the Nelder and Mead simplex method [82].

In this work, we will use a trust-region derivative free method. The basic idea behind these families of methods is firstly to build a surrogate model within a trust-region, then optimize the surrogate model, followed by moving the trust-region to the new optimal point and updating the model. This cycle is iteratively repeated. This concept was used in bits and pieces by Winfield [109] and Glad et al. [52]. The trust-region derivative free algorithm with least square optimization was presented by Powell [87]. There were extensions for non-smooth optimization , see Fletcher [47], and for constrained optimization, see Celis [28], Powell and Yuan [90], Byrd et al. [26], Toint [107] and many others. In this thesis, we extend the optimization method *SNOWPAC* [8] to solve mixed integer problems. SNOWPAC (Stochastic Non-linear Optimizer With Path Augmented Constraints) is a tool that solves both stochastic and deterministic non-linear constrained optimization problems.

In training of a neural network there are many integer hyperparameters. For example, the number of neurons in a given layer cannot be a decimal number. This adds to another key requirement for the optimizers. It should be able to solve mixed-integer optimization problems. There are various algorithms discussed in the past to tackle mixed integer optimization problems, refer Floudas [48], Grossmann and Kravanja [56]. Trust-region based optimization to solve mixed-integer optimization was done by Exler and Schittkowski [45] and Newby [85].

This far we have discussed methods that deal only with deterministic functions. In real-life applications, we encounter many stochastic functions. For training of neural network, we obtain different validation accuracy for different random seeds of weight initialization. This makes our function to be stochastic and puts forward the requirement of our solver to optimize stochastic functions. Robin and Monroe [95] pioneered Stochastic Approximation (SA) in 1951. We refer to [15, 67, 103] for the detailed introduction and analysis of SA. In spite of the rich literature, stochastic optimization still remains a challenging problem [103].

1.2. Motivation

As mentioned before, there are various approaches to optimize hyperparameters. In this thesis, we introduce a new method to optimize hyperparameters. However, all the methods uses deterministic optimization method. In this work we also want to optimize the hyperparameters using SNOWPAC which is a stochastic non-linear optimizer. We can divide our work into two parts.

Firstly, we need to develop a mixed-integer stochastic non-linear optimization tool. The existing version of SNOWPAC can handle only continuous parameters. Therefore, we adapt it to solve mixed-integer problems. We also need to benchmark the developed tool against existing mixed-integer optimization tools.

Secondly, we use the newly made mixed-integer SNOWPAC to optimize hyperparameters of the neural network. This compares the performance of mixed-integer SNOWPAC against existing neural network hyperparameter optimization methods. For that, we should create multiple neural networks, optimize its hyperparameters using all methods and compare performance on various scales.

Part II.

Method and Theory

2. NOWPAC

NOWPAC [7] (Nonlinear Optimization With Path Augmented Constraints) is an algorithm to optimize nonlinear functions subject to non-linear inequality constraints. This algorithm belongs to the family of trust-region derivative free algorithms. We will be looking into the optimization algorithms that is of following form:

$$\begin{aligned} min \quad & f(x) \\ s.t. \quad & c_i(x) \leq 0, \quad i = 1...r \end{aligned} \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function representing black-box and $c_i : \mathbb{R}^n \to \mathbb{R}, \quad i = 1...r$ are the model constraints, we impose on the design parameters $x \in \mathbb{R}^n$

To explain the underlying algorithm of NOWPAC, we divide this chapter in small sections. Firstly, we explain a simple trust-region based method for unconstrained problems in Section 2.1 and for the constrained optimization in Section 2.2. In Section 2.3, we take a cursory look over surrogate model. Finally, we describe how points are chosen within the trust-region and procedure to improve the distribution of points in Section 2.4. Detailed proof of convergence is not described in this thesis. Interested readers can refer to [7]. All steps of NOWPAC is summarized in Algorithm 1.

2.1. Trust-region derivative free unconstrained optimization

Firstly, let us consider an unconstrained optimization problems to understand the fundamentals of the trust-region methods. We start with an initial point(x_0) inside the given domain. The main idea behind this method is to model the objective function in a neighborhood of a given point using any surrogate model(m_k^f) of our choice. The neighborhood considered is called the trust-region, thereby giving the method its name. In NOWPAC [7], the trust-region is in shape of a ball represented as $B(x_k, \rho_k) = \{x \in \mathbb{R}^n : \|x - x_k\| \leq \rho_k\}$, where x_k is the center and ρ_k is the trust-region radius. Then, we can easily optimize the surrogate model using various existing methods because we have detailed information about surrogate. NOWPAC [7] uses Lagrange polynomials to build the surrogate. We calculate the next trial point($\bar{x_k}$) as following:

$$\bar{x_k} = \underset{x \in B(x_k, \rho_k)}{\arg\min} \quad m_k^f(x) \tag{2.2}$$

Then, we calculate the acceptance ratio(r_k) as following:

$$r_k = \frac{f(x_k) - f(\bar{x_k})}{m_k^f(x_k) - m_k^f(\bar{x_k})} \tag{2.3}$$

Algorithm 1: Nonlinear Optimization With Path-Augmented Constraints algorithm.

```
1  Set k = 0 and build initial fully linear surrogate model m_0^f(x), m_0^c(x);
2  while ρ ≥ ρ_min do
       /* STEP 1:  Criticality Step                                        */
3      if α_k(ρ_k) ≤ ε_c then
4          if m_k^f and m_c^f are not fully linear in B(x_k, ρ_k) or ρ_k > μα_k(ρ_k)^(1/2) then
5              Set ρ_k = ωρ_k ;
6              Build fully-linear surrogate model m_k^f and m_k^c ; Goto line 4;
7          end
8      end
       /* STEP 2:  Trial point calculation                                 */
9      x̄_k = argmin_{x∈B(x,ρ)} m_k^f(x) ;
       /* STEP 3:  Check feasibility of trial point                        */
10     if c_i(x̄_k) > 0 then
11         Set ρ_k = γρ_k, build fully-linear surrogate model m_k^f and m_k^c and Goto line 4;
12     end
       /* STEP 4:  Check acceptance of trial point                         */
13     Calculate acceptance ratio r_k = (f(x_k) − f(x̄_k)) / (m_k^f(x_k) − m_k^f(x̄_k));
14     if r_k ≥ η_0 then
15         Set x_{k+1} = x̄_k and include x̄_k into set of sample points;
16         Update fully-linear surrogate model m_{k+1}^f and m_{k+1}^c ;
17     end
18     else
19         Set x_{k+1} = x_k, m_{k+1}^f = m_k^f, m_{k+1}^c = m_k^c ;
20     end
       /* STEP 5:  Trust-region Update                                     */
21     (see equation below)
       /* STEP 6:  Model Improvement                                       */
22     if r_k < η_0 then
23         Improve quality of model m_{k+1}^f and m_{k+1}^c
24     end
25     k = k + 1 ;
26 end
27 Optimal result is x* = x_k
```

Line 21:

$$\rho_k = \begin{cases} \gamma_{inc}\rho_k & \text{if} \quad r_k \geq \eta_1 \\ \rho_k & \text{if} \quad \eta_0 \leq r_k \leq \eta_1 \\ \gamma_{shrink}\rho_k & \text{if} \quad 0 < r_k < \eta_0 \\ \gamma\rho_k & \text{if} \quad r_k \leq 0 \end{cases}$$

Acceptance ratio is a measure of how much the objective function has improved with respect to the surrogate model. We decide the status of the new point based on the value of

acceptance ratio. We choose four parameters namely η_0, η_1, γ and γ_{inc} satisfying $\gamma \in (0, 1)$, $\gamma_{shrink} \in (0, 1)$ and $0 \leq \eta_0 \leq \eta_1 < 1 < \gamma_{inc}$ (with $\eta_1 \neq 0$). Decision is done as per following rules:

1. Step Accepted : If $r_k \geq \eta_0$, then the center of the trust-region is shifted to $\bar{x}$ and it becomes the new best point. Under this condition, trust region radius is updated as:
   - if $r_k \geq \eta_1$ we increase the trust-region radius by factor of γ_{inc}.
   - if $r_k \leq \eta_1$ we do not modify the radius of trust-region
   - if $0 < r_k < \eta_0$, then step is acceptable but with shrink of trust region radius by factor γ_{shrink}
2. Step Rejected : If $r_k \leq 0$ then we reject trial point and decrease radius of trust region by factor of γ.

Then, we update the model(m_k^f and $m_k^{c_i}$) and continue with the steps as mentioned before until the trust-region radius is below a certain limit.

2.2. Constrained Optimization

NOWPAC introduces a new way of handling non-linear black-box constraints using *inner boundary path*. It is an offset function to constraints which locally convexify the feasible domain. We need the domain to be convex because in order to find a unique solution of surrogate model, function and constraints must be convex. The inner boundary path also guides the next trial step to become feasible.

We introduce an offset to the constraint function(*inner boundary path*) as following:

$$h_x(x+d) : \begin{cases} \mathbb{R}^n & \to \mathbb{R} \\ x+d & \mapsto \varepsilon_b \left\| d \right\|^{\frac{2}{1+p}} \end{cases} \tag{2.4}$$

with order reduction $p \in (0, 1)$ and define the *inner-boundary-path-augmented local feasible domain* at $x \in X$ as

$$X_x^{ibp} := \{x + d : c(x + d) + h_x(x + d) \leq 0\} \cap B(x, 1). \tag{2.5}$$

Figure 2.2 shows how inner boundary path make a concave function into convex inside the trust-region. However, this also makes a part of the domain which should be feasible into infeasible. But when trust-region radius decreases then the percentage of the domain that is turned infeasible also decreases.

There is only one modification done in overall algorithm to handle constraints. We check feasibility of trial point after calculating trial point. If the trial point is not feasible then trust-region radius is decreased by a factor of γ.

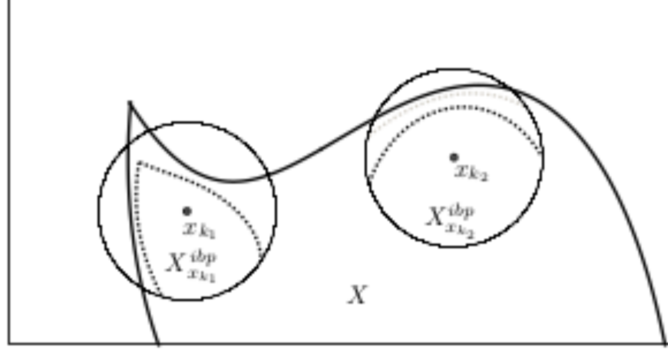


Figure 2.1.: Local convexification(black dotted lines) of feasible domain X (solid line) around two center points x_{k_1} and x_{k_2} with $\varepsilon = 10$ and $p = 0.2$. Gray dotted line shows inner boundary path with $\varepsilon = 2.5$. Trust-region radius is 1 [7]

.

2.3. Surrogate Model

The purpose of the surrogate model is to locally mimic the behavior of the objective function inside the trust-region. The choice of the model and its effects on the convergence is studied in detail in [34, 32, 33]. There are two main factors which the surrogate model should qualify, namely:

1. It should be able to accurately model the feature like slope, curvature etc of the objective function within the trust-region.

2. It should be easy to optimize within the trust-region.

Higher the order of the polynomial better will be its capability to capture the detailed features. However, higher order polynomials come with their own difficulties. Firstly, difficulty to find the local minimum of a polynomial with in the trust-region increases with the increasing order. Secondly, we need more black-box evaluations to make best use of the higher order. But each evaluation can be computationally expensive. These are two contradicting factors. In NOWPAC we build the model which is at least fully linear [7]. This ensures global convergence to a first-order critical point[33]. Model should be able to approach quadratic order, to allow global convergence of second order [33]. Surrogates must be twice continuously differentiable and satisfy the following error bound condition [34]:

$$\left\| f(x_k + s) - m_k^f(x_k + s) \right\| \leq \kappa^f(\nu^f + \left\| H_k^f \right\|)\rho_k^2 \tag{2.6a}$$

$$\left\| \nabla f(x_k + s) - \nabla m_k^f(x_k + s) \right\| \leq \kappa_1^f(\nu^f + \left\| H_k^f \right\|)\rho_k \tag{2.6b}$$

$$\left\| c_i(x_k + s) - m_k^{c_i}(x_k + s) \right\| \leq \kappa^{c_i}(\nu^{c_i} + \left\| H_k^{c_i} \right\|)\rho_k^2 \tag{2.6c}$$

$$\left\| \nabla c_i(x_k + s) - \nabla m_k^{c_i}(x_k + s) \right\| \leq \kappa_1^{c_i}(\nu^{c_i} + \left\| H_k^{c_i} \right\|)\rho_k \tag{2.6d}$$

Therefore, NOWPAC[7] uses polynomial of the quadratic form :

$$m_k(x_k + s) = m_k(x_k) + s^T g_k + \frac{1}{2} s^T H_k s \tag{2.7}$$

where, g_k and H_k represents the gradient and the hessian respectively. So, optimizing the problem is simplified as solving the linear system $H_k s = g_k$. NOWPAC uses *NLopt* [64] to solve the surrogate problems.

NOWPAC starts with building linear model, and gradually improves if the trial point is not acceptable. This forms an under-determined system. From equations 2.6, we conclude that error depends upon the norm of the hessian. The best model will have the smallest norm of Hessian. Therefore, NOWPAC uses the *least frobenius norm* method [34, 7, 88] to form the surrogates. Let, $m(x)$ represent the model, α_L represent coefficient of linear and constant terms in the surrogate model and α_Q represent coefficient of the quadratic terms in the surrogate model

$$m(x) = \alpha_L^T \bar{\phi}_L(x) + \alpha_Q^T \bar{\phi}_Q(x)$$

Let, $Y = \{y^0, y^1, ..., y^p\}$ be set of points inside the trust-region. Only quadratic term contributes to the hessian. So the least frobenius norm model is defined as:

$$\begin{aligned} min \quad & \frac{1}{2} \|\alpha_Q\|^2 \\ s.t. \quad & m(y) = f(y) \qquad \forall y \in Y \end{aligned} \tag{2.8}$$

2.4. Poisedness

The distribution of points inside the trust-region is crucial to capture the behavior of the objective function. Points should be able to explore the domain to capture the features of the function. Here *poisedness* is a measure to quantify how well points are distributed.

Definition 2.1 ([34]) *Let $\Lambda > 0$ and a set $B \in \mathbb{R}^D$ be given. Let $\phi = \{\phi_0(x), \phi_1(x), ..., \phi_p(x)\}$ be a basis in $\mathcal{P}_n^d$. A poised set $Y = \{y^0, y^1, ..., y^p\}$ is said to be Λ-poised in B (in interpolation sense) if and only if*

1. *for the basis of Lagrange polynomials associated with Y*

$$\Lambda \geq \max_{0 \leq i \leq p} \max_{x \in B} |l_i(x)|$$

or, equivalently,

2. *replacing any point in Y by any $x \in B$ can increase the volume of the set $\{\phi(y^0), \phi(y^1), ..., \phi(y^p)\}$ at most by a factor Λ*

Value $\max_{0 \leq i \leq p} \max_{x \in B} |l_i(x)|$ is called Poisedness for the given set of points. Smaller value of poisedness represents better distribution of points.

NOWPAC uses Lagrange polynomials to calculate and improve poisedness of the set of points. Algorithm 2 is used for improving poisedness.

We can see from Figure 2.2 that points gets properly spread in just four steps of Algorithm 2. We start with points clustered at corners with $\Lambda = 5324$ and end up with well-distribute points with $\Lambda = 1.11$.

Algorithm 2: Improve Poisedness

Data: Given set of points Y, corresponding Lagrange Polynomials l_i, lower limit of poisedness(Λ_{min})

Result: set of well poised points

1 Calculate $\Lambda = \max_{0 \leq i \leq p} \max_{x \in B} |l_i(x)|$;

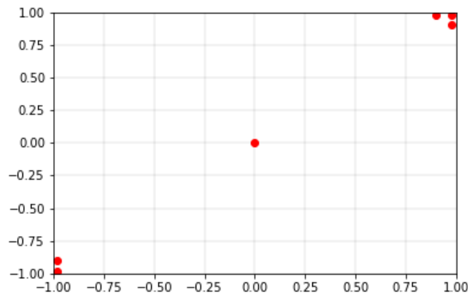
2 **while** $\Lambda \geq \Lambda_{min}$ **do**

3   Calculate $j_i = \arg\max_{0 \leq i \leq p} \arg\max_{x \in B} |l_i(x)|$;

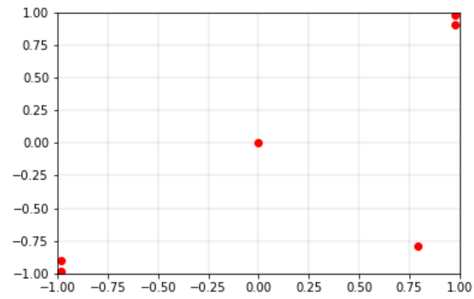
4   Replace y^i by y^{j_i} in the set Y;

5   Calculate $\Lambda = \max_{0 \leq i \leq p} \max_{x \in B} |l_i(x)|$;

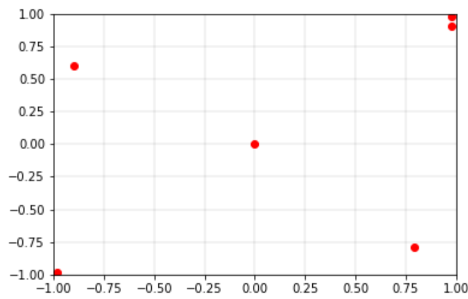
6 **end**



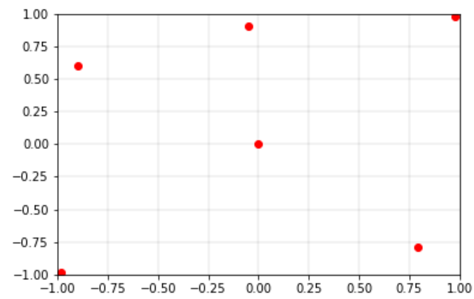
(a) $\Lambda = 5324$



(b) $\Lambda = 36.88$



(c) $\Lambda = 15.66$



(d) $\Lambda = 1.11$

Figure 2.2.: Four steps of Algorithm 2 starting with edge clustered points [34]

2.5. Parameter Default Values

In this chapter, we have introduced various parameters required by NOWPAC. Table 2.1 shows the default values of these parameters.

Parameter	Default Value
γ	0.8
γ_{inc}	1.4
ω	0.6
θ	0.5
η_0	0.1
η_1	0.7
μ	1.0
ϵ_c	10^{-6}
ϵ_b	10.0

Table 2.1.: List of default value for *SNOWPAC*

3. SNOWPAC

SNOWPAC (Stochastic Nonlinear Optimization With Path Augmented Constraints)[8] is an algorithm to optimize stochastic non-linear function subject to stochastic/non-stochastic non-linear constraints. This algorithm is an extension of *NOWPAC* [7]. The basic prototype of the problem statement is as following:

$$\begin{aligned} min \quad & f(x, \theta) \\ s.t. \quad & c_i(x, \theta) \leq 0, \quad i = 1...r \end{aligned} \tag{3.1}$$

where $x \in \mathbb{R}^n$ is a set of deterministic parameters and $\theta \in \mathbb{R}^m$ is a set of uncertain parameters. In other words our target is to find the value of deterministic parameters for which stochastic function generates minimum value. However, we have to take care of the variability of uncertain parameters θ into optimization. To solve this, we reformulate problem statement 3.1 by introducing *robustness measure*. The robustness measure does not contain the uncertain parameter but it is an approximation of the stochastic nature of the function. The new problem statement is:

$$\begin{aligned} min \quad & \mathcal{R}^f(x) \\ s.t. \quad & \mathcal{R}^{c_i}(x) \leq 0, \quad i = 1...r \end{aligned} \tag{3.2}$$

This chapter gives a concise explanation of *SNOWPAC* [8]. Firstly, we explain robustness measure which is the actual target function optimized in Section 3.1. One of the approaches to find value of robustness measure is to use large samples to obtain statistical moments. When each black-box evaluation takes a lot of time then this approach is infeasible. We can use small sample size to approximate the value. However, this leads to introduction of noise. *SNOWPAC* [8] uses Gaussian Process [93] to cancel effect of external noise. Therefore in Section 3.2, we explain Gaussian process and its application in *SNOWPAC* [8].

3.1. Robustness Measure Formulation

The robustness measure is the actual parameter optimized by *SNOWPAC* [8]. It gives a measure of the physical quantity by taking required statistical moment. To simplify the notation, we refer the objective function f and constraints c as black-box b and corresponding robustness measure as $\mathcal{R}^b$. Further, let us assume that b is square integrable with respect to θ, i.e. b has finite variance and for every design point $x \in \mathbb{R}^n$, cumulative distribution function is continuous and invertible.

All the robustness measures discussed in this thesis are coherent risk measures [5]. The risk measure can be thought of as a map from spaces of probability distributions to the real numbers [1]. In following discussions, we use the notation $\theta := (\theta_1, ..., \theta_m) : (\Omega, \mathcal{F}, \mathbb{P}) \rightarrow (\Theta, \mathcal{B}(\Theta), \mu)$ for uncertain parameters mapping from probability space $(\Omega, \mathcal{F}, \mathbb{P})$ to $(\Theta, \mathcal{B}(\Theta), \mu)$, $\Theta \subseteq \mathbb{R}^m$ [8].

3.1.1. Robustness Measure

There are a variety of options available for the robustness measure in literature [1, 5, 71, 98, 99, 100, 108]. Table 3.1.1, gives the a comprehensive list of commonly used robustness measures.

Robustness measure	Integrand B(x)
$\mathcal{R}_0^b$	$b(x,\theta)$
$\mathcal{R}_1^b$	$(b(x,\theta) - \mathcal{R}_0(x))^2$
$\mathcal{R}_2^b$	$\gamma c_0 b(x,\theta) + (1-\gamma)c_1(b(x,\theta) - \mathcal{R}_0(x))^2$
$\mathcal{R}_4^{b,\beta}$ [91]	$\mathbb{1}(b(x,\theta) \geq 0) - (1-\beta)$
$\mathcal{R}_1^{b,\beta}$ [1, 99]	$\gamma + \frac{1}{1-\beta}[b(x,\theta) - \gamma]^+$

Table 3.1.: List of Robustness Measure and corresponding integrands

In the Table 3.1.1, $b(x,\theta)$ represents the stochastic black-box.

3.1.2. Statistical Estimation

All the robustness measures mentioned in Section 3.1.1 can be written in term of expectation

$$\mathcal{R}^b(x) := \mathbb{E}_\theta[B(x,\theta)] \tag{3.3}$$

where function is defined in Table 3.1.1

To approximate Equation 3.3, we can use sample average E_N based on N samples that follows our required distribution of uncertain parameter. Approximation can be mathematically represented as:

$$\mathbb{E}_\theta[B(x,\theta)] = E_N[B(x,\theta)] + \varepsilon_x = \frac{1}{N}\sum_{i=1}^{N} B(x,\theta_i) + \varepsilon_x. \tag{3.4}$$

where, ε_x represents the error because of sample approximation. According to Central Limit Theorem, the means of a random sample of size N, from a population with mean μ and variance σ^2, distribute normally with mean μ and variance $\frac{\sigma^2}{\sqrt{N}}$ [72]. In other words, error term ε_x in Equation 3.4 is inversely proportional to $\sqrt{N}$ and approaches zero when $N \to \infty$. We want $\mathbb{E}_\theta[B(x,\theta)]$ to fall within a confidence level approximated by $E_N[B(x,\theta)]$ with high probability. To achieve confidence interval of $[E_N[B(x,\theta)] - \bar{\varepsilon}_x, E_N[B(x,\theta)] + \bar{\varepsilon}_x]$ with probability greater than $\nu \in (0,1)$, we can compute sample standard deviation $s_N(x)$ as:

$$s_N(x)^2 = \frac{1}{N-1}\sum_{i=1}^{N}(B(x,\theta_i) - E_N[B(x,\theta)])^2, \tag{3.5}$$

with

$$\bar{\varepsilon}_x = \frac{t_{N-1,\nu}\, s_N(x)}{\sqrt{(N)}}, \tag{3.6}$$

where $t_{N-1,\nu}$ is the ν-quantile of Student-t distribution. In this thesis, we select $t_{N-1,\nu} = 2$ which gives confidence interval exceeding 0.975 for sample size $N \geq 60$.

3.2. Gaussian Process Supported Trust Region Management

SNOWPAC [8] uses robustness measure to optimize problem mentioned in Equation 3.1. Therefore, accurate approximation of robustness measure is crucial to the performance. As mentioned in Section 3.1.1, we can use sample approximation to estimate robustness measure. When sample size is large then approximation is accurate. But this approach increases the computational requirement. In this section we discuss the method to use Gaussian Process to improve robustness approximation and reduce noise in black-box evaluation. Firstly, we introduce Gaussian process and regression. Then, brief explanation of hyperparameters of gaussian process and method to tune it. Finally, we discuss its application to SNOWPAC. Main sources referred are [8, 93].

3.2.1. Introduction to Gaussian Process

Gaussian process is a statistical model that predicts the value of function for the given continuous domain. It can be interpreted as multidimensional gaussian distribution with every point representing each dimension. In other words every, function value at every point of design space is normally distributed with some mean and standard deviation. Thus, we can say Gaussian process is infinite dimensional multivariate gaussian distribution.

As mentioned before Gaussian process can be described by its mean $m(x)$ and covariance k :

$$\begin{aligned} m(x) &= \mathbb{E}[f(x)] \\ k(x,x') &= \mathbb{E}[(f(x)-m(x))(f(x')-m(x'))] \end{aligned} \tag{3.7}$$

We can write a random function f following gaussian process using notation:

$$f(x) \sim \mathcal{G}(m(x), k(x,x')) \tag{3.8}$$

Covariance between two points is represented using *kernel function*. Kernel function is heart of any gaussian process. There are a variety of kernels mentioned in literature [36, 37, 51].

Only symmetric kernel functions can be used to express a covariance function. A covariance function is symmetric by definition. Also, kernel matrix formed by calculating covariances between individual points in the domain must be *positive semidefinite*. Such kernels are called *Mercers Kernels*. A kernel is positive semidefinite if

$$\int k(x,x')f(x)f(x')d\mu(x)d\mu(x') \geq 0, \tag{3.9}$$

Kernel matrix formed is referred as *Gram Matrix* [93, 21, 96]. For sake of simplicity in notation let us represent Gram Matrix as K.

Gaussian process is extensively used for regression. To find regression-based prediction for a test point x^*, conditional distribution is computed on the test output given the training data and the test input. This is Gaussian distribution $p(y^*|X,Y,x^*) = \mathcal{N}(\mu^*, \Sigma^*)$ with predictive mean and covariance given by

$$\begin{aligned} \mu^* &= k^{*T}(K+\sigma^2 I)^{-1}y, \\ \Sigma^* &= k(x^*,x^*) - k^{*T}(K+\sigma^2 I)^{-1}k^* + \sigma^2 I \end{aligned} \tag{3.10}$$

where $k^* = [k(x^*, x_1), k(x^*, x_2), ..., k(x^*, x_n)]$.

GP regression as mentioned in Equation 3.10, is used only for small datasets. The time complexity is $O(n^3)$ because there is inversion of Gram matrix. Various other approximate methods have been developed [17, 18, 27, 29, 49, 92, 73].

Hyperparameter learning is done by maximizing marginal log-likelihood:

$$p(y|X, \theta) = -\frac{1}{2}\log|K + \sigma^2 I| - \frac{1}{2}y^T(K + \sigma^2 I)^{-1}y - \frac{n}{2}\log 2\pi \tag{3.11}$$

where I is the identity matrix of same dimension as K and σ is the standard deviation of additive gaussian noise.

3.2.2. Gaussian Process Surrogates

As mentioned in Section 3.2 , Gaussian process not only provides a way to smoothen out noisy evaluations but also gives an estimation of robustness measure credibility. Therefore, SNOWPAC [8] uses GP surrogates as measure to balance out the error introduced to small sample size N. We build surrogates in same way as mention in Chapter 2. But now instead of just using the value of black-box evaluation, we use weighted mean of black-box evaluation and GP surrogate as following:

$$\begin{aligned} \hat{\mathcal{R}}^b_{k,i} &= \gamma^i_s \mathcal{G}^k_b(x^{(i)}_k) + (1 - \gamma^i_s)\mathcal{R}^b_i \\ \hat{\varepsilon}^b_{k,i} &= \gamma^i_s t_{N-1,\beta}\sigma_b(x^{(i)}_k) + (1 - \gamma^i_s)\varepsilon^b_i \end{aligned} \tag{3.12}$$

where $\sigma_b(x^{(i)}_k)$ denotes the standard deviation of $\mathcal{G}^k_b(x^{(i)}_k)$ at point $x^{(i)}_k$. The weight factor is calculated as following:

$$\gamma^i_s := \exp(-\sigma_b(x^{(i)}_k)) \tag{3.13}$$

There is decrease in the value of σ_b when more observation points are added in GP. We have fewer data points during the initial steps. Our GP surrogate is inaccurate. Therefore, GP surrogate should have low weight in the beginning. But, as we move forward in optimization new points are observed and our GP surrogate improves. So, weight of GP surrogate must increase. Equation 3.13 follows this trend.

3.2.3. Noise Adapted Trust Region

Similar to deterministic case where we have fully linear model satisfying error bound given by Equation 2.6, SNOWPAC also builds fully linear surrogate model using noise corrupted black-box evaluations satisfying:

$$\begin{aligned} \left\|\mathcal{R}^b(x_k + s) - m^{\mathcal{R}^b}_k(x_k + s)\right\| &\leq \kappa_1\rho^2_k \\ \left\|\nabla\mathcal{R}^b(x_k + s) - \nabla m^{\mathcal{R}^b}_k(x_k + s)\right\| &\leq \kappa_2\rho^2_k \end{aligned} \tag{3.14}$$

with high probability α for constants [65]

$$\kappa_i = \kappa_i(\bar{\varepsilon}^k_{max}\rho^{-2}_k), \qquad i \in \{1, 2\}. \tag{3.15}$$

The constants κ_1 and κ_2 depends on the poisedness constant $\Lambda > 1$ as well as on the estimates of the statistical upper bounds for the noise term $\bar{\varepsilon}^k_{max} = \max_{i=1,..,n} \bar{\varepsilon}_i$, where ε is from Equation 3.4. Due to presence of positive noise, the term $\bar{\varepsilon}^k_{max}\rho_k^{-2}$ increases with decrease in the radius of trust region. Thus κ_1 and κ_2 has unbounded growth when trust-region radius shrinks, thereby violation linearity conditions. To ensure full linearity condition of the surrogate model, we have to put an upper bound on the error term $\bar{\varepsilon}^k_{max}\rho_k^{-2}$. This is equivalent to putting lower bound on trust-region radius ρ_k as following:

$$\bar{\varepsilon}^k_{max}\rho_k^{-2} \leq \lambda_t^{-2}, \quad \text{resp.} \qquad \rho_k \geq \lambda_t \sqrt{\bar{\varepsilon}^k_{max}}, \tag{3.16}$$

where, $\lambda_t \in (0, \infty)$.

This restriction will prevent the radius from converging to zero. However, from Equation 3.12 we can see that as weight of GP surrogate increases, noise term decreases. In this way, GP helps in convergence of trust-region radius to zero.

4. Mixed Integers

In various design problems, there are parameters which are constrained to be integers. The category of problems in which some parameters are integers whereas others are real numbers are called the mixed integer problems. Till now, we have only discussed optimization problem where all the parameters are real numbers. The mixed-integer problem statement can be mathematically represented as:

$$\begin{aligned} \min_{x} \quad & f(x), \\ s.t. \quad & c_i(x) \leq 0 \qquad i = 1, 2, ..., r, \\ & l \leq x \leq u, \\ & x = [x_c^T, x_d^T]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}. \end{aligned} \tag{4.1}$$

Various heuristic approaches exists where one can round off the real numbers to the nearest integer [2, 14]. This thesis uses the trust-region based derivative free algorithm for the mixed-integer optimization problem. The techniques discussed in this chapter are developed and mentioned in various literatures [85, 45, 76, 77].

We discuss the algorithm that converges to a local minima. This chapter starts with the definition of local minima of a mixed-integer problems in Section 4.1. In this thesis, we modify SNOWPAC [8] to handle mixed integer problem. The first step to implement mixed-integer part of SNOWPAC was to handle box constrained optimization problems. There are no constraints $c_i(x)^s$ except the upper and lower bounds of design variables. Such problems are stated as:

$$\begin{aligned} \min_{x} \quad & f(x), \\ s.t. \quad & l \leq x \leq u, \\ & x = [x_c^T, x_d^T]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}. \end{aligned} \tag{4.2}$$

We discuss the modifications done in the existing code in Section 4.2. There were two major changes, namely the shape of trust regions and the algorithm for local optimization of the surrogate model. Firstly, we discuss the motivation behind changing the shape of the trust-region from closed ball to closed box. Then, we give a description of the Branch and Bound algorithm is used for optimizing the surrogate model. We provide an overview of other minor modifications. Lastly, we discuss the method in which all the aforementioned techniques are integrated with SNOWPAC in Section 4.2.

4.1. Local minima of mixed integer optimization problems

There are several definitions of local minima of a mixed integer problem. In this section we will just discuss one of them, followed by an example. Before proceeding to the formal

definitions we will mention the notation that is followed. Let Ω_m denote the feasible region of problem 4.1. For special cases of $n_d = 0$ and $n_c = 0$, feasible domains are represented by Ω_c and Ω_d respectively. For any point x_c and $\varepsilon > 0$, let $B_\varepsilon(x_c)$ is notation of an open ball defined as $\{y_c \in \mathbb{R}^{n_c} : \|y_c - x_c\| < \varepsilon\}$. Let $\mathcal{N}_d(x_d)$ and $\mathcal{N}_m$ denote the user defined neighborhood for discrete dimensions and problem statement respectively. Let $\mathcal{N}_r$ represent a neighborhood of x as:

$$\mathcal{N}_r(x) = \{y \in \mathbb{R}^n : y_c = x_c, y_d \in \mathcal{N}_d(x_d)\}.$$

$\mathcal{N}_r \subseteq \mathcal{N}_m$ and for unconstrained box-problems 4.2, both of them are same. For some feasible value of the discrete points x_d, the remaining degree of freedoms in continuous dimension forms a manifold referred as *feasible continuous manifold* represented as $f_\mathcal{M}(y)$ and formally defined as:

$$f_\mathcal{M}(y) = \{f([y^T, x_d^T]^T) : [y^T, x_d^T]^T \in \Omega_m\}$$

feasible continuous manifold depends on the value of x_d. In the box constrained problems, there are finite number of values that x_d can hold. This leads to finite feasible continuous domain.

Before going further let us formally define continuous local minimum($n_d = 0$), discrete local minimum ($n_d = 0$) and global minimum.

Definition 4.1 (*Continuous local minimum* **[86]**) *A point $x^* \in \Omega_c$ is a local minimum if for some $\varepsilon > 0$,*

$$f(x^*) \leq f(x), \qquad \forall x \in \Omega_m \cap B_\varepsilon(x^*).$$

Definition 4.2 (*Discrete local minimum* **[83]**) *A point $x^* \in \Omega_d$ is a local minimum if,*

$$f(x^*) \leq f(x), \qquad \forall x \in \mathcal{N}_d(x^*).$$

Definition 4.3 (*Global minimum* **[86]**) *A point $x^* \in \Omega_m$ is a local minimum if,*

$$f(x^*) \leq f(x), \qquad \forall x \in \Omega_m.$$

Mixed-integer optimization problem is a $\mathcal{NP}$-hard [85], so definition for mixed integer problem should allow some control over the size of neighborhood. For a convex problem, a local minimum is a global minimum. Also, the definition should be valid under special cases, for example when all the variables are continuous etc. This puts additional constraints on the definition of mixed integer local minimum:

1. The definition should be valid even when $n_d = 0$
2. The definition should be valid even when $n_c = 0$
3. If $\mathcal{N}_m$ contains at least one point on each feasible continuous manifold and objective function and constraints are convex, then a point is a local minimum of mixed integer problem if and only if it is a global minimum.

The definition of *separate local minimum* given by [76] does not satisfy the third condition. So, in this thesis we follow the *combined local minimum* [85]

Definition 4.4 (*Separate local minimum*) *A point $x^* \in \Omega_m$ is a separate local minimum of problem 4.1 if for some $\varepsilon > 0$,*

$$\begin{aligned} f(x^*) &\le f(x), \qquad \forall x \in \{x : x_c \in B_\varepsilon(x_c^*), x_d = x_d^*\} \cap \Omega_m \\ f(x^*) &\le f(x), \qquad \forall x \in \mathcal{N}_r(x^*) \cap \Omega_m \end{aligned}$$

where $\mathcal{N}_{comb}(x^*)$ is the set of the smallest local minima on each feasible continuous manifold on which $\mathcal{N}_r(x^*)$ has a point.

Definition 4.5 (*Combined local minimum*) *A point $x^* \in \Omega_m$ is a combined local minimum of problem 4.1 if for some $\varepsilon > 0$,*

$$\begin{aligned} f(x^*) &\le f(x), \qquad \forall x \in \{x : x_c \in B_\varepsilon(x_c^*), x_d = x_d^*\} \cap \Omega_m \\ f(x^*) &\le f(x), \qquad \forall x \in \mathcal{N}_{comb}(x^*) \cup \mathcal{N}_r(x^*) \end{aligned}$$

where $\mathcal{N}_{comb}(x^*)$ is the set of the smallest local minima on each feasible continuous manifold on which $\mathcal{N}_r(x^*)$ has a point.

We will now formally define $\mathcal{N}_{comb}(x^*)$. For that let us define $\mathcal{A}(\bar{x})$ as:

$$\mathcal{A}(\bar{x}) = \{\bar{x} : \bar{x}_d = \tilde{x}_d, f(\bar{x}) \le f(x) \quad \forall x \in \{x : x_c \in B_\varepsilon(\bar{x}_c), x_d = \tilde{x}_d\} \cap \Omega_m\}$$

$\mathcal{N}_{comb}(x^*)$ is then given by

$$\mathcal{N}_{comb}(x^*) = \left\{ \arg\min_y [f(y) \quad s.t. \quad y \in \mathcal{A}(\tilde{x})] : \tilde{x} \in \mathcal{N}_r(x^*) \setminus \{x^*\} \right\} \tag{4.3}$$

We will explain Definition 4.5 using an example. Let us consider following mixed integer optimization problem [85]:

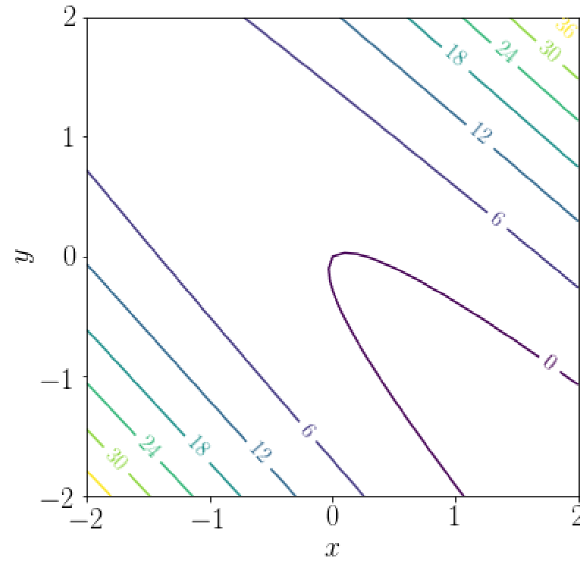
$$\begin{aligned} \min_{[y,x]} \quad & \frac{5}{2}(x+y)^2 + \frac{1}{\sqrt{2}}(y-x) \\ s.t. \quad & -2 \le x, y \le 2, \\ & y \in \mathbb{R}, \quad x \in \mathbb{Z} \end{aligned} \tag{4.4}$$

It is a simple box-constraint problem, which is convex. The objective function is also convex because it is a sum of two convex terms. Figure 4.1(a) and Figure 4.1(b) shows contours and feasible continuous manifold of objective function respectively. Red dots in Figure 4.1(b) shows minimum value of each manifold. Each red dot is a *separate local minimum*. By Definition 4.5, $\mathcal{N}_{comb}$ contains these five dot points. The *combined local-minimum* of the mixed integer problem is the smallest value amongst these points. As seen from figure 4.1, the combined local minimum of problem 4.4 is $x = 2$ and $y = -2$.

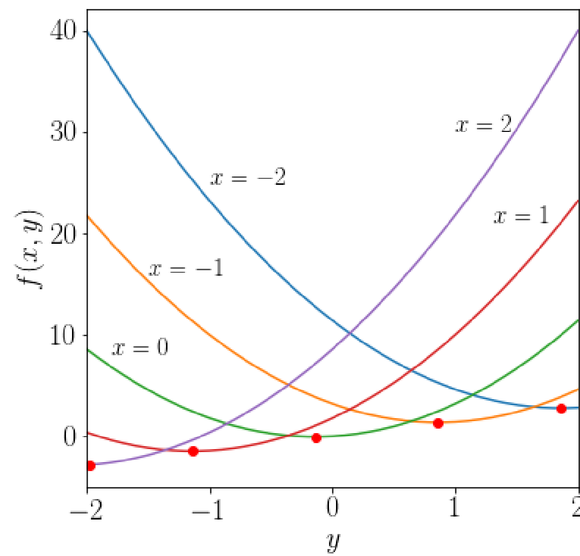
4.2. Modifications for Mixed Integers

4.2.1. Trust Region Shape

In NOWPAC [7] and SNOWPAC [8], the trust-region is in shape of a ball represented as $B(x_k, \rho_k) = \{x \in \mathbb{R}^n : \|x = x_k\| \le \rho_k\}$, where x_k is the center and ρ_k is trust-region radius. For the mixed integer problems we have two types of design parameters, namely



(a) Contour plot of problem 4.4



(b) Value of function for every discrete dimension x for over continous dimension y

Figure 4.1.: Figure illustrating features of problem 4.4. Red dots in Figure (b) represent minimum point for every value of discrete dimension

integer parameters and continuous parameters. Smallest change in the direction of integer parameter is one. Suppose, the radius of trust region(ρ_k) falls below 1. In such scenarios, the value of integer parameters cannot change. Thereafter, optimizer stops exploring other integer values and gets stuck at one integer parameter. In such scenarios, optimizer converges to a *separate minimum*. If we want to converge to a *local minimum*, then we cannot use the trust region as used in NOWPAC and SNOWPAC.

One of the solutions to the aforementioned problem is restricting the radius of trust-region radius to one. If we do this, behavior of the surrogate model may not improve properly along continuous parameter. So, we will not be able to obtain the accurate optimized point specially for continuous part.

The solution that we have adopted in this work is to modify the shape of trust-region from ball to box. The trust-region $B(x_k, \rho_k)$ is now defined as:

$$B(x_k, \rho_k) = \{x \in \mathbb{R}^n : |x - x_k| \leq \rho_k\} \tag{4.5}$$

where $\rho_k \in \mathbb{R}^n$, is a n-dimensional vector with each dimension representing the size of box in each parameter dimension. Lower bound of the integer dimension is one.

The upper and the lower bound of the design parameters also put restriction on the size of the box. The box-size in any given dimension must not exceed the difference between the upper, and the lower bound(or the spread of a dimension). If size of trust-region exceeds the spread in any dimension, then we restrict the box-size in that dimension to the spread in that dimension. In other words, spread of the variable in any given dimension is the upper bound to the trust region size.

We summarize all the aforementioned restriction and formally define ρ_k as:

$$\begin{aligned} \rho_k &= [\rho^{\mathbb{Z}^T}, \rho^{\mathbb{R}^T}]^T \in \mathbb{R}^{n_d} \times \mathbb{R}^{n_c} \\ \textit{where } \rho_i^{\mathbb{Z}} &\in [1, u_i - l_i) \forall i = 1, ..., n_d \\ \textit{and } \rho_j^{\mathbb{R}} &\in (0, u_j - l_j) \forall j = 1, ..., n_c \end{aligned} \tag{4.6}$$

Here n is total number of design parameters, n_d is number of integer design parameters and n_c is number of continuous design parameters.

4.2.2. Surrogate Model Optimization

As stated earlier in Chapter 2, the basic approach of trust-region derivative free method is to build a surrogate model in a local *trust region* and optimize the surrogate instead of optimizing the actual function. For solving the optimization problems as stated in Equation 4.1, the surrogate optimization subproblem is stated as following:

$$\begin{aligned} \bar{x_k} = &\underset{x \in B(x_k, \rho_k)}{\arg\min} \; m_k^f(x) \\ s.t. \quad & m_k^{c_i}(x) \leq 0 \qquad i = 1, 2, ..., r, \\ & l \leq x \leq u, \\ & x = [x_c^T, x_d^T]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}. \end{aligned} \tag{4.7}$$

Similarly, the surrogate optimization subproblem for the family of problems represented by Equation 4.2 is as:

$$\begin{aligned} \bar{x_k} = & \underset{x \in B(x_k, \rho_k)}{\arg\min} \; m_k^f(x) \\ s.t \quad & l \leq x \leq u, \\ & x = [x_c^T, x_d^T]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}. \end{aligned} \tag{4.8}$$

As mentioned in Chapter 2, the highest order of the surrogate model of both the objective function(m_k^f) and the constraints($m_k^{c_i}$) is quadratic. So, the optimization problems of form 4.8 and 4.7 belongs to the family of *Mixed Integer Quadratic Problems (MIQP)* and *Mixed Integer Quadratic Constrained Problems (MIQCP)* respectively.

There are various approaches to solve such family of problems. Detailed study of these problems is done in [16, 19, 9, 20]. The method to be applied depends upon the Hessian of the surrogate model. If the hessian is positive definite, we can use the commercial software solver CPLEX [23]. If the hessian is indefinite and the ${n_c}^{th}$ principle leading sub-matrix is positive semidefinite then we can solve using the convex reformulation scheme described in [20]. If none of the aforementioned conditions are satisfied, then we can use general *Branch and Bound* algorithm to solve the problem as mentioned in [84].

In this work, we have only used branch and bound the algorithm to optimize the surrogate model. We describe the branch and bound the algorithm for box-constrained problems. Thereafter, we elaborate the new scheme to update trust-region.

4.2.3. Branch and Bound

The branch and bound is one of the most commonly used approaches applied to solve $\mathcal{NP}$-hard combinatorial problems. It can be applied to solve various categories of problems like the Traveling Salesman problem, the Graph Partitioning problem, the Quadratic Assignment problem etc [31]. For the first time, it was used to solve Mixed Integer Linear Programming problems in [40]. Later, it was extended to non-linear problems in [57, 24, 106].

We describe the steps to use the branch and bound algorithm for mixed integer problems in Algorithm 3. We start by describing the method with box-constrained problem 4.8. The basic idea behind the branch and bound algorithms is to use the continuous relaxation of the problem 4.8. Thereby, we obtain the valid lower and upper bounds and explore the space of integer variables using tree search.

The branch and bound algorithm is implemented in recursive manner by continuously building a tree. We recursively divide the domain into smaller sub-domains. Each subdomain represents a different sub-problem. We represent each sub-problem using one node of the tree.

Each node of the tree is formed by relaxing the integer constraint of the problem 4.8 or 4.7. Now, each sub-problem is a simple optimization problem. We can obtain a unique solution if the new sub-problem is convex. As stated in Chapter 2, path augmented constraints convexify the constraints. Sub-problem optimization is done using *COBYLA* [87, 89] method as provided by open source software *NLOPT* [64].

Let us consider a schematic diagram 4.2. We initialize the value of function minimum (f^*) with a very large number. We start our algorithm with the original lower and upper bounds. It is shown in 4.2(a) and represent it by S. We relax the integer constraints

and solve the sub-problem as mentioned earlier. We get minimum ($\bar{f}$) with parameters $\bar{x}$. Suppose, we do not obtain integral solutions for the required set of variables, then, we divide the domain into two sub-parts thereby creating two new sub-problems as represented in 4.2(b) as S_1 and S_2. To find the criterion of division, we find the integer variable in $\bar{x}_j \quad j = 1, ..., n_d$ which has the largest absolute decimal value as following:

$$l = \underset{0 \leq j \leq n_d}{\arg\max} \quad (|\bar{x}_j| - floor(|\bar{x}_j|))$$

S_1 will have same upper and lower bound as S with one modification. The lth component of upper bound will be the nearest and smallest integer of $\bar{x}_l$. Similarly, S_2 will have same upper and lower bound as S with one modification. The lth component of lower bound will be the nearest and biggest integer of $\bar{x}_l$. We relax the integer criterion and solve the subproblem again. In this way, the binary tree is built recursively as shown in 4.2(c), 4.2(d) and so on. At every branch, we check if the minimum of the sub-problem($\bar{f}$) is lower than previous minimums (f^*) and if all the integer variables are integers i.e. $\bar{x}_j \in \mathbb{Z} \quad \forall \quad j = 1, ..., n_d$. If this criterion is satisfied, then we update the value of (f^*) and optimal parameters as $x^* = \bar{x}$. By the end of this algorithm, values f^* and x^* will be the mixed integer minimum value and the corresponding parameter respectively of the optimization problem.

We can prune the tree and make the algorithm more efficient by including the concept of *Fathomed Branch*. Suppose, we keep building a tree and reach a branch say S_{112}, whose minimum($\bar{f}$) is greater than already existing function minimum f^*. So the given branch cannot provide a value better than the existing minimum. We call such a branch as *fath-*

omed. We stop building the tree any further.

Algorithm 3: Branch and Bound for mixed integer problems

Data: initial upper and lower bound(ub, lb), function(m_k^f), constraints($m_k^{c_i}$), list of discrete and continuous variable
Result: Optimum Point(x^*), Function minimum value(f^*)

```
1  Set f* = ∞;
2  Function BranchAndBound(lb, ub) : is
3      Create a sub-problem(branch) similar to 4.8 or 4.7 but with bounds lb and ub ;
4      Relax integer criterion;
5      Solve x̄ = min_{m_k^{c_i} ≤ 0} m_k^f(x);
6      Set local minimum of the branch(f̄) as f̄ = f(x̄) ;
7      if f̄ > f* then
8          Branch is Fathomed ;
9          return;
10     end
11     else if x̄_j ∈ ℤ   ∀   j = 1, ..., n_d then
12         Set f* = f̄ ;
13         Set x* = x̄ ;
14         return ;
15     end
16     Select l = arg max_{0≤j≤n_d} (|x̄_j| − floor(|x̄_j|));
17     Set l̃b = lb and ũb = ub;
18     Set l̃b_l = ciel(x̄_l) ;
19     Call BranchAndBound(l̃b_l, ũb) ;
20     Set ũb_l = floor(x̄_l) ;
21     Call BranchAndBound(l̃b, ũb_l) ;
22     return;
23 end
```

4.2.4. Trust Region Update

The trust-region update procedure is similar to that of NOWPAC [7] as described in Section 2.1. We just make one modification. In NOWPAC [7], if the step is rejected, we shrink the trust-region by a factor of γ where $\gamma \in (0, 1)$. When we shrink the trust-region, none of the outliers are used to build the surrogate model. If the number of points are not enough to build a fully linear model (i.e. number of active points is less than $n + 1$, where n is the dimension of design space), then we sample a few points inside the trust-region to ensure that there are enough active points to build a fully linear model. We are not able to build a higher order model because every time we shrink the trust-region, few points may become outliers. This is a bigger issue when each black-box evaluation is expensive. In other words, this was very inefficient because we spent a lot of computational effort to evaluate the value of the black-box function at a point and it goes quickly out of the

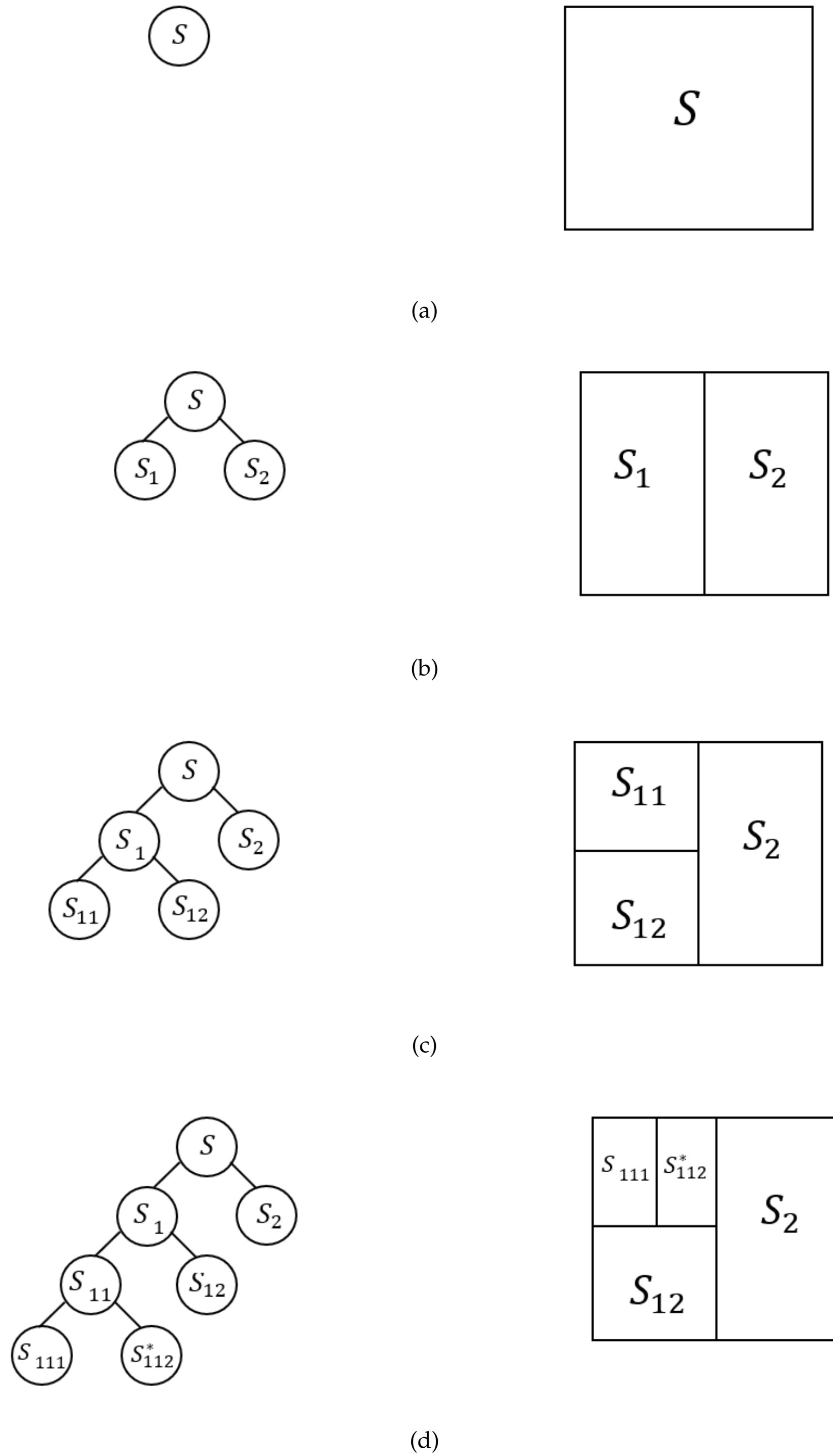


Figure 4.2.: Illustrative tree

trust-region.

We could solve this problem by increasing the value of γ. However, this slows down the convergence. So, we needed to find a way to ensure that we are able to use more points inside the trust-region to make it closer to a quadratic model and also the convergence speed is not adversely affected.

We did that by putting extra constraints on the step rejection criterion. If the step is rejected, then we decrease the trust-region radius by a factor of γ only if the number of points inside the trust-region is greater that $2n+1$, where n is the dimension of the design space. The new trust-region update scheme is summarized as:

$$\rho_k = \begin{cases} \gamma_{inc}\rho_k & \text{if} \quad r_k \geq \eta_1 \\ \rho_k & \text{if} \quad \eta_0 \leq r_k \leq \eta_1 \\ \gamma_{shrink}\rho_k & \text{if} \quad 0 < r_k < \eta_0 \\ \gamma\rho_k & \text{if} \quad r_k \leq 0 \quad \text{and} \quad n_{active} > (2n+1) \end{cases} \tag{4.9}$$

where n_{active} is the number of points that are inside the trust-region.

4.3. Adapting SNOWPAC for Mixed Integer problems

To use the *SNOWPAC* package, we have to create a constructor. We modified the constructor to include the mixed integer features. The new constructor has the following structure:

```
NOWPAC ( int n, int numint, std::vector<double> scaling);
```

Here, `n` represents the dimension of the optimization problem, `numint` represents the number of integer variables and `scaling` a user-defined `n`-dimensional vector that stores scaling along all parameters. If `numint` is greater than zero, then the problem becomes a mixed integer optimization problem. Otherwise, it is treated as a general optimization problem. SNOWPAC provides an interface for users to input their *black-box function*. The interface is as following:

```
virtual void evaluate (std::vector<double> const &x,
                       std::vector<double> &vals,
                       void *<user parameter>);
```

The user must ensure that the first `numint` variables are supposed to be integer variables, rest are continuous variables.

We added a new class to the existing code `BranchAndBound`. We also modified the existing code to accommodate the algorithm changes mentioned in this chapter.

5. Artificial Neural Network

This chapter introduces the basics of artificial neural networks. We start by discussing the mathematical model of a *feed forward neural network* in Section 5.1. This section discusses the structure and building blocks of a simple *Multi-Layer Perceptron (MLP)*. In Section 5.2 we discuss the data driven supervised training methods. Thereafter, we take an overview of a *Convolutional Neural Network (CNN)* in Section 5.3. Then, in Section 5.4 we discuss the regularization techniques that are used to prevent overfitting. Here, we discuss about the loss function and the stochastic gradient method.

5.1. Feed forward Neural Network

As the name suggests *artificial neurons* are inspired by brain cells. Brain cells have multiple input signals and produce a single output. Output is the weighted sum of inputs over which an *activation function* act. Similar structure is followed in artificial neurons.

We mathematically state the structure of a single artificial neuron. In the further discussions, we follow the mathematical notation and formulas used in [54]. A single artificial neuron can be represented as a non-linear function $g : \mathbb{R}^K \to \mathbb{R}$, with K input values and parameters are a *weight vector* w, a *bias* b and a non-linear *activation function* σ:

$$g(x) = \sigma\Big(\sum_{k=0}^{K} w_k x_k + b\Big) = \sigma(w^T x + b) \tag{5.1}$$

Figure 5.1(b) illustrates an artificial neuron.

We consider a set of neurons arranged in form of layers, where a layer l feeds its output only into the next layer $l + 1$. This is called the *feed forward property* and this network of multiple layers is called *feed forward neural network*. We can also visualize the neural network as a collection of neurons that are connected in an acyclic graph. Such networks are also referred as *"Artificial Neural Network (ANN)"*.

A layer l with $K^{(l)}$ neurons which operate on an input vector $x^{(l-1)}$, represents a non-linear function $f^{(l)} : \mathbb{R}^{K^{(l-1)}} \to \mathbb{R}^{K^{(l)}}$, that produces an output vector $x^{(l)}$:

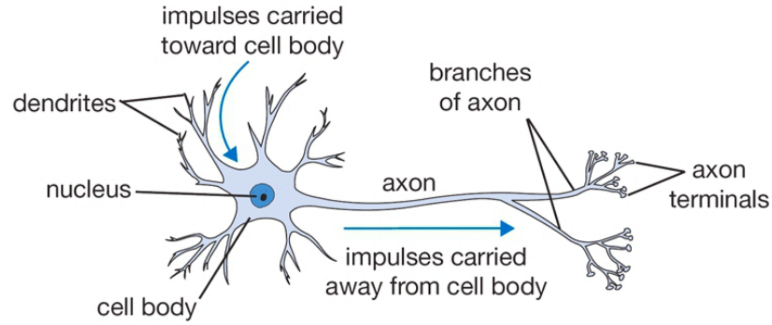
$$x^{(l)} = f^{(l)}(x^{(l-1)}) \tag{5.2}$$

A layer function at layer l is defined as:

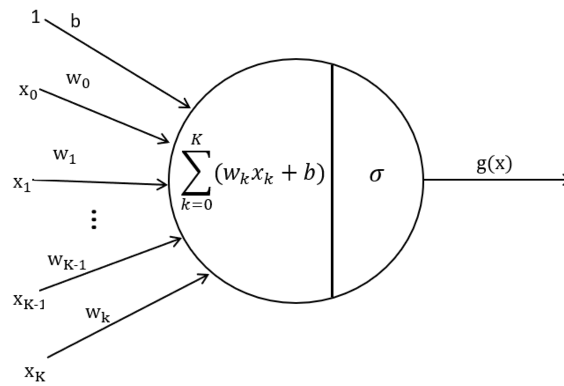
$$f^{(l)}(x) = \sigma^{(l)}(\mathbf{W}^{(l)} x + \mathbf{b}^{(l)}) \tag{5.3}$$

where the *weight matrix* $\mathbf{W}^{(l)}$ and the *bias vector* $\mathbf{b}^{(l)}$ are constructed by weight vectors and biases of individual neurons.

Let us consider a network with L consecutive layers and overall *network parameters* $\boldsymbol{\theta}$. Overall network parameter $\boldsymbol{\theta}$ is a collection of all individual layer parameters. When we



(a) Single brain cell [66]



(b) Artificial neuron as stated in Equation 5.1

Figure 5.1.: Analogy between brain cells and artificial neurons.

explicitly specify the overall network parameter $\boldsymbol{\theta}$, the network is thereby represented as a function $\mathbf{y} = f(x; \boldsymbol{\theta})$. The given function is written as decomposition of the individual layer functions $f^{(l)}$ as $f : \mathbb{R}^{K^{(0)}} \rightarrow \mathbb{R}^{K^{(L)}}$ as following:

$$f(x; \boldsymbol{\theta}) = (f^{(L)} * f^{(L-1)} * ... * f^{(1)})(x) \tag{5.4}$$

The number of neurons in each layer determines the *width* of a given layer. First layer and last layers are input and output layers respectively. Other layers sandwiched between the input layer and the output layer are known as *hidden layers*. The number of hidden layers determines the depth of MLP. If there are N hidden layers, then the network is called N-layer neural network [66]. When every neuron of a given layer is connected to every other neuron of next layer, then the network is called a *fully connected layer*. Figure 5.1 shows a fully connected neural network with two hidden layers. A "*Multi-Layer Perceptron(MLP)*" is a class of ANN which consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer.

The next important aspect is the activation function σ. An activation function is a non-linear function that takes single input and performs a specific mathematical operation on it. They introduce non-linear properties to our network. There are various types of activation

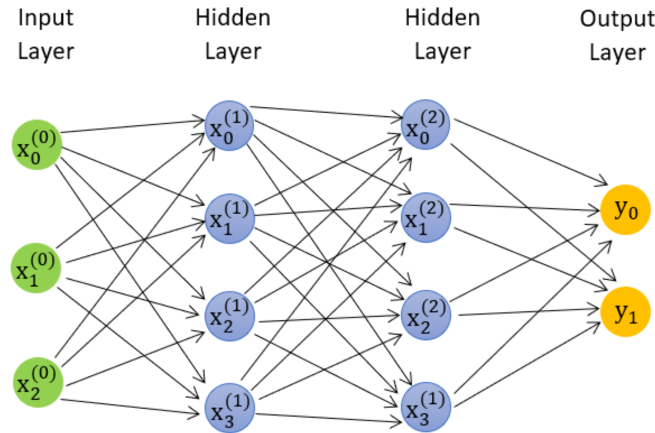
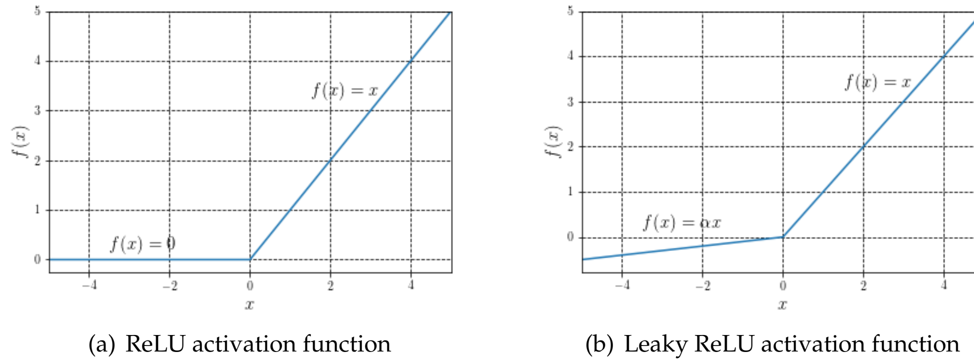


Figure 5.2.: A fully connected neural network with two hidden layers.



(a) ReLU activation function

(b) Leaky ReLU activation function

Figure 5.3.: ReLU and Leaky ReLU activation functions.

functions discussed in [54, 70, 58]. In this thesis, we have used the following activation functions:

- **ReLU :** The Rectified Linear Unit computes the function $f(x) = \max(0, x)$. It performs better than sigmoid/tanh activation functions [70] because it does not saturates at high input values. However, when input is negative then the neuron dies.
- **Leaky ReLU :** The problem of dying neurons is solved by leaky rectified linear unit by providing a small slope to the line in negative region. The function of leaky ReLU is

$$f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x \geq 0)(x)$$

where "α" is a small positive constant. Advantages of Leaky ReLU and its overall effects are studied in [58].

In basic structure of ANN, features like slope of leaky ReLU(α), width of a neural network etc are hyperparameters. In other words, before training of the neural network, we

need to take a wise guess about its values. A wrong guess can lead to improper training and bad accuracy.

5.2. Supervised Training

In the previous section we have discussed various building blocks of a neural network. Our assumption till now was that we knew the overall *network parameters* $\boldsymbol{\theta}$. In this section we take a cursory look of the methods used to obtain $\boldsymbol{\theta}$. This procedure is called the training of a neural network. When a labeled dataset is used to train a network then the process is known as *supervised learning*. In this section we discuss the supervised training to obtain the network parameters $\boldsymbol{\theta}$.

The goal of supervised learning is to learn a function $y^* = f^*(x)$ by approximating it with a neural network $y = f(x)$ [54]. The challenge here is the unknown true function f^*. We use the provided dataset which contains the labeled output y^* for some finite input x to approximate f^*.

Let us represent set of labeled data as $\mathcal{Z}$. It is divided into three sets namely *training* ($\mathcal{Z}_{train}$), *validation* ($\mathcal{Z}_{val}$) and *testing* ($\mathcal{Z}_{test}$) set. The first step is to initialize the weights and biases of the network with random numbers. The weight initialization of a network is an important step, and it has been studied in various literatures. Some of the commonly used techniques are Kaiming Initialization [58], Xavier Initialization [53] etc. In this thesis, we have done initialization with normally distributed random number. The mean and standard deviation of the normal distribution serves as hyperparameters. During training, the network predicts y for input x from training set. These predictions are then compared with corresponding labels y^* to generate a *loss function* $J(x, \boldsymbol{\theta}; \mathcal{Z}_{train})$ under current set of parameters $\boldsymbol{\theta}$. Our aim is to find the parameters $\boldsymbol{\theta}$ for which loss function is minimum for the training dataset. Formally it can be stated as:

$$\boldsymbol{\theta} = \arg\min_{\theta} J(x, \theta; \mathcal{Z}_{train}) \tag{5.5}$$

In this way we are trying to find a function f which is a good approximation of actual function f^*.

When we are training our network, we also need to ensure that performance for unseen data is good. For that reason, we calculate the loss and accuracy with respect to the validation data set($\mathcal{Z}_{val}$). One of the scenarios to explain the importance of validation dataset is to check if our network overfits the training dataset or not. If the training loss is low but validation loss is high then we can conclude that there is overfitting. We must regularize the parameters to avoid this. We discuss the details of regularization in Section 5.4.

There are various options for loss functions like cross-entropy loss, hinge loss etc [66]. In this thesis we use *softmax loss* function. The cross-entropy loss function is defined as:

$$L_i(y) = \frac{e^{y_i}}{\sum_{k=1}^{N} e^{y_k}} \tag{5.6}$$

There are many approaches to solve the minimization problem mentioned in Equation 5.5. One of the methods is gradient descent with momentum. It can be mathematically

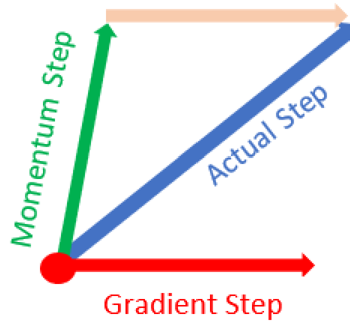


Figure 5.4.: One step of the gradient descent with momentum [66].

represented as:

$$\begin{aligned} v_n &= \beta v_{n-1} + (1-\beta)\nabla_{\theta_{n-1}} L(y) \\ \theta_n &= \theta_{n-1} - \alpha v_n \end{aligned} \tag{5.7}$$

where, α is the learning rate, β is the momentum factor and v_n is the momentum at n^{th} iteration.

In almost all the real-world scenarios, the dataset is very large and there are many network parameters. So, evaluation of the gradient for the complete dataset is restricted due to hardware limitation or loss in computational performance when the whole dataset is used. So, we use small batches of data and train out network. This approach when combined with gradient descent is called *stochastic gradient descent* method. This method is used in this thesis.

5.3. Convolutional Neural Networks

In Section 5.1 we discussed MLP for which the input was one-dimensional i.e. $x \in \mathbb{R}^n$. However, images are generally two-dimensional (gray-scale) $x \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_y}$, three-dimensional (RGB) $x \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_z}$ or higher dimensional. To directly use MLP, we need to convert input into a one-dimensional vector by straightening the higher-dimensional inputs. This will lead to a loss of the spatial informations. For handling such cases we use *Convolutional Neural Networks (CNN)*. In this section, we will discuss the structure and components of a Convolutional Neural Network.

5.3.1. Convolutional Layer

Let us consider an input of dimension $w_i \times h_i \times d_i$. A convolutional layer with filter size $w_f \times h_f \times d_i$, where $w_f \leq w_i$, $h_f \leq h_i$, when applied on each spatial position of a input layer yields a output of shape $w_o \times h_o \times 1$. This procedure can be visualized as a window sliding over input and doing some mathematical operations in the intersected zone. The amount by which the filter moves in respective dimension is called the stride and is represented by s_x and s_y for x-direction and y-direction respectively. The number of strides and the dimension of the filter determines the output size.

The application of convolution layer causes a shrink in the size of the output layer. This will add a restriction on the depth of the neural network. We can increase the size of the output layer to our requirement by adding some padding(p_x, p_y) in input. There are various options of padding like zero padding, replicate, etc. The spatial dimensions of output is given as following:

$$\begin{aligned} w_o &= \frac{w_i - h_f + 2p_x}{s_x} + 1 \\ h_o &= \frac{h_i - h_f + 2p_y}{s_y} + 1 \end{aligned} \tag{5.8}$$

In most of the cases, there are more than one filter in a convolutional layer. If there are n filters, then output will be of form $w_o \times h_o \times n$. After the convolution operation, we operate activation function to introduce non-linearity. It is the same as discussed in Section 5.1.

5.3.2. Pooling

The function of pooling is to progressively reduce the spatial size of the representation thereby reducing the number of parameters and computation in the network [66]. The pooling layer operates independently on every depth slice of the input and re-sizes it spatially [66]. The commonly used pooling layer is average pooling and max pooling. In this thesis we have used *max pooling*.

5.3.3. Fully connected layer

After desired number of convolutions and pooling layers, the output is straightened and provided to a fully connected layer. The size of the output layer is the required number of output classes.

5.4. Regularization

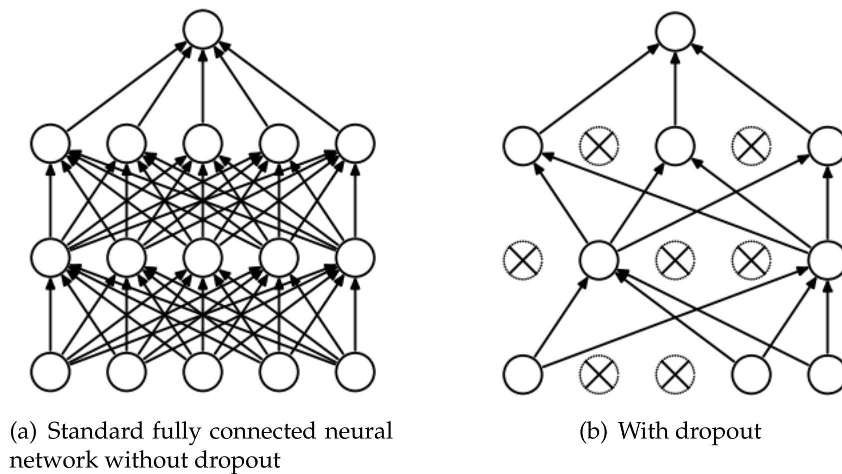
As discussed in Section 5.2, model might overfit the training dataset. In this section, we discuss regularization methods to prevent overfitting. This ensures that our model also predicts unseen data accurately.

5.4.1. Weight Decay

Weight Decay which is commonly known as L^2 *regularization* is one of the most commonly used regularization techniques in machine leaning. It is implemented by adding a penalty term to the loss function. These penalization term limits the capacity of the model. The new loss function is as follows [66]:

$$J_{decay}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \frac{1}{2}\lambda\|\boldsymbol{\theta}\|_2^2 \tag{5.9}$$

where, $\lambda > 0$ is a hyperparameter that controls the regularization strength. The factor of $\frac{1}{2}$ before the penalization term is to ease the differentiation of loss function with respect to the network parameters.



(a) Standard fully connected neural network without dropout

(b) With dropout

Figure 5.5.: Dropout Neural Network model, **Left:** A standard two-layered fully connected neural network, **Right:** An example of neural network after applying dropout [104].

5.4.2. Dropout

Dropout [104] is a regularization technique where we randomly disable a neuron and its connections during the time of training. Applying dropout to a neural network is equivalent to sampling a thinned network as seen in Figure 5.5(b). Disabling neurons will prevent the network to over-rely on certain inputs too much, thereby leading to a better generalization by utilizing more of the inputs. Each neuron is disabled with an independent probability $p \in [0, 1)$.

Second advantage of the dropout is that it combines many different neural network architectures efficiently [104]. Combining models generally improves performance of machine learning methods. However, it will be inefficient to train many models independently. By including the dropout, we are effectively training and combining many models at the same time. Moreover, this method is also computationally more efficient [104].

5.4.3. Batch Normalization

When we update the network parameters $\boldsymbol{\theta}$ as per equation 5.7, we assume that network parameters $\boldsymbol{\theta}$ in other layer are constant. However, we update all of them at the same time. *Batch Normalization* is a regularization technique which reduces the amount by which the weights of hidden layers shift around (covariance shift). [54].

For a batch of m outputs of a layer, each value h_i is normalized to y_i' as [54] :

$$y_i' = \frac{h_i - \mu}{\sigma} \tag{5.10}$$

We compute mean(μ) and standard deviation(σ) during training. The element-wise cal-

culation for each value of h_i across the batch is done as [54] :

$$\mu = \frac{1}{m}\sum_{i=0}^{m-1} h_i$$
$$\sigma = \sqrt{\delta + \frac{1}{m}\sum_{i=0}^{m-1}(h_i - \mu)^2} \tag{5.11}$$

where $\delta > 0$ is a small constant to prevent division by zero in equation 5.10. Final bath-normalization value for each entry is computed as [54]:

$$y_i = \gamma y_i' + \beta \tag{5.12}$$

where γ and β are parameters learned during training.

5.5. Hyperparameter Optimization

We have mentioned throughout this chapter that there are many hyperparameters in a neural network. The designer of the network has to carefully initialize the hyperparameter values. But this is a very difficult task and it is almost impossible to get the best accuracy simply by human guess. As a result of which, automatic optimization of hyperparameters for deep learning algorithms has been attracting a lot of attention. In this section, we formally state the hyperparameter optimization problem.

5.5.1. Problem Statement

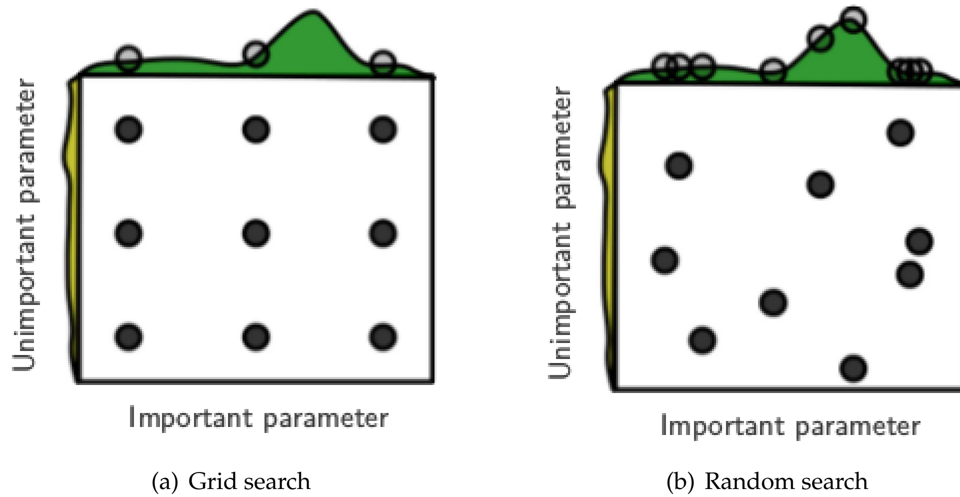
Let $\lambda = [\lambda_d^T, \lambda_c^T]$ such that $\lambda_d \in \mathbb{Z}^{n_d}$ and $\lambda_c \in \mathbb{R}^{n_c}$ denote required set of hyperparameters, f denote a negative validation accuracy. Hyperparameter Optimization is a global minimization of a black-box function f when the supervised training of the neural network is done using training dataset. The problem statement is mathematically stated as:

$$\begin{aligned} \min_{\lambda} \quad & f(x, \lambda, \boldsymbol{\theta}; \mathcal{Z}_{val}) \\ s.t. \quad & \boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} J(x, \lambda, \boldsymbol{\theta}; \mathcal{Z}_{train}) \end{aligned} \tag{5.13}$$

The problem mentioned in Equation 5.13, is a very challenging problem because of the complexity of the function f.

5.5.2. Existing Methods

We can find a rich literature of various hyperparameter optimization methods. The most naive solution to the aforementioned problem is to do *Grid search*. In this method, we create a grid of points in hyperparameter space, Then, we train our network for each point and choose the hyperparameter set with least validation accuracy. Instead of creating a uniform grid, we can also form a random grid and follow the method mentioned before. This is called *Random search* and it performs better than grid search [10]. Figure 5.6 shows and illustrative example of the grid and random search methods. Both these methods are



(a) Grid search (b) Random search

Figure 5.6.: Illustrative example of grid search and random search [66].

relatively easy to implement. However, they suffer from the curse of dimensionality. As the number of hyperparameters increases, it becomes infeasible to implement these methods. Moreover, these methods also are not able to find the best possible combination of hyperparameters. We will need a very fine grid to obtain better set. But fine grid requires more training, thereby making it computationally more expensive.

Other, approach is to use *Bayesian optimization* [80]. It is a method to find global minimum of a black-box function. This requires building a gaussian process surrogate of the required function by evaluating at some points. Then, we find the location point that has highest probability of the optimum point. We evaluate value of black-box at that point and use its value to improve the gaussian process which again is used to find next expected minimum. This process is repeated till required number of black-box evaluations are done. One of the commonly used tool that is uses this method is *Spearmint* [62, 35].

We need to solve a linear system of equation to obtain next best point in bayesian optimization. The size of linear system increases as more black-box evaluations are completed or adding more hyperparameters. This makes bayesian optimization very time consuming. To circumvent this problem, we can add tree based search method. This approach is done in *SMAC* [60, 59] and *TPE* [11, 12]. But this leads to loss in the quality of final results. In other words, optimizer is faster, but it is not able to find the lowest value.

Another recent development in hyperparameter optimization field is *HORD* (Hyperparameter Optimization using RBF and Dynamic co-ordinate search) [61]. This method is similar to the trust-region optimization method as discussed earlier. HORD uses radial basis functions and distorts design space co-ordinates with some probability. For more details of the algorithm refer [94].

Part III.

Experiment Setup and Results

6. Mixed Integer SNOWPAC Performance

This chapter discusses the behavior and performance of Mixed Integer SNOWPAC. We start by providing an illustrative example in Section 6.1. We use SNOWPAC to solve a two dimensional optimization problem and analyze the salient steps of the algorithm. In Section 6.2, we take an overview of the benchmark used to compare various optimization tools. Thereafter, we benchmark the performance of SNOWPAC against some existing tools for mixed-integer box-constrained problems and stochastic mixed-integer box-constrained problems in Sub-sections 6.2.1 and 6.2.2 respectively.

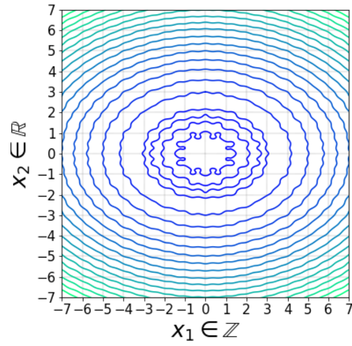
6.1. Illustrative Two Dimensional Example

We will first illustrate the behavior of the different aspects of mixed integer trust-region optimization method discussed in Chapter 4. Let us consider the mixed-integer *Bohachevsky Problem-1* [3, 22] stated as:

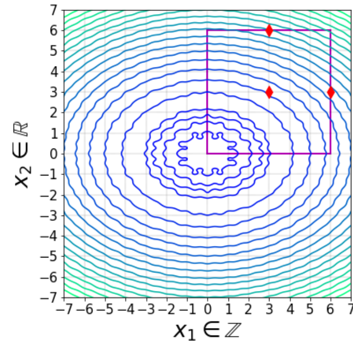
$$\begin{aligned} \min_{x} \quad & f(x) = x_1^2 + 2x_2^2 - 0.3cos(3\pi x_1) - 0.4cos(4\pi x_2) + 0.7, \\ s.t. \quad & -7 \leq x_i \leq 7, \qquad i = 1,2 \\ & x_1 \in \mathbb{Z}, \\ & x_2 \in \mathbb{R}. \end{aligned} \tag{6.1}$$

Figure 6.1(a) shows the contour lines of the *Bohachevsky Problem-1*. It is a multi-modal function with many local minimums. The global minimum is at $x^* = (0,0)$ with $f(x^*) = 0$. As mentioned in Chapter 4, let ρ_i represent half of the trust-region box size in i^{th} dimension. Let N represent the number of function evaluations.

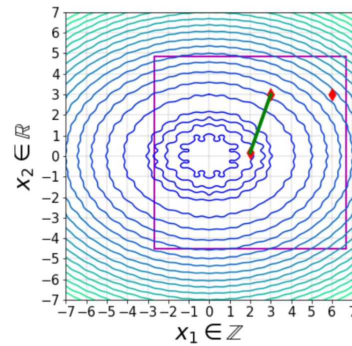
We start with the initial point $x^0 = (3,3)$ and half box-size as $\rho_1 = 3$ and $\rho_1 = 3$. Figure 6.1(b) shows the starting configuration of the optimizer. We use the points $(3,3)$, $(6,3)$ and $(3,6)$ for the initial evaluation of the fully liner model. We get the minimum of model within the trust-region at $(2,0.17)$. We update the trust-region region radius according to the acceptance ratio value. The updated trust-region is shown in Figure 6.1(c). Point $(3,6)$ falls out of the trust region. So, we remove it from the list of active points and do not use it to build the next surrogate model. We follow similar steps as mentioned before. The configuration after $N = 5, 18, 40$ is shown in Figure 6.1(d), 6.1(e) and 6.1(f) respectively. We can see from Figure 6.1(f) that the trust-region box size was becoming less than one. As mentioned in Chapter 4, the half box-dimension along the integer-dimension (ρ_1 in the given example) cannot be less than one. Hence, we restrict ρ_1 to one.



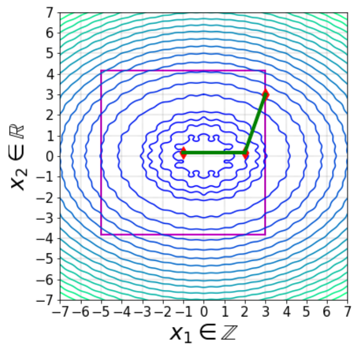
(a) Contour lines for the optimization problem 6.1



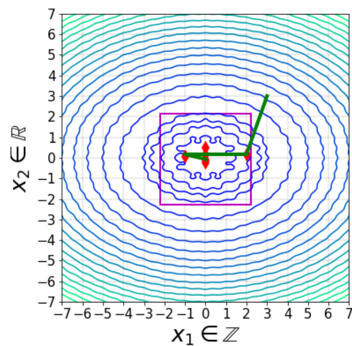
(b) $N = 3, \rho_1 = 3, \rho_2 = 3$



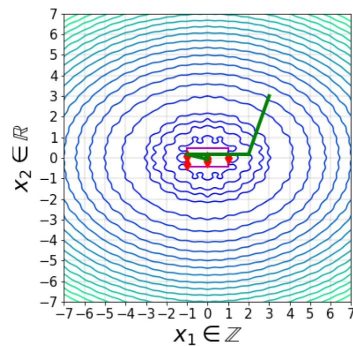
(c) $N = 4, \rho_1 = 4.68, \rho_2 = 4.68$



(d) $N = 5, \rho_1 = 4, \rho_2 = 4$



(e) $N = 18, \rho_1 = 2.2, \rho_2 = 2.2$



(f) $N = 40, \rho_1 = 1, \rho_2 = 0.45$

Figure 6.1.: Steps of solving Bohachevsky Problem 1 using Mixed Integer SNOWPAC. Magenta box represents the boundary of the trust-region. Red diamonds represents the active nodes that are used to construct the surrogate model at any given iteration. Dark green line represents the path of the optimum point during the given iteration number.

6.2. Performance Profiling Benchmarks

There are various methods to compare different optimization solvers. Some of the benchmarking efforts involve tables displaying the performance of each solver on various problems for a set of metrics such as CPU time, number of function evaluations, or iteration counts for algorithms where an iteration implies a comparable amount of work [44]. In this thesis, we use performance profiling as described by Moore et al [81] and box-plot depicting distribution of evaluation points to compare various methods.

We generate benchmark results by running a given solver on a set of problems $\mathcal{P}$ and recording the information of interest such as the number of function evaluations, distance from the optimum point, etc. Let, $t_{p,S}$ denote the minimum number of function evaluations taken by a solver S to solve problem $p \in \mathcal{P}$ with the accuracy requirement given as:

$$f(x) - f(x^*) \leq \epsilon_f$$

where $f(x)$ represent the function value at a given step, $f(x^*)$ is the known minimum of the function and ϵ_f is the required level of accuracy. Let n_p denote the number of design parameters of the optimization problem $p \in \mathcal{P}$. Then, the data profile is calculated as:

$$d_s(\alpha) = \frac{1}{|\mathcal{P}|} \left| \left\{ p \in \mathcal{P} : \frac{t_{p,S}}{n_p + 1} \leq \alpha \right\} \right| \tag{6.2}$$

where $|\mathcal{P}|$ represents the total number of benchmark runs. For example, there are 10 benchmarks and we run each benchmark 10 times, then the value of $|\mathcal{P}|$ is 100. Higher the value of $d_s(\alpha)$, better is the optimizer.

6.2.1. Mixed Integer Box-Constrained Problem

In this section, we compare the performance of various optimization tools to solve the mixed-integer box constrained problems. The solvers compared are Mixed Integer SNOWPAC (developed as part of this thesis), pySOT [39], TPE (Tree-Structured Parzen's Estimator) [11, 12], SMAC (Sequential Model-based Algorithm) [60, 59] and NOMAD [74, 38].

<table>
<tr><th>Sr.No.</th><th>Category</th><th>Problem Name</th><th>n</th><th>n_d</th><th>n_c</th></tr>
<tr><td>1</td><td rowspan="5">Category 1</td><td rowspan="5">Sphere Problem [110]</td><td>2</td><td>1</td><td>1</td></tr>
<tr><td>2</td><td>4</td><td>2</td><td>2</td></tr>
<tr><td>3</td><td>6</td><td>3</td><td>3</td></tr>
<tr><td>4</td><td>8</td><td>4</td><td>4</td></tr>
<tr><td>5</td><td>10</td><td>5</td><td>5</td></tr>
<tr><td>6</td><td rowspan="5">Category 2</td><td>Ackley's Function [3, 105, 110]</td><td>8</td><td>3</td><td>5</td></tr>
<tr><td>7</td><td>De Jong's Problem [110]</td><td>5</td><td>3</td><td>2</td></tr>
<tr><td>8</td><td>Bohachevsky Function 1 [3, 22]</td><td>2</td><td>1</td><td>1</td></tr>
<tr><td>9</td><td>Bohachevsky Function 2 [3, 22]</td><td>2</td><td>1</td><td>1</td></tr>
<tr><td>10</td><td>Griewank Function [110]</td><td>10</td><td>5</td><td>5</td></tr>
</table>

Table 6.1.: List of benchmark problems used for comparison of optimization tools.

The list of benchmark problems is mentioned in Table 6.2.1. We present the complete problem statement in Appendix A.1. We divide the benchmarks into two categories. The first category belongs to set of simple *Sphere problems*. The functions are convex with one unique minimum. There are five problems with increasing number of design parameters. We use this category to study the effect of increasing the dimension on behavior of various optimization tools.

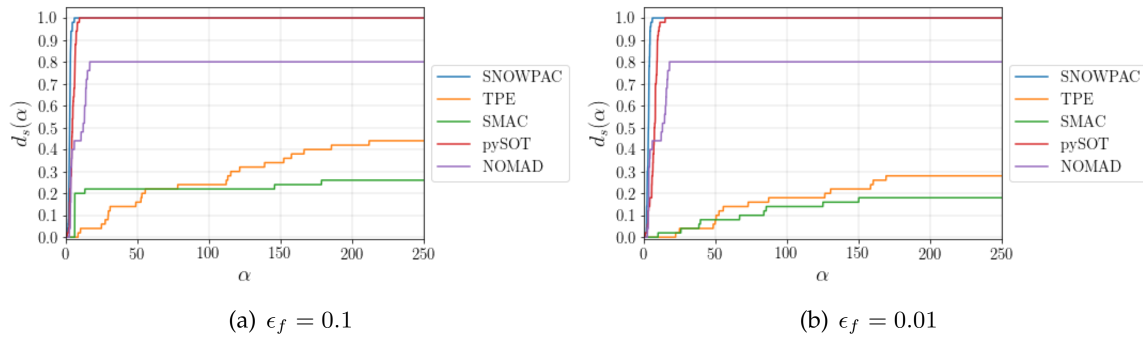
We put some difficult optimization problem in the second category. Some problems in this category are multi-modal (i.e. many local minimums) whereas in other problems axis of some design parameters are elongated. By this category of optimization problems, we can judge the capability of tools to solve certain typical problems.

We draw the performance profiling graph using two levels of accuracy requirements as $\epsilon_f = 0.1$ and $\epsilon_f = 0.01$. Each benchmark problem is solved 10 times. In this section, we only show some the graphs to explain the salient performance observations. For all other plots, please refer to Appendix A.1.

In addition to the provided information, SMAC and SNOWPAC also needs a starting point. For every set of experiment, we choose a random integer starting point within the trust region. SNOWPAC also needs initial box-dimension which we set as three for all the problems.

Figure 6.2 shows the overall performance profile for the first category of benchmark problems. SNOWPAC (blue line) shows the best performance among all the optimizer, very closely followed by pySOT (red line), NOMAD (purple line), TPE (orange line) and SMAC (green line). The performance of SNOWPAC and pySOT is way superior as compared to the other tools. This is because both of them are based on trust-region derivative free optimization method. Both of them swiftly move towards a local minimum. SNOWPAC beats pySOT for small α because SNOWPAC starts optimization with fully linear model which requires at least $n + 1$ evaluation points, where n is the number of design parameters. However, pySOT need $2n + 1$ points. So, SNOWPAC starts searching for trial points before pySOT, which explains better behavior of SNOWPAC for low α. NOMAD is directional search method, and it shows inferior performance as compared to trust-region methods. This behavior is also shown in [85]. SMAC and TPE are methods based on bayesian optimization and tree structure is added to speed-up calculation of trial nodes [11, 60]. It has been shown in [8] that Bayesian Optimization performs worse than trust-region methods for continuous constrained problems. Similar behavior is observed in the Figure 6.2. There is deterioration in performance when the value of accuracy requirement(ϵ_f) is decreased because the optimizer will need more function evaluation to reach more accurate points.

Figure 6.3 shows the statistical distribution of the function evaluation points by optimization tool in the design parameter space. The dotted blue line represents the actual minimum of the benchmark function. We observe from the figure that SNOWPAC, pySOT and NOMAD have more function evaluation near the required minimum. This means that these three optimization tool do more exploration of the design space near the optimum parameters. This accounts for their better performance as compared to SMAC and TPE. As mentioned before, TPE and SMAC are based on bayesian optimization. Bayesian optimization is used to find the global minimum, by gradually improving the surrogate gaussian process model over the whole domain. So, both needs to explore whole domain thereby causing a wide statistical distribution.

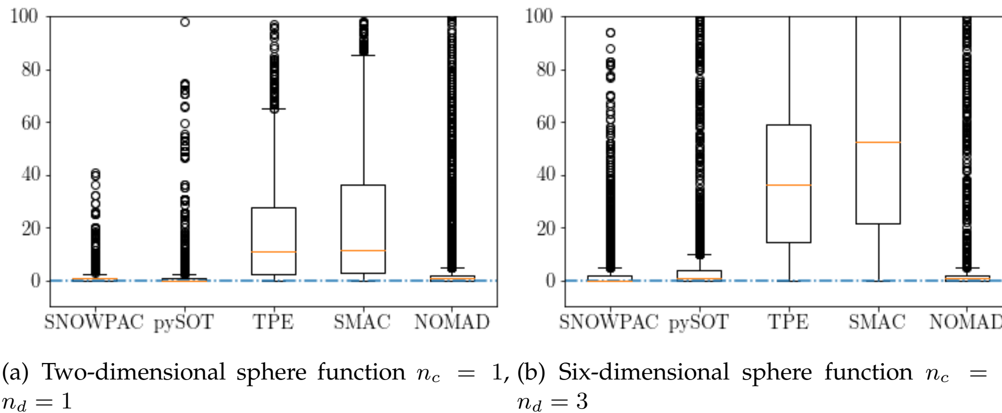


(a) $\epsilon_f = 0.1$

(b) $\epsilon_f = 0.01$

Figure 6.2.: Overall performance profiling result for the first category of benchmark problems.

We can also observe from Figure 6.3 that distribution more wide spread when dimension of design space increases. All the optimization tools suffer from increasing dimension. As we increase the dimension of the optimization problem, we will need more points to build the surrogate thereby causing more spread of the evaluation points in the design space. This also explains deterioration in performance profiling graphs when the dimension is increased (refer to Appendix A.1).

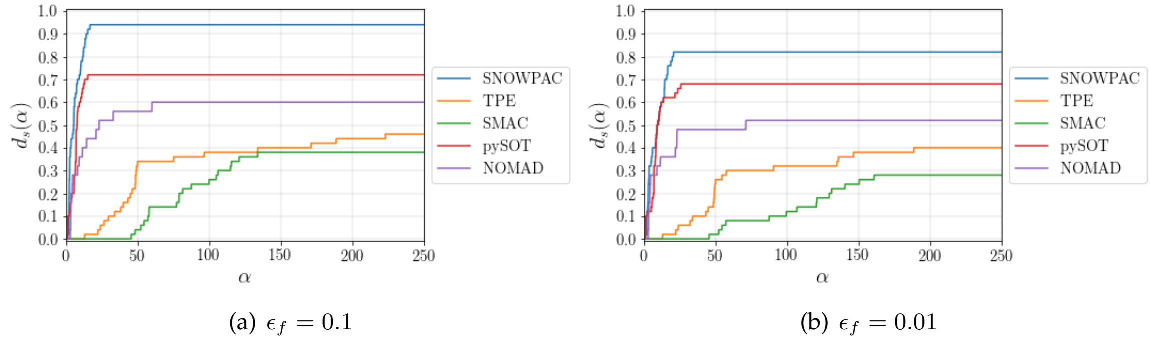


(a) Two-dimensional sphere function $n_c = 1$, $n_d = 1$

(b) Six-dimensional sphere function $n_c = 3$, $n_d = 3$

Figure 6.3.: Box plot distribution of evaluation points for first category of benchmarks.

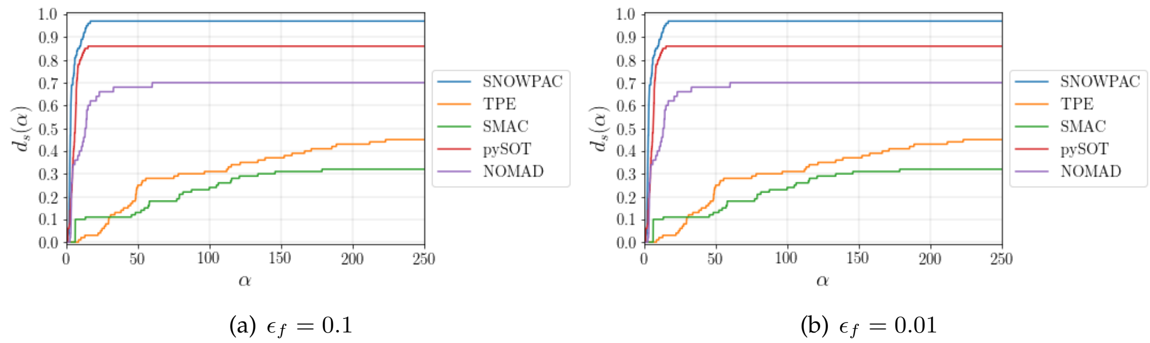
Figure 6.4 shows the performance profile of benchmarked optimization tools over the second category of benchmark problems. These optimization problems are more difficult to solve as compared to the first category. As expected, the performance declines as compared to the first category. SNOWPAC's plot is highest as compared to others. We also observe that performance profile of TPE and SMAC is lowest. This is because both of them are based on bayesian optimization. Bayesian optimization performs poorly in optimizing multi-modal functions because it generates surrogate for the complete design space. Multi-modal functions being highly oscillatory needs many points to generate global surrogate. Since the surrogate is inaccurate, we observe an under-par performance by TPE and SMAC. The detailed comparison between various bayesian optimization tools



(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure 6.4.: Overall performance profiling result for the second category of benchmark problems.

for multi-modal function is done by Dewancker et. al. [43]. For individual profile graph and box-plots refer to Appendix B.



(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure 6.5.: Overall Profiling Result for box-constrained problems.

In contrast to the bayesian optimization method, the trust-region based method builds only local surrogate. Just few points can be used to build an accurate local surrogate model. Therefore, the trust-region based optimization methods converges quickly towards the local minimum. However, for multi-modal functions there are many local minimums. Nevertheless, we cannot guarantee convergence to global minimum. If the global minimum is dominant (difference between function values from local minima is large) then chances to find global minimum increases.

Figure 6.5 shows the combined performance profiles for all the benchmark problems mentioned in Table 6.2.1. SNOWPAC shows the best performance for solving mixed integer box-constrained optimization problems as compared to the other optimization tools for the given benchmark setup.

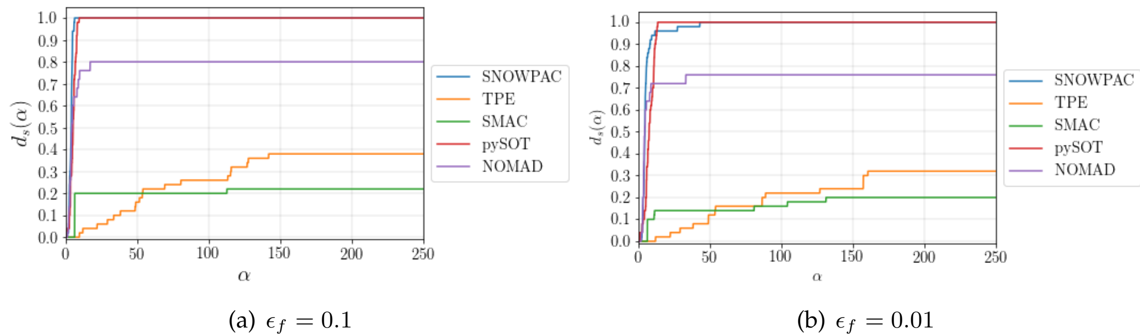
6.2.2. Mixed Integer Stochastic Box-Constrained Problem

In this section, we discuss the performance comparison between the optimization tools as mentioned in subsection 6.2.1 but for stochastic functions. We consider only the first

category of benchmark problems (Sphere Problem [110]). We add small uniform noise to the function. We consider the expected value as the robustness measure to be optimized which is calculated using a sample size of 100.

Figure 6.6 shows the performance profile graph for stochastic sphere functions. For all graphs refer Appendix B. The behavior is similar to Figure 6.2. The low performance of TPE and SMAC is because Bayesian optimization under-performs for high-dimensional problems. SNOWPAC and pySOT performs equally well. This can also be explained by the fact that both of them are based on the trust-region method.

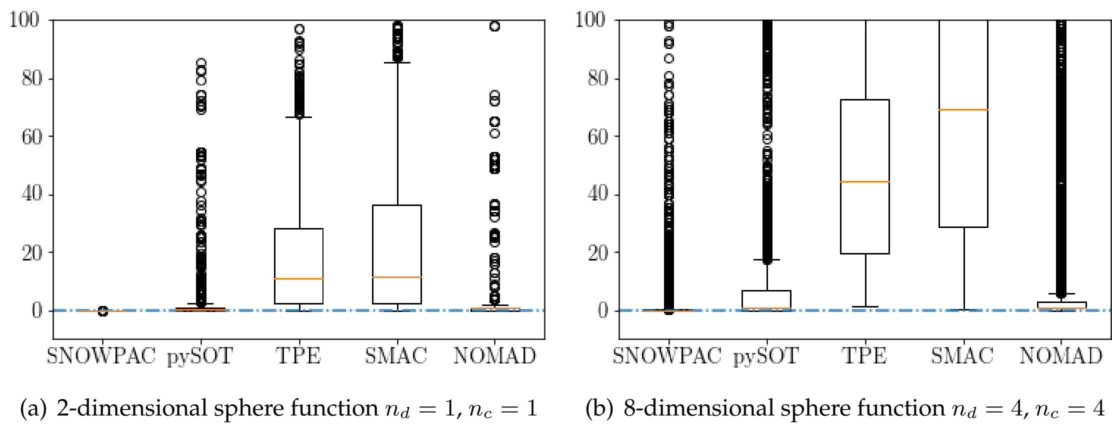
Figure 6.7(a) and Figure 6.7(b) shows the statistical distribution of evaluation points in form of box-plot for 2-dimensional and 4-dimensional stochastic sphere function respectively. As expected the variance of evaluation points increases with increasing dimension of design parameters. However, SNOWPAC shows the smallest variance as compared to other optimization tools. This points out a slight advantage of SNOWPAC over pySOT. There is higher variance for pySOT because it starts by building quadratic model over entire design space and then it centers towards the optimum point. However, SNOWPAC starts by building a fully linear surrogate model over the given trust-region which is not necessarily the whole domain.



(a) $\epsilon_f = 0.1$

(b) $\epsilon_f = 0.01$

Figure 6.6.: Overall Profiling Result for stochastic sphere functions.



(a) 2-dimensional sphere function $n_d = 1$, $n_c = 1$

(b) 8-dimensional sphere function $n_d = 4$, $n_c = 4$

Figure 6.7.: Box plot distribution of evaluation points for mixed-integer stochastic optimization done on stochastic sphere function.

7. Neural Network Hyperparameter Optimization

In this chapter, we compare the performance of various optimization tools to optimize the hyperparameters of neural network. We benchmark the performance of SNOWPAC against various optimizers that are commonly used for hyperparameter optimization. In Section 7.1, we describe the experimental setup and the categories. Thereafter, we present results from the experiment, comparison and discussion of outcomes in Section 7.2.

7.1. Experimental Setup

We devise eight different setups for comparing various optimizers. We can divide the experimental setups into two categories of four each. Both these categories have a similar problem set containing same neural network architecture, dataset and hyperparameters to be optimized. The only difference lies in the scale of individual hyperparameters. In the first category we do not scale the hyperparameters whereas in the second category we scale the hyperparameters so that all are of comparable magnitude. For example, learning rate is in the natural scale in the first category whereas in the second category it is in the logarithmic (log_{10}) scale. We provide the details of scaling in Table C.2, C.4, C.6 and C.8. We will now describe the basic structure of the four problems.

As mentioned earlier, each category has four hyperparameter optimization problems. The problem design is taken from Ilievski et. al. [61]. In the first problems we optimize six hyperparameters of the MLP network. Out of the six hyperparameters to be optimized, two are integer parameters and four are continuous parameters. The complete list of the hyperparameters optimized are mentioned in Table C.1 and Table C.2 for the first and the second category respectively. We refer this problem as **6-MLP** in subsequent discussions. The MLP network consists of two hidden layers with ReLU activation between them and SoftMax loss term. We use Stochastic Gradient Descent (SGD) for training the weights of the neural network. The network is trained to classify grayscale images of handwritten digits from the popular benchmark dataset *MNIST* [75]. MNIST is a database of handwritten digits in form of gray-scale images. Each image is labeled from 0 to 9 depending upon the image. There are 60000 samples for training and 10000 samples for testing/validation. The digits are size-normalized and centered in a fixed-size image. Each image has 28×28 pixels. Figure 7.1 shows few sample images from MNIST dataset.

The second problem is to optimize eight hyperparameters of a Convolutional Neural Network (CNN) with four integer hyperparameters and four continuous hyperparameters. The architecture of CNN includes two convolutional blocks, each containing one convolutional layer with batch normalization. It is followed by ReLU activation and a 3×3 max-pooling layer. Next to the convolutional block, we place two fully-connected layers

Figure 7.1.: Some sample images from MNIST dataset [75].

with LeakyReLU activation. We use SoftMax loss term. We train the network to classify images in digits and use MNIST dataset. We call this problem as **8-CNN** in subsequent discussions. For the details of hyperparameters optimized, refer to Table C.3 and Table C.4 for the first and the second category respectively.

The third problem uses the exact same setup as the second problem. We just increase the number of target hyperparameters to be optimized to fifteen, with five integer parameters and ten continuous parameters. We will call this problem as **15-CNN** in further discussions. For the details of hyperparameters optimized, refer to Table C.5 and Table C.6 for the first and the second category respectively.

The fourth problem optimizes nineteen hyperparameters, which consists of five integer parameters and fourteen continuous parameters. We call this problem as **19-CNN** in subsequent discussions. The architecture is the same as that of 8-CNN and 15-CNN, except that we include dropout layers after convolutional layers and fully-connected layers. For the details of hyperparameters optimized, refer to Table C.7 and Table C.8 for first and second category respectively. We design the network to classify colored images from *CIFAR-10* dataset. CIFAR-10 [69] dataset consists of 60000 color images each with each image having 32×32 pixels. There are 10 classes, with each class having 6000 images. There are 50000 training images and 10000 test images. Figure 7.1 shows few sample images of every class from CIFAR-10 dataset.

We compare Mixed Integer SNOWPAC (developed as part of this thesis) against HORD [61], HORD-ISP [61] (both HORD and HORD-ISP uses pySOT [39]), Spearmint [35] (code used is wrapper around original Spearmint code [62]), SMAC [60, 59] and TPE [11, 12]

Training a neural network is typically a computationally expensive process. So, it is desirable to find the optimum hyperparameters values within limited number of full trainings. Therefore, we limit the number of optimization iterations to 200 black-box evaluations. Here, one black-box evaluation corresponds to one full training of the neural network. We run five trials of each setup. We do each trial with different random seed for neural network weight initialization, where the random seed is constant for all optimization tools. Some of the optimizers need a starting point. We summarize the initial points in Appendix C. These initial points are like *a priori* information that is provided by the

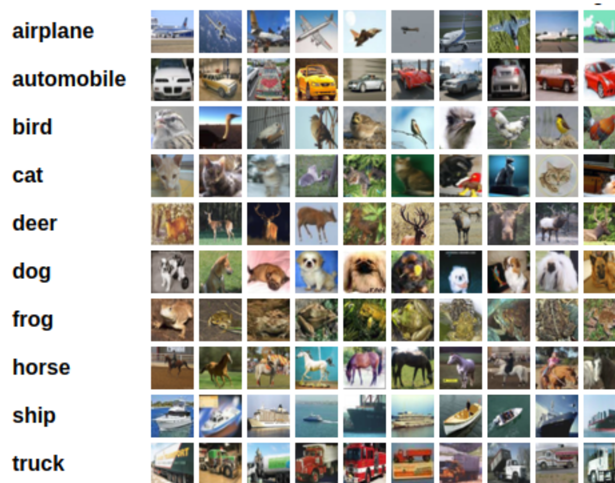


Figure 7.2.: Some sample images of every class from CIFAR10 dataset [69].

user to the optimizer. The optimization tools that need initial starting points are SNOWPAC, HORD-ISP and SMAC. We also need to provide the starting trust-region dimension to SNOWPAC. We choose the value in such a way that the whole hyperparameter space is covered within the initial trust-region. The starting trust-region box-dimension is in Table C.9.

7.2. Results and Discussion

In this section, we present the results from the experiments explained in the previous section. We asses the efficiency of various optimization tools with respect to the increase in the dimension of hyperparameter space, how likely will it be to produce best validation accuracy, how many number of function evaluations does it need to get best accuracy, the statistical distribution of the evaluation points and the amount of non-evaluation time. Firstly, we discuss the results obtained from the first category in subsection 7.2.1. Next, we present the results from the second category of problems in subsection in 7.2.2. Then we compare the non-evaluation of in subsection 7.2.3. Finally, we show the results obtained from running the stochastic 8-CNN second category problem in subsection 7.2.4.

7.2.1. First Category (Unscaled Hyperparameter Space)

We first discuss the results with respect to the first category of problems (unscaled hyperparameter space). Table 7.1 shows the mean and the standard deviation of percentage validation error over five experiments as described in Section 7.1 . We observe that, in two out of four setups, SNOWPAC yields the lowest mean percentage validation error. For 15-CNN, SNOWPAC comes very close to the lowest error. However, it is being outperformed by Spearmint and HORD-ISP in 19-CNN setup.

As mentioned in Chapter 4, the trust region based optimization methods converges to a local minimum. The loss-surface of a multilayer network is highly convex and has many

	MNIST 6-MLP	MNIST 8-CNN	MNIST 15-CNN	CIFAR-10 19-CNN
SNOWPAC	**1.72 (0.05)**	**0.75 (0.03)**	0.76 (0.08)	20.65 (1.21)
HORD	1.88 (0.09)	0.77 (0.02)	0.83 (0.08)	23.68 (1.86)
HORD-ISP	1.88 (0.07)	0.76 (0.02)	**0.75 (0.04)**	19.6 (1.91)
Spearmint	1.82 (0.05)	0.77 (0.03)	0.81 (0.01)	**19.18 (0.71)**
TPE	2 (0.06)	0.82 (0.04)	1.04 (0.05)	24.77 (1.83)
SMAC	1.9 (0.05)	0.85 (0.04)	0.94 (0.04)	25.58 (1.54)

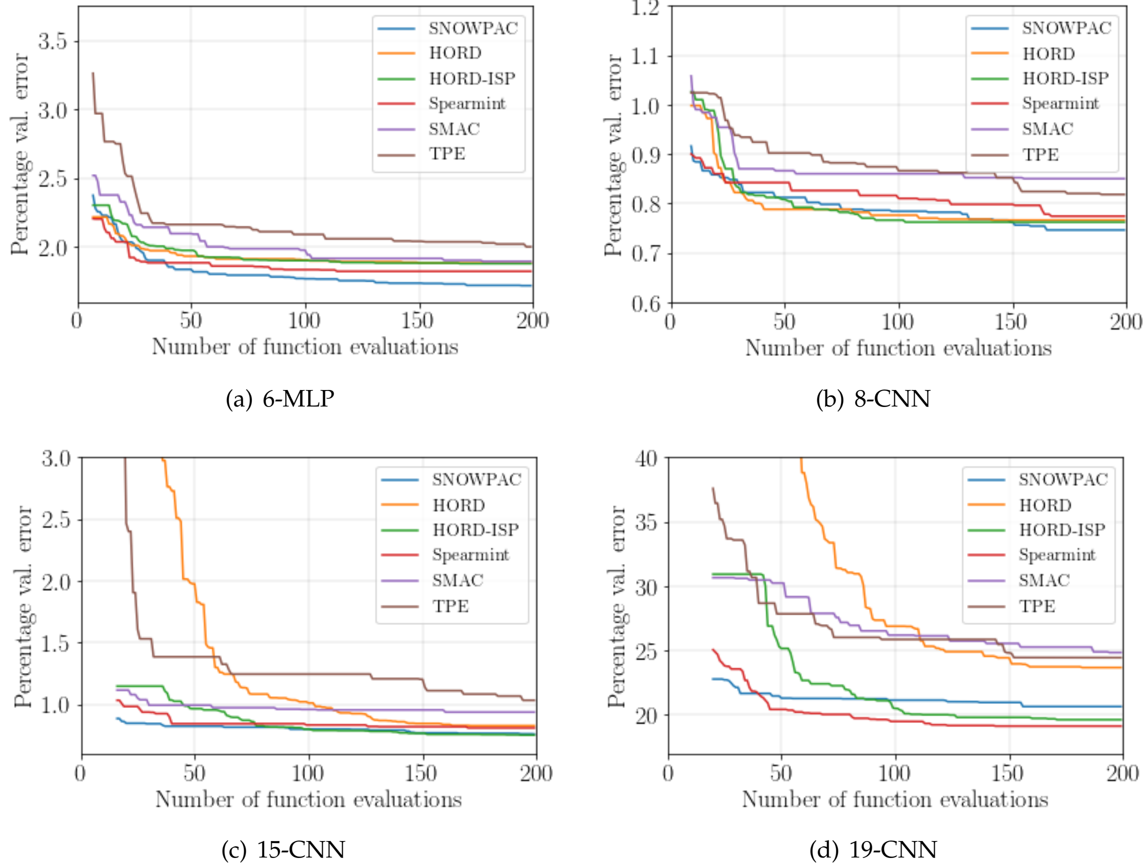
Table 7.1.: Mean and standard deviation (inside parenthesis) of percentage validation error over 5 sample of experiments for the first category of problems (unscaled hyperparameter space). The optimization tool with minimum mean validation error is shown in bold.

local minimums [30]. Similarly, there can be many local minimums for validation error with respect to the hyperparameters too. SNOWPAC being a trust-region based optimization method, gets stuck to one of the local minimums. However, Spearmint being based on Bayesian optimization method is designed to find a global minimum. This can explain the reason why Spearmint performs better for 19-CNN setup. But, the main disadvantage of Spearmint is high non-evaluation time as compared to other methods. We will explain this point later in the sub-section 7.2.3, using Table 7.4. TPE and SMAC are also based on Bayesian optimization but the algorithm is modified by including a tree-structure to ensure fast evaluation of the trial points. This makes them faster (lower non-evaluation time) as compared to Spearmint (see Table 7.4) and can be used specifically when dimension of the design space is high. However, this comes with a cost of not yielding the most optimum point. This behavior was also observed in previous Chapter 6 while discussing the results of performance profile graphs.

Figure 7.3 plots the average validation accuracy against the number of function evaluations. The plots of individual trials are in Appendix D. Since our target is to minimize the validation error (same as minimizing the negative of validation accuracy), smaller values represents better performance of the given optimization tool. We can observe that SNOWPAC performs better than TPE, SMAC and HORD in all the four tests. For the 6-MLP problem and the 8-CNN problem it performs better than Spearmint and HORD-ISP (see Figure 7.3(a), 7.3(b)). For the 15-CNN problem, performance is comparable to Spearmint and HORD-ISP (see Figure 7.3(c)). But for 19-CNN, Spearmint performs best followed by HORD-ISP and then SNOWPAC (see Figure 7.3(d)). As discussed earlier, SNOWPAC might get stuck in local minimum which contributes to poor performance for the 19-CNN problem. This aligns with the discussion done in the previous paragraph.

Figure 7.4 shows the statistical distribution of the evaluation points. A good optimizer should evaluate more points near the optimum value. We observe that, the distribution widens as number of hyperparameters to be optimized increases. This is because more points will be needed to build the surrogate as dimension is increased and those initial points should be more evenly distributed across the hyperparameter space. We can observe that Spearmint and SNOWPAC show a good distribution of the evaluation points.

Table 7.2 shows the final trust-region dimension in for SNOWPAC after 200 function



(a) 6-MLP

(b) 8-CNN

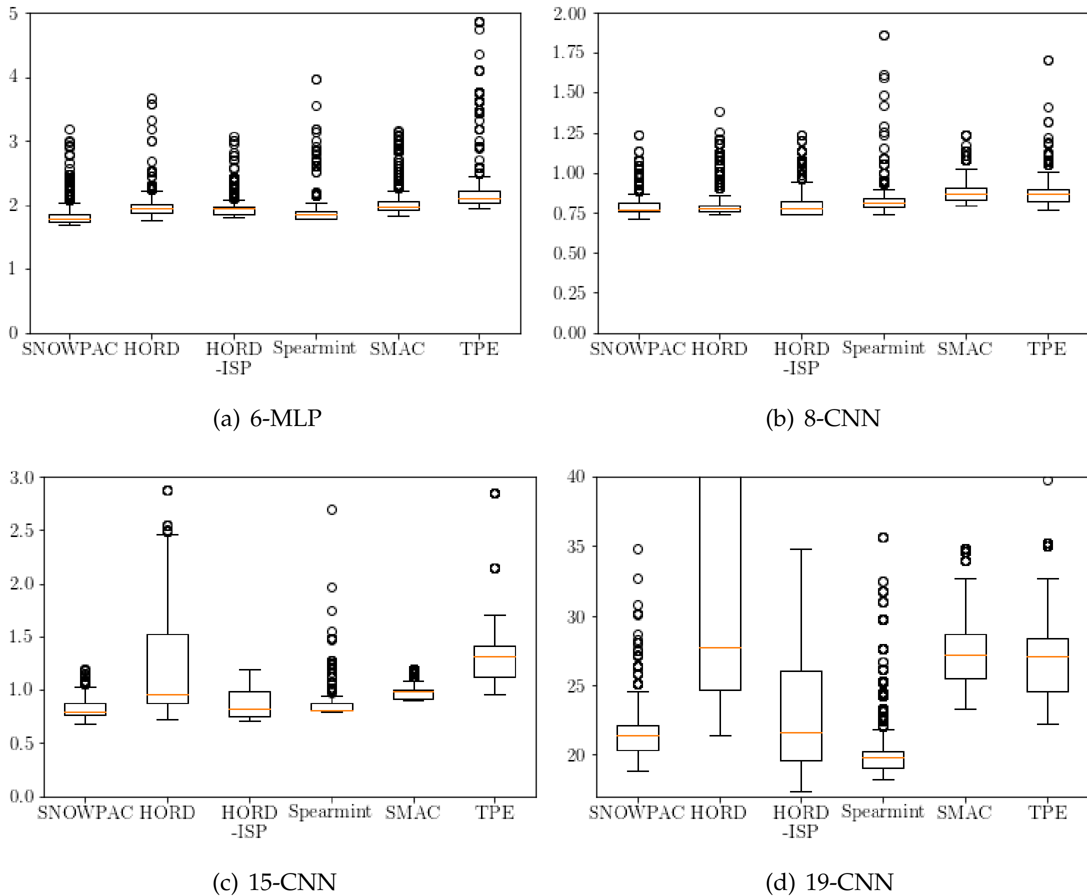
(c) 15-CNN

(d) 19-CNN

Figure 7.3.: Average percentage validation error vs number of function evaluations for all the setups in unscaled hyperparameter space(first category).

	6-MLP	**8-CNN**	**15-CNN**	**19-CNN**
Category 1	7.3×10^{-5}	6.2×10^{-4}	7.8×10^{-2}	3.95
Category 2	1.2×10^{-5}	1.2×10^{-4}	4.2×10^{-2}	6.5×10^{-2}

Table 7.2.: Final half box-dimension for SNOWPAC after 200 function evaluations.



(a) 6-MLP

(b) 8-CNN

(c) 15-CNN

(d) 19-CNN

Figure 7.4.: Box plot distribution of evaluation points for neural network hyperparameter optimization on unscaled setup.

evaluations. As shown in Chapter 2, smaller the trust-region better will be the accuracy of the surrogate model. After 200 function evaluations (complete optimization) the magnitude of maximum trust-region dimension for the 19-CNN problem is 3.95 (see first row and last column of Table 7.2). Let us consider hyperparameters like learning rate, weight decay, dropout, etc. whose range is much smaller than the maximum trust-region dimension. This means that the box-dimension for such hyperparameters do not change during the course of optimization. In other words, the trust-region covers the whole design space along these hyperparameters. Therefore, we have accuracy of the model near the optimum point as compared to other cases when the trust-region is much smaller (see Table 7.2). One of the ways to address this problem is to decrease the trust-region shrinking factor(γ) to a smaller value so that over 200 function evaluation the trust-region dimension goes to a smaller value. Another way to handle this issue is to design the optimization problem in a way that the range for all the hyperparameters are of comparable magnitude. We can do that by scaling the hyperparameters as discussed in the next subsection.

7.2.2. Second Category (Scaled Hyperparameter Space)

In this subsection, we discuss the results in the scaled hyperparameter space. In the first category of problem, the range of certain hyperparameters were very large as compared to others. For instance in 15-CNN unscaled problem, range of number of hidden nodes in first fully connected layer is $[100, 400]$. Whereas the range of weight decay is $[0.0, 0.01]$. The range of weight decay is approximately 10000 times smaller than that of hidden nodes number. As discussed in the last paragraph of the subsection 7.2.1, this leads to the inaccurate model along certain hyperparameters. The second reason for scaling the hyperparameters is to ensure that percentage validation error does not change abruptly with a small modification in the value of certain parameters. For example, let us consider the learning rate where a small change from 0.1 to 0.01 can cause a big change in the validation accuracy. On the other hand, changing the number of neurons by a small integer generally does not have a significant effect on the output. In such scenarios, the surrogate will not be accurate because we need more points to build a good surrogate model. Therefore, it makes sense to keep the learning rate in the logarithmic scale (log_{10}). Moreover, we keep the number of neurons, depth of convolution layer and mini-batch size in power of two to make best use of GPU resources. We summarize the scaling factors, range and starting points for all the setups in Table C.2, C.4, C.6 and C.8.

Table 7.3 shows the mean and the standard deviation of percentage validation error over five experiments as described in Section 7.1 for the second category. On comparison of the results obtained from the first category (see Table 7.1), we observe that the validation error obtained by all the optimizers is better in the first category (unscaled parameters) than in the second category (scaled parameters) for all the setups except the 6-MLP problem. In the second category the maximum number of training epoch is 50 whereas it is 20 in the first category. This explains why the second category outperforms the first one for the 6-MLP problem. To understand the reason behind the better performance of the first category in all other setups, we need to look into the scaling of the integer parameters. All these parameters are in the power of two. This causes a big jump in the actual values. For example, the values of 2^{λ} changes from 256 to 512 when λ increase from 8 to 9. This ignores the possible integer values in the range $(256, 512)$. The function minimum can lie in those ignored values. This is the reason why the second category under-performs as compared to the first one.

We can observe from Table 7.3 and Figure 7.5 that SNOWPAC find hyperparameters with the leat validation error for all the setups except in 8-CNN. For the 8-CNN problem too the difference is very small. Unlike the first category, SNOWPAC even outperforms in the 19-CNN problem. When we scaled the integer hyperparameters, we decreased the number of values that it can have. This removes a lot of local minimums. Moreover, the dimension of trust-region after 200 function evaluations is 6.5×10^{-2} (see second row and fourth column of Table 7.2). This means that more accurate surrogate is built around the optimum point. This explains the reason why SNOWPAC performs better than other tools in the scaled hyperparameter space.

	MNIST 6-MLP	MNIST 8-CNN	MNIST 15-CNN	CIFAR-10 19-CNN
SNOWPAC	**1.38 (0.03)**	0.87 (0.08)	**0.92 (0.13)**	**22.83 (1.27)**
HORD	1.45 (0.03)	0.89 (0.05)	1.1 (0.14)	23.41 (0.73)
HORD-ISP	1.45 (0.05)	1.01 (0.06)	1.35 (0.36)	24 (0.26)
Spearmint	1.44 (0.04)	**0.85 (0.07)**	1.03 (0.06)	23.85 (0.78)
TPE	1.58 (0.02)	1.11 (0.04)	1.58 (0.09)	26.4 (0.38)
SMAC	1.51 (0.05)	0.92 (0.02)	1.35 (0.12)	24.06 (0.92)

Table 7.3.: Mean and standard deviation (inside parenthesis) of percentage validation error over 5 sample of experiments for the second category of problems (scaled hyperparameter space). The optimization tool with minimum mean validation error is shown in bold.

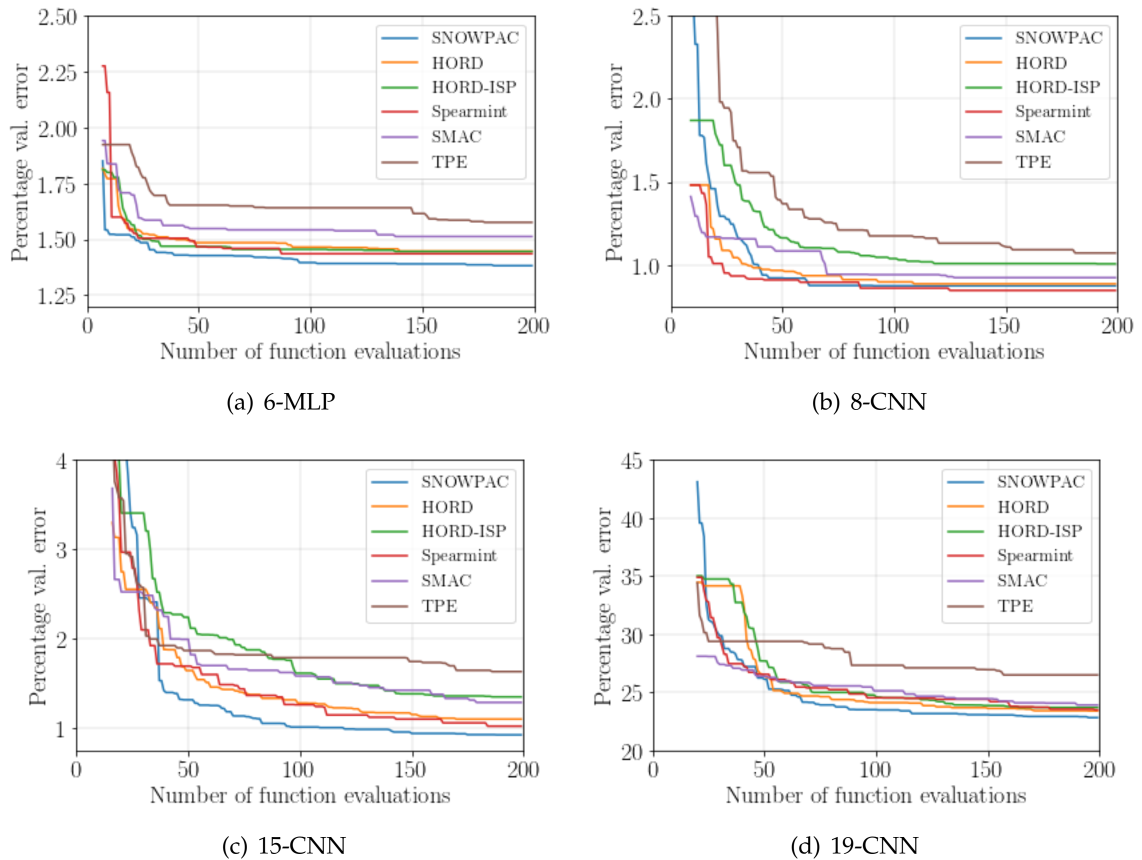


Figure 7.5.: Average percentage validation error vs number of function evaluations for all the setups in scaled hyperparameter space (second category).

7.2.3. Non-evaluation Time

In this section, we compare the time taken by various optimizers. The total time taken by black-box function evaluation (neural network training) depends upon the value of hyperparameters. For instance, more neurons in a layer increases the number of weights thereby increasing the training time. So, including black-box function evaluation time might bias the comparison. Therefore, we subtract the black-box function evaluation time from the total time. Then we divide the left over time by total number of function evaluation to obtain the mean time per step taken by optimization tool. For further discussions we refer this quantity as the *non-evaluation time*.

Table 7.4 shows the mean of non-evaluation time (in seconds) per optimization step taken by each optimizer. We observe that the non-evaluation time increases with increasing number of hyperparameters to be optimized. SNOWPAC takes least time for 6-MLP and 8-CNN problem. However, non-evaluation time taken increases by a big margin when we optimize 15-CNN and 19-CNN problem. This can be accounted by the fact that, mixed integer SNOWPAC uses branch and bound algorithm. As dimension of the problem increases, the tree depth might increase and one needs to solve more local optimization problems. Therefore, more time will be taken. However, we cannot explain this behavior in the term of algorithm complexity because branch and bound is a $\mathcal{NP}$-hard optimization algorithm.

	MNIST 6-MLP	MNIST 8-CNN	MNIST 15-CNN	CIFAR-10 19-CNN
SNOWPAC	**0.016 (0.07)**	**0.062 (0.08)**	0.454 (0.98)	0.57 (0.12)
HORD	0.328 (1.36)	0.363 (0.45)	0.410 (0.87)	0.433(0.08)
HORD-ISP	0.332 (1.45)	0.398 (0.49)	0.411 (1.03)	0.420 (0.11)
Spearmint	136.299 (86.88)	143.306 (62.71)	303.169 (85.94)	350.8 (28.77)
TPE	0.039 (0.15)	0.089 (0.12)	**0.154 (0.39)**	**0.175 (0.04)**
SMAC	1.47 (6.75)	2.044 (2.75)	3.216 (8.48)	4.848 (1.11)

Table 7.4.: Mean of non-evaluation time(in seconds) per optimization step taken by each optimization tool for optimizing hyperparameters. Mean percentage of non-evaluation time with respect to total optimization time is shown inside parenthesis. Entry with least non-evaluation time for a given optimization setup is depicted in bold.

We observe that amongst all the algorithms, Spearmint takes the maximum amount of non-evaluation time (see fourth row of Table 7.4). For the 6-MLP problem Spearmint is approximately 8500 times slower than SNOWPAC. The slow speed of Spearmint is its main disadvantage. Spearmint is based on bayesian optimization, which involves solving a linear system of equations to evaluate the next node. This method builds a global gaussian process surrogate. As the number of evaluation points increases, the matrix of the linear system to be solved also gets bigger. This is the reason why Spearmint has a large non-evaluation time. SMAC and TPE are also based in bayesian optimization but they are modified with addition of tree algorithm to increase the speed specially for the high dimensional problems. SNOWPAC, HORD and HORD-ISP are all based on the trust-region optimization. In such methods, we only optimize for a small local region with limited

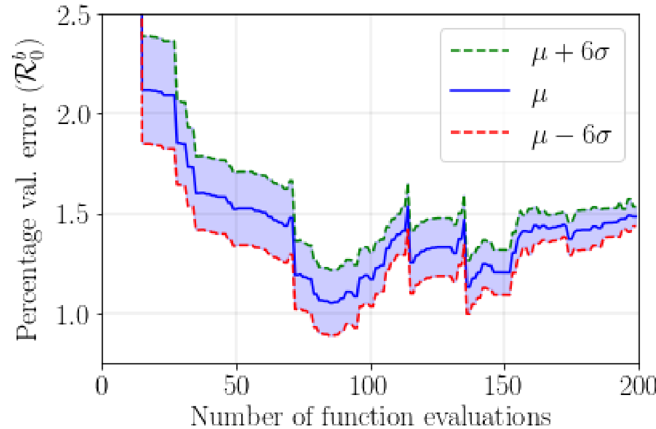


Figure 7.6.: Percentage mean validation error ($\mathcal{R}_0^b$) vs number of function evaluations for stochastic 8-CNN problem over sample size of $N = 100$. The mean robustness measure (μ) is represented by blue solid line. The lower specification line ($\mu - 6\sigma$) and the upper specification limit ($\mu + 6\sigma$) is represented by red and green dotted line respectively.

number of points. So, the linear system of equations does not get very big, causing a smaller non-evaluation time as compared to Spearmint (see Table 7.4).

Table 7.4 also shows the mean percentage of the non-evaluation time with respect to the total time (term inside the parenthesis). We want the percentage to be as small as possible. Spearmint consumes the highest percentage of the non-evaluation time. One must use Spearmint only when black-box evaluation time is very large. In that way, non-evaluation time will cover only a small percentage of total time, and the disadvantage gets hidden in the grand scheme. For example, black-box evaluation time for 15-CNN problem set is small as compared to 19-CNN. Therefore *Spearmint* takes 85.94% of the total time. However, it takes 28.77% of total time for 19-CNN. This makes *Spearmint* more useful for 19-CNN case than 15-CNN one. So, we conclude that for our given problem set it is not suggested to use *Spearmint* because the good performance with regard to the accuracy is undermined by high non-evaluation time.

7.2.4. Stochastic Optimization

In the discussion till now, we manually fixed the seed for the weight initialization of the neural network. This ensures same validation accuracy every time we train the network. With reference to the optimizer, this ensures that the black-box is deterministic. If we keep the seed random, then the black-box becomes stochastic. In this sub-section, we present and discuss the results of stochastic hyperparameter optimization.

In this section, we only consider the 8-CNN problem of the second category. We optimize the expected value of the sample ($\mathcal{R}_0^b$). The robustness measure is calculated over a sample size $N = 100$. We adapt the hyperparameters of the gaussian process after 10th and 50th black-box evaluation (GP adaption step).

Figure 7.6 shows the evolution of the percentage mean validation error and the noise

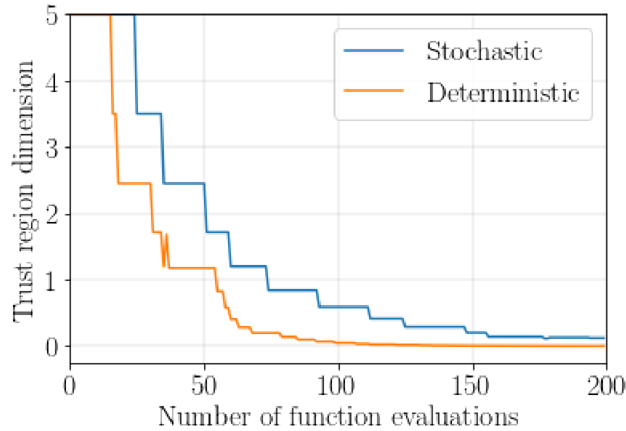


Figure 7.7.: Trust region dimension vs number of function evaluations for stochastic 8-CNN problem over sample size of $N = 100$.

against the number of function evaluations for the stochastic 8-CNN problem. After 200 black-box evaluations, we obtain the set of hyperparameters that yield expected error of 1.48 %. From the table 7.3, we can observe that the mean validation error for the deterministic case is 0.75. We obtain higher mean validation error from the stochastic optimization when compared to the deterministic one. This is because in the stochastic cases we are trying to optimize the mean of validation error over all the random seeds whereas in deterministic cases we optimize for a particular seed. The given set of hyperparameters might perform well for a particular random seed but may not be the optimum for another seed.

We can also see from the Figure 7.6 that the noise of the robustness measure decreases with increase in the number of function evaluations. This can explained using equation 3.13. As we evaluate more points around optimum point, the standard deviation of gaussian process surrogate around the optimum point decreases. As the standard deviation decreases, more weight is given to the gaussian process surrogate. This in turn decreases the noise of the robustness measure (see equation 3.12).

We also observe that the mean validation error falls to 1.1% around 80th iteration and then rises up. As mentioned in Section 3.2.2, the robustness measure is calculated as a weighted linear combination of sample robustness measure and the value from the gaussian process surrogate (see equation 3.12). As discussed in previous paragraph, the weight of the gaussian process in robustness measure increases with the increase in the number of function evaluations. We observe that the value of the gaussian process is lower than the sample robustness measure. The value of gaussian process is causing the final robustness measure to increase.

Figure 7.7, shows the trust region dimension with respect to the number of function evaluations. We observe that the final trust-region dimension after 200 function evaluations for the stochastic optimization is 0.122. From table 7.2, we observe that the final radius for the deterministic 8-CNN problem (second category) is 1.2×10^{-4}. As discussed in Section 3.2.3, lower bound of the trust region is governed by the noise. We have a finite non-zero noise which limits the trust-region dimension from falling below a certain value

governed by the equation 3.16. So, the final trust-region dimension from the stochastic case is higher than that of the deterministic case.

Part IV.

Conclusion

8. Conclusion and Future Works

In this chapter, we provide the concluding remarks to this thesis. Firstly, we discuss the major outcomes of this thesis in Section 8.1. We also provide an overview about the performance of mixed integer SNOWPAC benchmarked against some of the existing optimization tools. In Section 8.2, we provide the possible future works that can be done over this topic.

8.1. Outcomes

In this thesis, we gave a brief overview of SNOWPAC. We were able to modify SNOWPAC to solve the mixed integer problems. Then we introduced the problem of neural network hyperparameter optimization and provided an overview of some existing methods to optimize the hyperparameters. Then, we compared the performance of SNOWPAC against some of the existing optimization tools. SNOWPAC showed the best performance profiling graph amongst all for both deterministic and stochastic box-constrained mixed integer problems.

Thereafter, we tested performance of SNOWPAC for deterministic neural network hyperparameter optimization against exiting methods. We conducted various sets of experiments and compared the validation error and non-evaluation time. We found out that, SNOWPAC found least mean validation error in four out of eight problems (50% of cases) (refer Table 7.1 and 7.3). HORD and HORD-ISP found hyperparameters with slightly higher validation error. All these three mentioned methods are based on the trust-region derivative free optimization method. The algorithm is designed to converge to a local minimum. However, there can be many local minimums for hyperparameter optimization problem. This is why aforementioned methods did not find the most optimal set of hyperparameters in some scenarios.

Bayesian optimization method has widely been used to find the optimal value of hyperparameters because it is designed to find the global minimum. So, we also included three bayesian optimization tools, namely Spearmint, TPE and SMAC. Spearmint finds better optimum point than SNOWPAC in just two out of eight setups (see Table 7.1 and 7.3). Moreover, it has higher non-evaluation time too (see Table 7.4). When compared to SNOWPAC, non-evaluation time of Spearmint is approximately 8500 times greater. Therefore, it must be used only when the black-box evaluation is computationally very expensive, so that percentage of the non-evaluation time is smallest (see the values inside parenthesis of Table 7.4). TPE and SMAC are bayesian optimization methods modified to make it faster. However, SNOWPAC performs better than both TPE and SMAC in terms of obtaining more optimal set of hyperparameters.

We also carried out stochastic hyperparameter optimization. We considered the expected value of the validation accuracy as the robustness measure to be optimized. But

the best mean validation error was higher than that of deterministic case because a given set of hyperparameter might yield low error for some random seed but higher for others. We also observe less noise with increase in the number of function evaluations because of the gradual improvement in the gaussian process. The trust-region dimension for the stochastic case is higher than the deterministic case because of the lower bound caused due to the noise (see equation 3.16).

We need to manually provide certain parameters to SNOWPAC like initial starting point, beginning trust-region dimension, etc. The final performance depends upon these values and must be chosen wisely. If wrongly chosen this can lead to a poor result. For example, if the starting trust-region dimension is very small as compared to the domain and the function is non-monotonic then SNOWPAC will not explore the whole domain and may not converge to a good minimum. However, this can also be used to control the algorithm by preventing it to explore regions where the user knows that there cannot be any optimum points. Therefore, these parameters must be set carefully.

8.2. Future Works

As explained in last paragraph of subsection 7.2.1, the range of certain hyperparameters affects the quality of the surrogate model. In future we can try and improve this by independent initialization and modification of the box-dimension along every design parameter. We have used the Branch and Bound algorithm for the surrogate optimization. There are more efficient algorithms to solve the mixed integer quadratic optimization problem. We can implement those algorithms in our code. This can further improve the non-evaluation time. We can run more rigorous benchmarks on existing code to validate the performance of the mixed integer SNOWPAC. Moreover, we did not check the behavior for mixed integer SNOWPAC for mixed integer constrained problems. In future work, we can do these tests better validation of code.

All the experimental setups in Chapter 7 were box-constrained. In future, we can design a constrained hyperparameter optimization. For example, we can add all the three dimensions of the convolution kernel and the kernel-stride in the list of hyperparameters to be optimized. Then we can add a constraint on the stride that it cannot exceed the width and the height of the kernel. This convert the setup into a constrained optimization problem. HORD and HORD-ISP are not designed to solve constrained optimization problems giving SNOWPAC an edge over other methods.

Moreover, we can use the results obtained from the stochastic optimization as the starting point for the deterministic cases. This will provide a good a priori information to the deterministic solver and we expect to find better results.

Appendix

A. Benchmark Optimizations Problems

A.1. Box-Constrained Problems

1. ***Sphere Problem [110]:*** This is a simple convex problem. We use this problem with various number of design parameters. It is turned mixed-integer by imposing integer constraint on the first half number of design parameters. Mathematically, it is stated as:

$$\begin{aligned} \min_x \quad & f(x) = \sum_{i=1}^{n} x_i^2, \\ s.t. \quad & -7 \leq x_i \leq 7, \quad i = 1, ..., n \\ & x_j \in \mathbb{Z}, \quad j = 1, ..., \frac{n}{2} \\ & x_k \in \mathbb{R}, \quad k = \frac{n}{2} + 1, ..., n. \end{aligned} \tag{A.1}$$

This optimization problem has one global minimum at $x^* = (0, ..., 0)$ with $f(x^*) = 0$.

2. ***Ackley's Function [3, 105, 110]:*** This is a multi-modal function. In this thesis, we have used 8-dimensional Ackley's Function with 3 integer design parameters and 5 real design parameters. Mathematically, it is stated as:

$$\begin{aligned} \min_x \quad & f(x) = -20exp\left[-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right] - exp\left[\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)\right] + 20 + e \\ s.t. \quad & -7 \leq x_i \leq 7, \quad i = 1, ..., 8 \\ & x_j \in \mathbb{Z}, \quad j = 1, 2, 3 \\ & x_k \in \mathbb{R}, \quad k = 4, ..., n. \end{aligned} \tag{A.2}$$

where $n = 8$. This optimization problem has many local minimums, with global minimum at $x^* = (0, ..., 0)$ with $f(x^*) = 0$.

3. ***Weighted De Jong's Function [110]:*** This is a uni-modal functions. It is also called *hyper-ellipsoid function*. Weighting of axis, makes this optimization problem a little difficult to solve. We use 5-dimensional De Jong's function and turn it into mixed-integer by imposing integer constraint on the first three design parameters. Mathematically, it is stated as:

$$\begin{aligned} \min_x \quad & f(x) = \sum_{i=1}^{n} i x_i^2, \\ s.t. \quad & -7 \leq x_i \leq 7, \quad i = 1, ..., n \\ & x_j \in \mathbb{Z}, \quad j = 1, 2, 3 \\ & x_k \in \mathbb{R}, \quad k = 4, , , , n. \end{aligned} \tag{A.3}$$

where $n = 5$. This optimization problem has one global minimum at $x^* = (0, ..., 0)$ with $f(x^*) = 0$.

4. ***Bohachevsky Problem 1 (BF1) [3, 22]:*** This is a multi-modal function. In this thesis, we have imposed integer constraint on first variable. Mathematically, it is stated as:

$$\begin{aligned} \min_{x} \quad & f(x) = x_i^2 + 2x_2^2 - 0.3cos(3\pi x_1) - 0.4cos(4\pi x_2) + 0.7 \\ s.t. \quad & -7 \leq x_i \leq 7, \quad i = 1, 2 \\ & x_1 \in \mathbb{Z} \quad x_2 \in \mathbb{R}. \end{aligned} \tag{A.4}$$

This optimization problem has many local minimums, with global minimum at $x^* = (0, 0)$ with $f(x^*) = 0$.

5. ***Bohachevsky Problem 2 (BF2) [3, 22]:*** This is a multi-modal function. In this thesis, we have imposed integer constraint on first variable. Mathematically, it is stated as:

$$\begin{aligned} \min_{x} \quad & f(x) = x_i^2 + 2x_2^2 - 0.3cos(3\pi x_1)cos(4\pi x_2) + 0.3 \\ s.t. \quad & -7 \leq x_i \leq 7, \quad i = 1, 2 \\ & x_1 \in \mathbb{Z} \quad x_2 \in \mathbb{R}. \end{aligned} \tag{A.5}$$

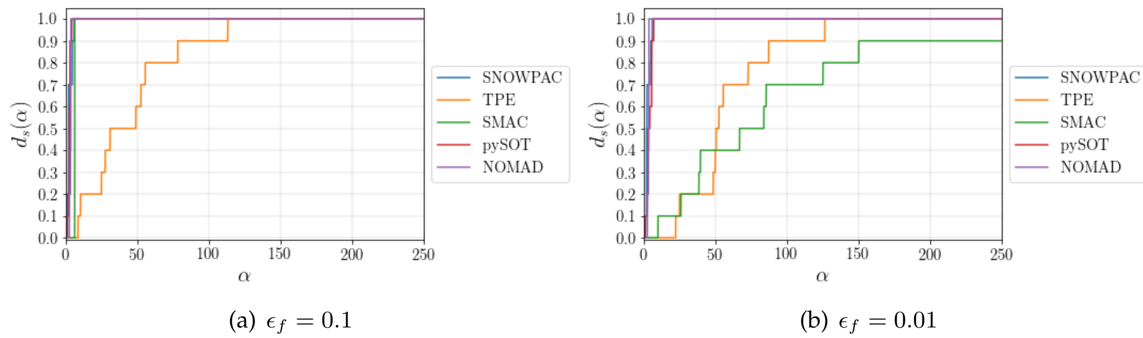
This optimization problem has many local minimums, with global minimum at $x^* = (0, 0)$ with $f(x^*) = 0$.

6. ***Griewank's Function [110]:*** This is a multi-modal function. In this thesis, we have used 10-dimensional Griewank's Function with 5 integer design parameters and 5 real design parameters. Mathematically, it is stated as:

$$\begin{aligned} \min_{x} \quad & f(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \\ s.t. \quad & -7 \leq x_i \leq 7, \quad i = 1, ..., 8 \\ & x_j \in \mathbb{Z}, \quad j = 1, , .., 5 \\ & x_k \in \mathbb{R}, \quad k = 4, ..., 8. \end{aligned} \tag{A.6}$$

where $n = 10$. This optimization problem has many local minimums, with global minimum at $x^* = (0, ..., 0)$ with $f(x^*) = 0$.

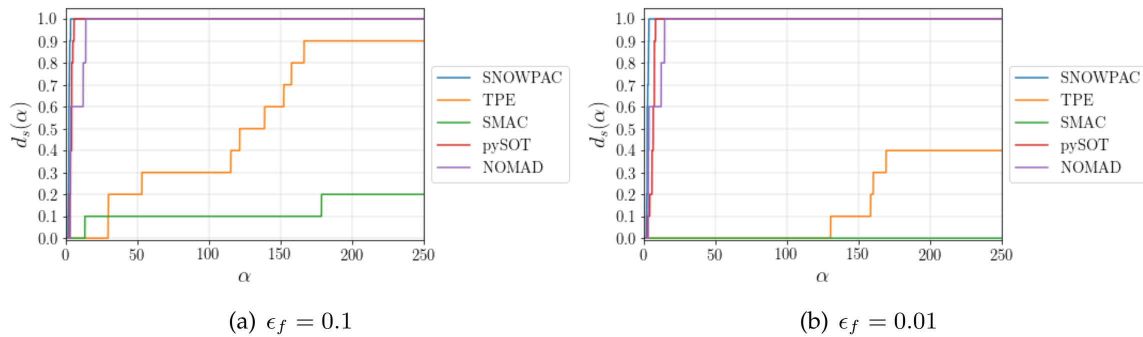
B. Optimization Benchmark Comparison Graphs



(a) $\epsilon_f = 0.1$

(b) $\epsilon_f = 0.01$

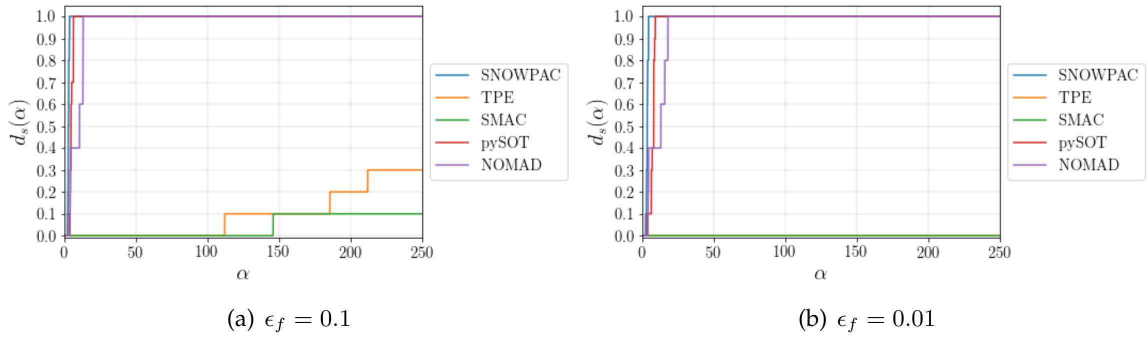
Figure B.1.: Profiling results from 2-dimensional sphere function [110] $n_c = 1, n_d = 1$.



(a) $\epsilon_f = 0.1$

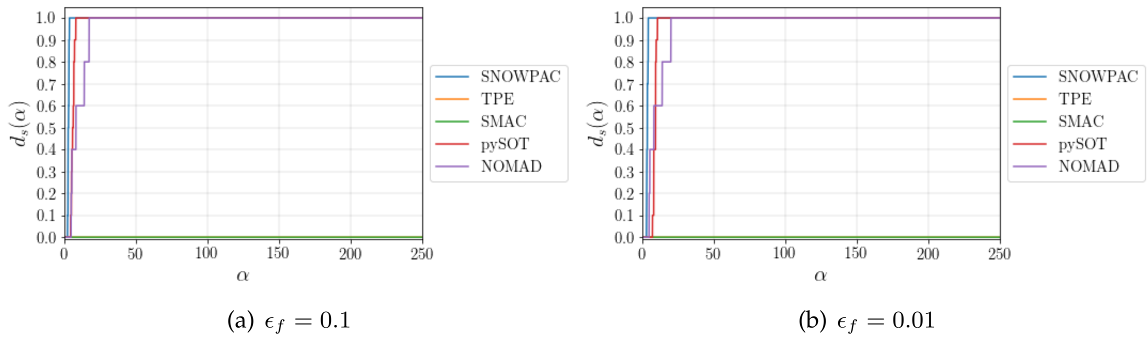
(b) $\epsilon_f = 0.01$

Figure B.2.: Profiling results from 4-dimensional sphere function [110] $n_c = 2, n_d = 2$.



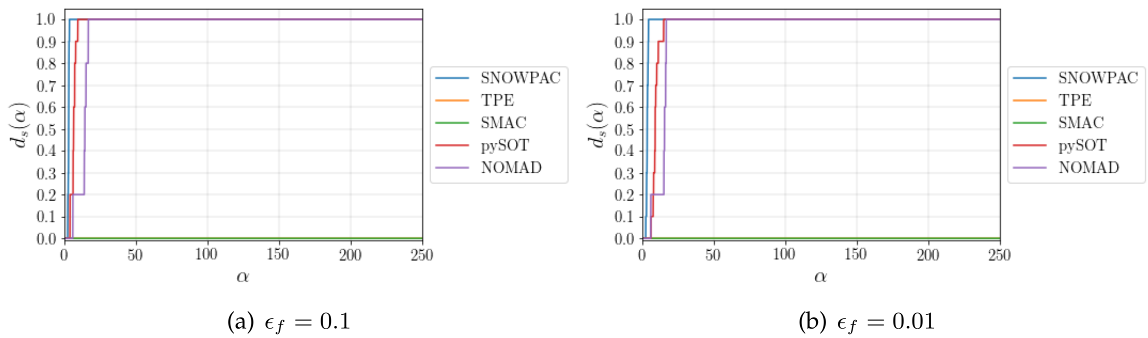
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.3.: Profiling results from 6-dimensional sphere function [110] $n_c = 3$, $n_d = 3$.



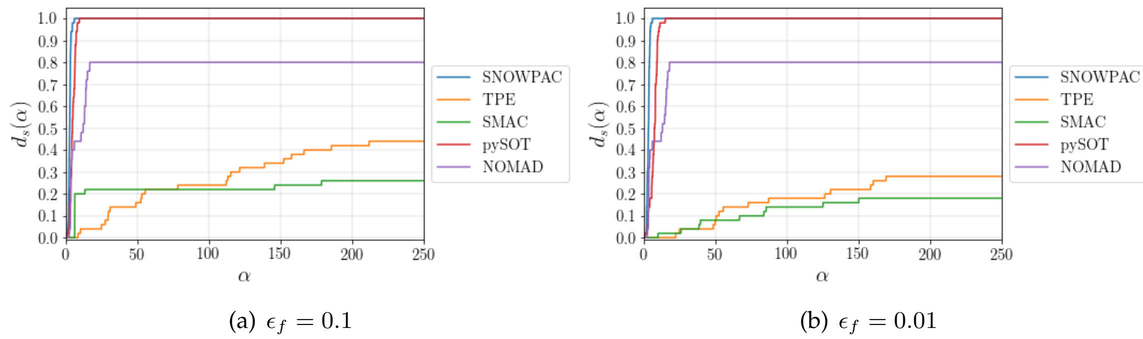
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.4.: Profiling results from 8-dimensional sphere function [110] $n_c = 4$, $n_d = 4$.



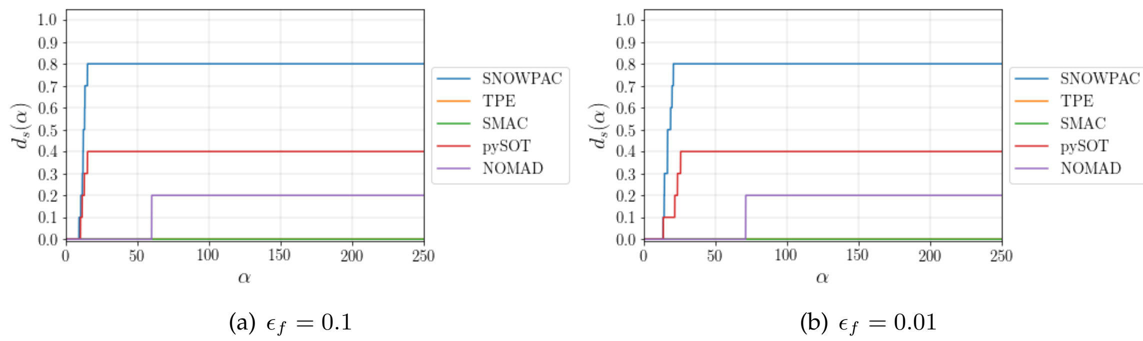
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.5.: Profiling results from 10-dimensional sphere function [110] $n_c = 5$, $n_d = 5$.



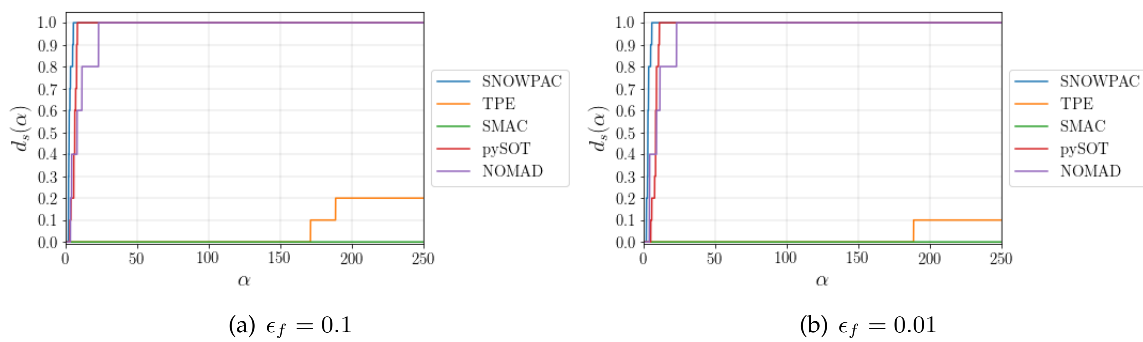
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.6.: Overall profiling result from convex function.



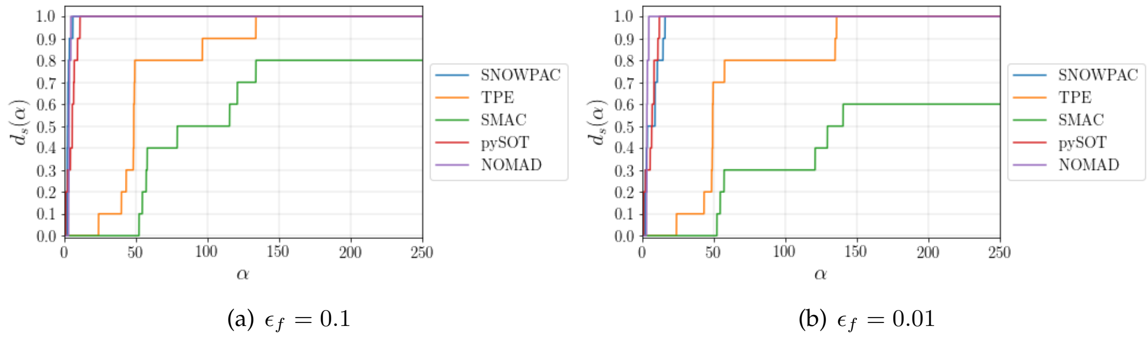
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.7.: Profiling results from Ackley's Function.



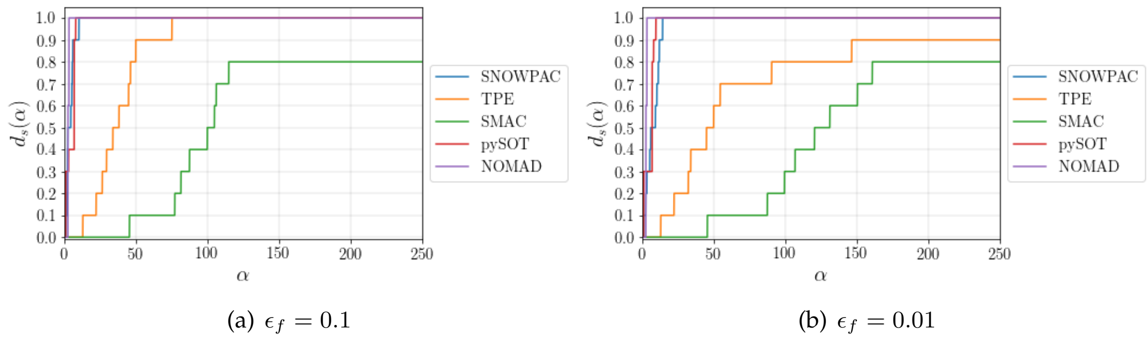
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.8.: Profiling results from Weighted De-jong Function [3].



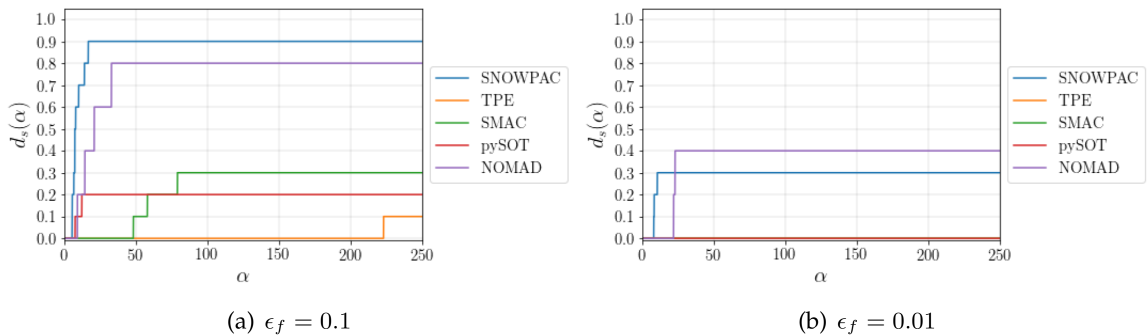
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.9.: Profiling results from Bohachevsky Problem 1 (BF1) [3, 22].



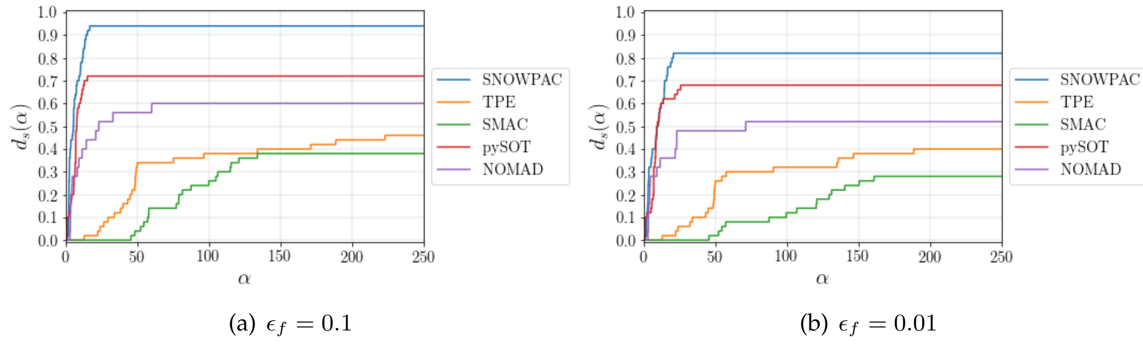
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.10.: Profiling results from Bohachevsky Problem 2 (BF2) [3, 22].



(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

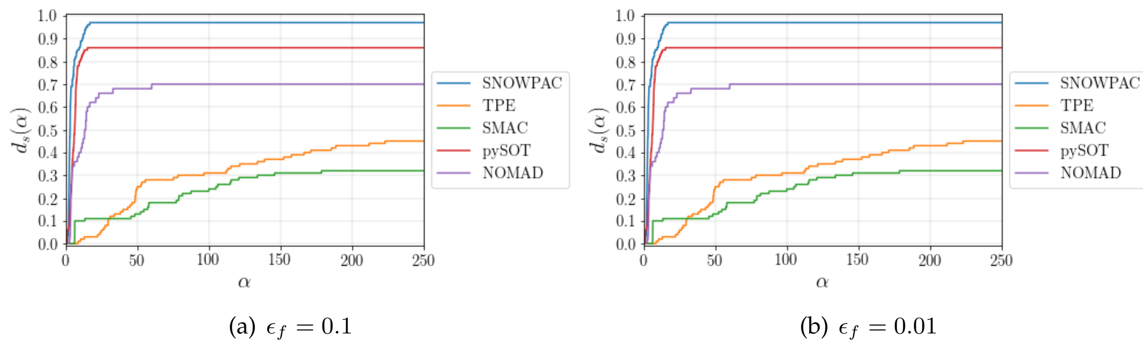
Figure B.11.: Profiling results from Griewank Function [3].



(a) $\epsilon_f = 0.1$

(b) $\epsilon_f = 0.01$

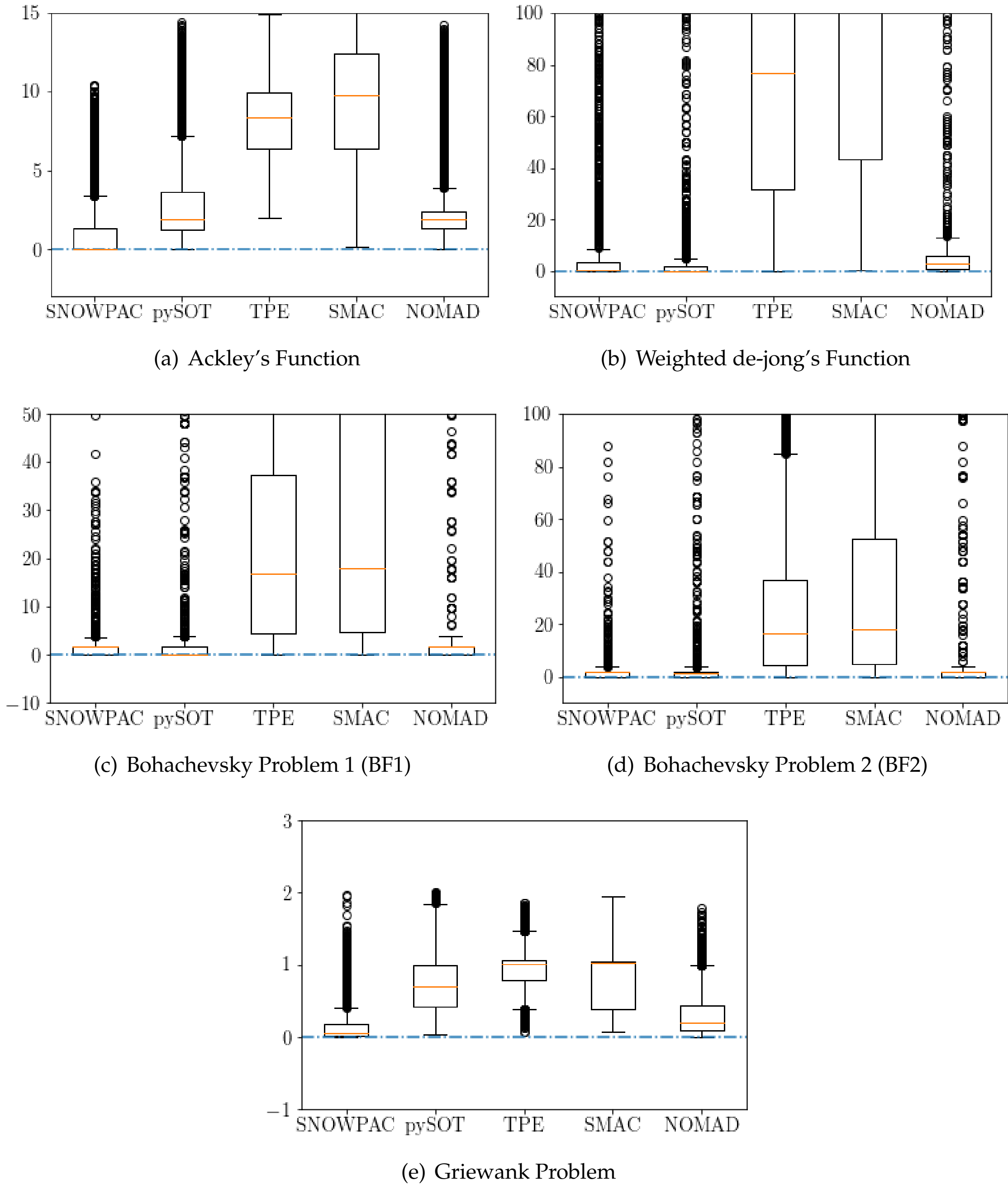
Figure B.12.: Overall profiling result from Multi-modal function.



(a) $\epsilon_f = 0.1$

(b) $\epsilon_f = 0.01$

Figure B.13.: Overall profiling result for box-constrained problems.



(a) Ackley's Function

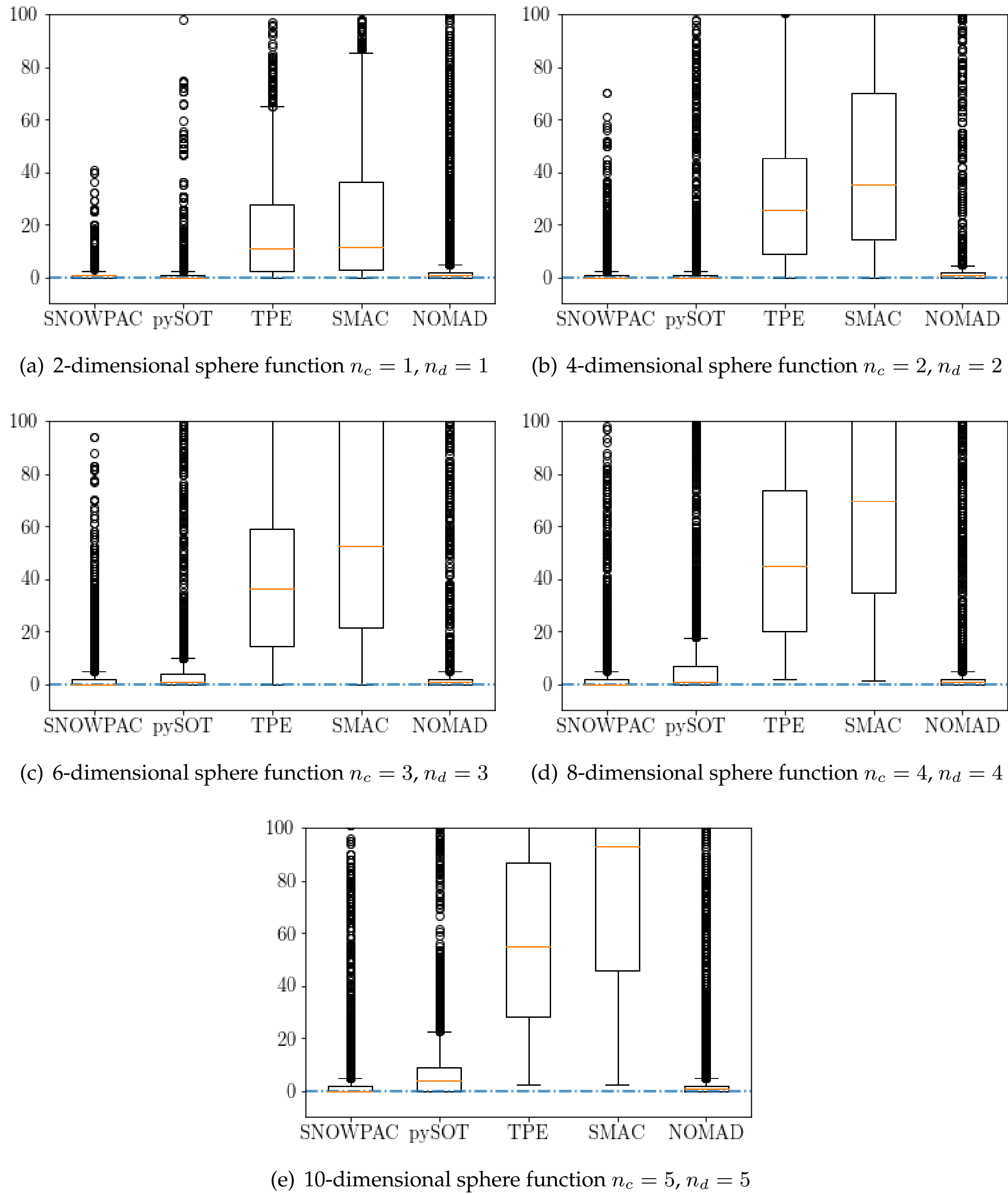
(b) Weighted de-jong's Function

(c) Bohachevsky Problem 1 (BF1)

(d) Bohachevsky Problem 2 (BF2)

(e) Griewank Problem

Figure B.14.: Box plot distribution of evaluation points for multi -modal benchmarks.



(a) 2-dimensional sphere function $n_c = 1$, $n_d = 1$

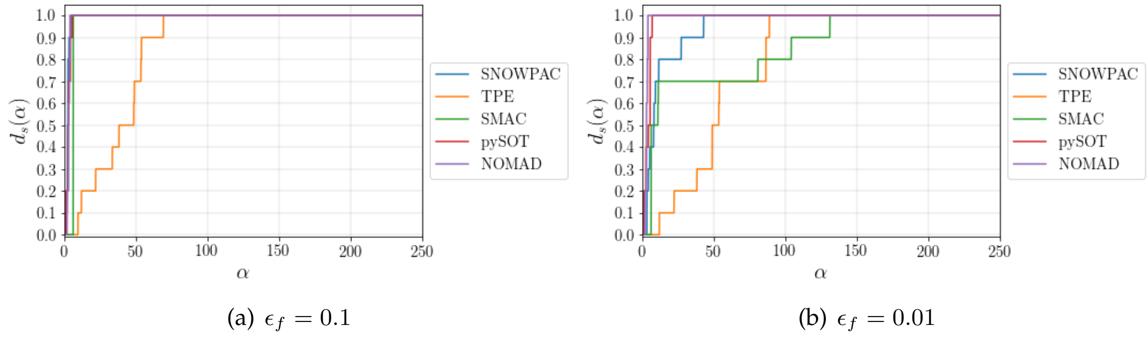
(b) 4-dimensional sphere function $n_c = 2$, $n_d = 2$

(c) 6-dimensional sphere function $n_c = 3$, $n_d = 3$

(d) 8-dimensional sphere function $n_c = 4$, $n_d = 4$

(e) 10-dimensional sphere function $n_c = 5$, $n_d = 5$

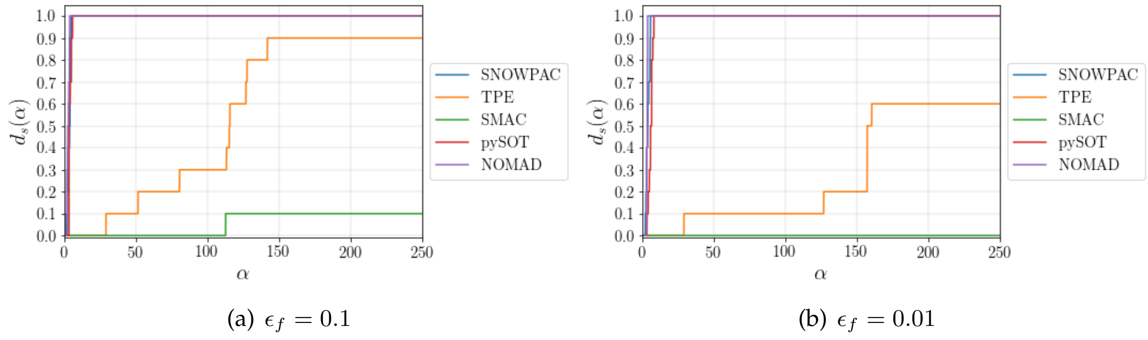
Figure B.15.: Box plot distribution of evaluation points for multi -modal benchmarks.



(a) $\epsilon_f = 0.1$

(b) $\epsilon_f = 0.01$

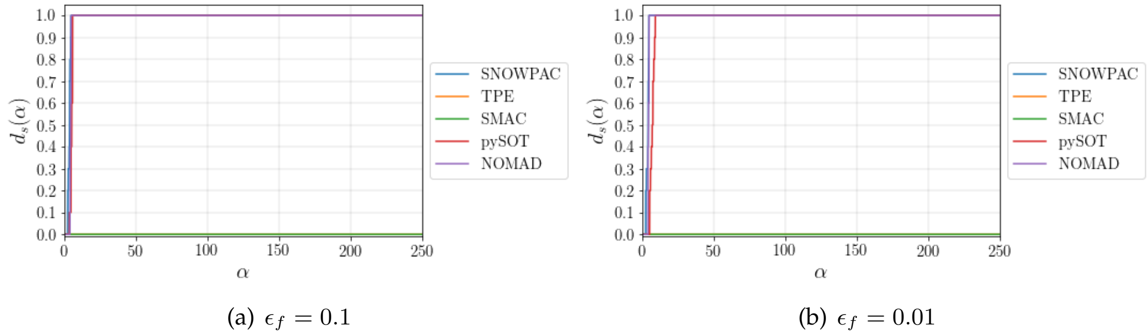
Figure B.16.: Profiling results from 2-dimensional stochastic sphere function $n_c = 1, n_d = 1$
.



(a) $\epsilon_f = 0.1$

(b) $\epsilon_f = 0.01$

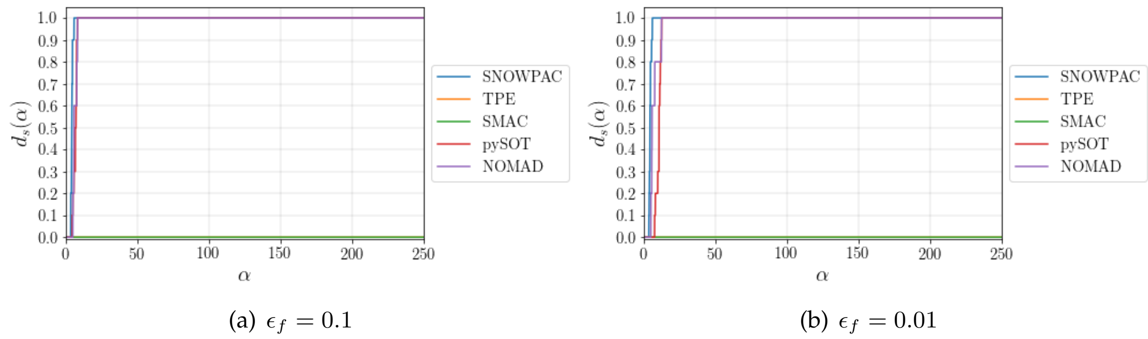
Figure B.17.: Profiling results from 4-dimensional stochastic sphere function $n_c = 2, n_d = 2$
.



(a) $\epsilon_f = 0.1$

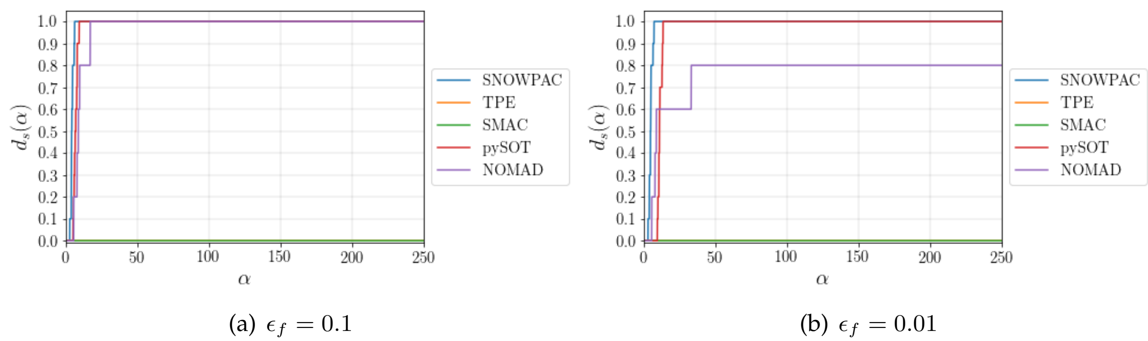
(b) $\epsilon_f = 0.01$

Figure B.18.: Profiling results from 6-dimensional stochastic sphere function $n_c = 3, n_d = 3$
.



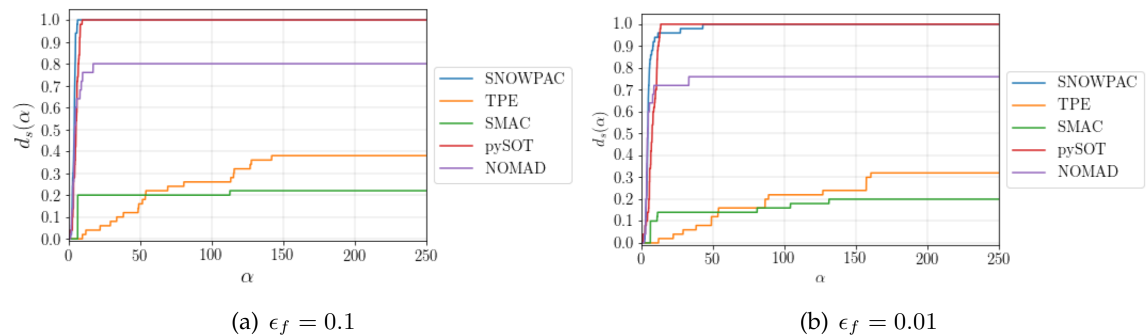
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.19.: Profiling results from 8-dimensional sphere stochastic function $n_c = 4$, $n_d = 4$.



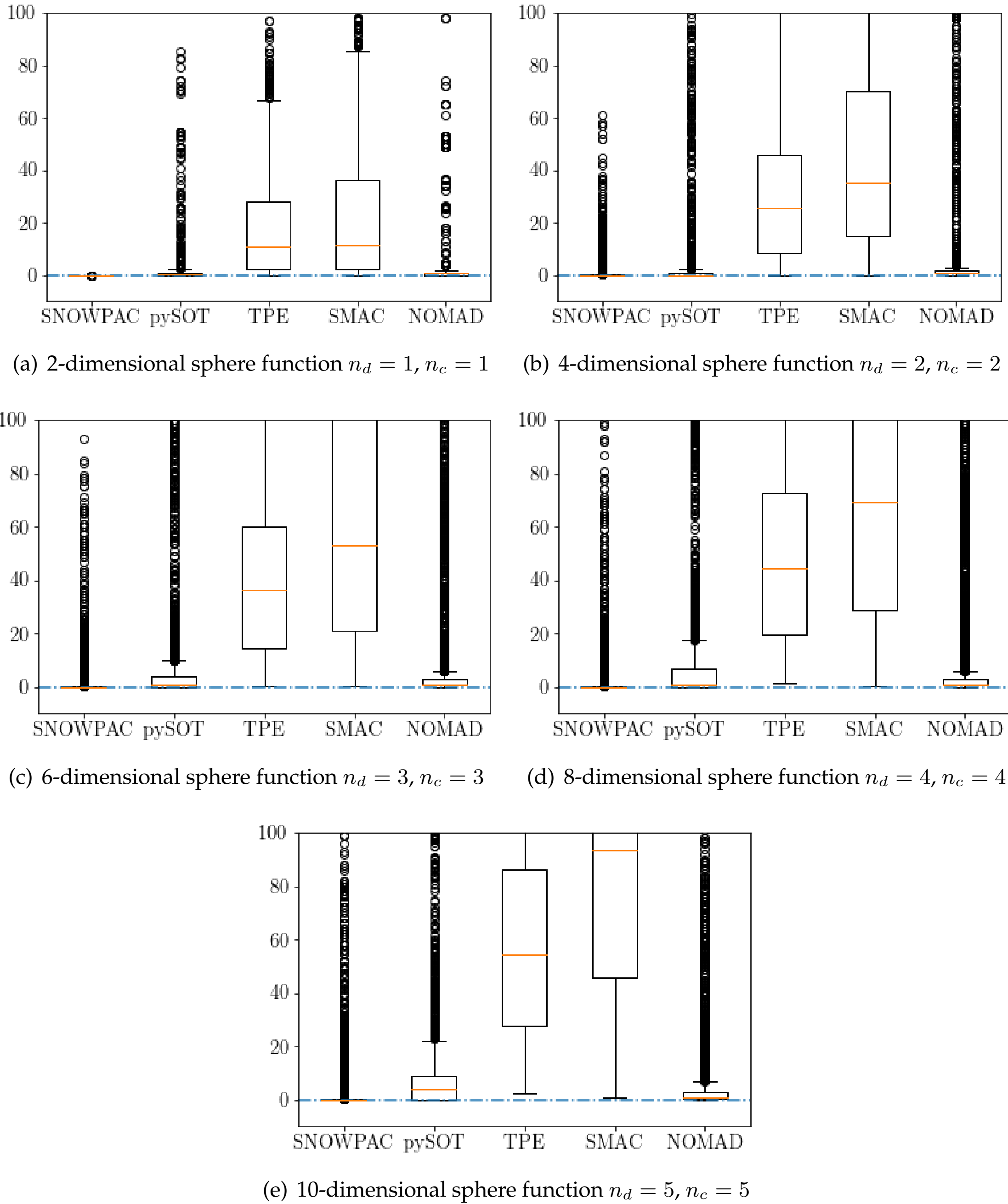
(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.20.: Profiling results from 10-dimensional stochastic sphere function $n_c = 5$, $n_d = 5$.



(a) $\epsilon_f = 0.1$ (b) $\epsilon_f = 0.01$

Figure B.21.: Overall profiling result for stochastic sphere functions.



(a) 2-dimensional sphere function $n_d = 1$, $n_c = 1$

(b) 4-dimensional sphere function $n_d = 2$, $n_c = 2$

(c) 6-dimensional sphere function $n_d = 3$, $n_c = 3$

(d) 8-dimensional sphere function $n_d = 4$, $n_c = 4$

(e) 10-dimensional sphere function $n_d = 5$, $n_c = 5$

Figure B.22.: Box plot distribution of evaluation points for mixed-integer stochastic optimization done on stochastic sphere function.

C. Hyperparameter List

Hyperparameter(λ)	Type	Range	Initial Point
Number of training Epochs	Integer	[8 , 20]	10
Number of hidden nodes	Integer	[50 - 200]	50
Learning rate of SGD	Continuous	[0.005 , 0.30]	0.1
Momentum of SGD	Continuous	[0.6 , 0.999]	0.9
Mean of Gaussian initialization	Continuous	[0.00 , 0.01]	0.00
Mean of Gaussian initialization	Continuous	[0.00 , 0.50]	0.01

Table C.1.: List of hyperparameters for first category of 6-MLP problems with range and initial point.

Hyperparameter(λ)	Type	Scaling	Range	Initial Point
Number of training Epochs	Integer	5λ	[1 , 10]	2
Number of hidden nodes	Integer	2^{λ}	[4 , 10]	5
Learning rate of SGD	Continuous	10^{λ}	[-3 , $log_{10}0.2$]	-2
Momentum of SGD	Continuous	$\lambda/10$	[6 , 9]	8
Mean of Gaussian initialization	Continuous	10^{λ}	[-5 , -2]	-4
Mean of Gaussian initialization	Continuous	10^{λ}	[-5 , -2]	-4

Table C.2.: List of hyperparameters for second category of 6-MLP problems with range and initial point.

Hyperparameter(λ)	Type	Range	Initial Point
Depth of first Conv layer	Integer	[16 , 128]	32
Depth of second Conv layer	Integer	[16 , 128]	64
Number of hidden nodes in first FC layer	Integer	[100 , 400]	200
Number of hidden nodes in second FC layer	Integer	[100 , 400]	256
Learning rate of SGD	Continuous	[0.005 , 0.30]	0.1
Momentum of SGD	Continuous	[0.6 , 0.999]	0.9
Weight decay rate	Continuous	[0.00 , 0.01]	0.0005
Learning rate decay	Continuous	[0.001 , 1.00]	0.0002

Table C.3.: List of hyperparameters for first category of 8-CNN problems with range and initial point.

Hyperparameter(λ)	Type	Scaling	Range	Initial Point
Depth of first Conv layer	Integer	2^λ	[1 , 7]	2
Depth of second Conv layer	Integer	2^λ	[1 , 7]	2
Number of hidden nodes in first FC layer	Integer	2^λ	[1 , 10]	5
Number of hidden nodes in second FC layer	Integer	2^λ	[1 , 10]	5
Learning rate of SGD	Continuous	10^λ	[-3 , $log_{10}0.3$]	-1
Momentum of SGD	Continuous	$\lambda/10$	[6 , 9]	8
Weight decay rate	Continuous	10^λ	[-5 , -2]	-4
Learning rate decay	Continuous	10^λ	[-5 , -2]	-4

Table C.4.: List of hyperparameters for second category of 8-CNN problems with range and initial point.

Hyperparameter(λ)	Type	Range	Initial Point
Mini-batch size	Integer	[32 , 512]	128
Depth of first Conv layer	Integer	[16 , 128]	32
Depth of second Conv layer	Integer	[16 , 128]	64
Number of hidden nodes in first FC layer	Integer	[100 , 400]	200
Number of hidden nodes in second FC layer	Integer	[100 , 400]	256
Learning rate of SGD	Continuous	[0.005 , 0.30]	0.1
Momentum of SGD	Continuous	[0.6 , 0.999]	0.9
Weight decay rate	Continuous	[0.00 , 0.01]	0.0005
Learning rate decay	Continuous	[0.001 , 1.00]	0.0002
α leaky ReLU in first FC Layer	Continuous	[0.00 , 0.50]	0.01
α leaky ReLU in second FC Layer	Continuous	[0.00 , 0.50]	0.01
STD of Gaussian initialization for first FC layer	Continuous	[0.00 , 0.50]	0.01
STD of Gaussian initialization for second FC layer	Continuous	[0.00 , 0.50]	0.01
STD of Gaussian initialization for first Conv layer	Continuous	[0.00 , 0.50]	0.01
STD of Gaussian initialization for second Conv layer	Continuous	[0.00 , 0.50]	0.01

Table C.5.: List of hyperparameters for first category of 15-CNN problems with range and initial point.

Hyperparameter(λ)	**Type**	**Scaling**	**Range**	**Initial Point**
Mini-batch size	Integer	2^{λ}	[4 , 8]	5
Depth of first Conv layer	Integer	2^{λ}	[1 , 7]	2
Depth of second Conv layer	Integer	2^{λ}	[1 , 7]	2
Number of hidden nodes in first FC layer	Integer	2^{λ}	[1 , 10]	5
Number of hidden nodes in second FC layer	Integer	2^{λ}	[1 , 10]	5
Learning rate of SGD	Continuous	10^{λ}	[-3 , $log_{10}0.3$]	-1
Momentum of SGD	Continuous	$\lambda/10$	[6 , 9]	8
Weight decay rate	Continuous	10^{λ}	[-5 , -2]	-4
Learning rate decay	Continuous	10^{λ}	[-5 , -2]	-4
α leaky ReLU in first FC Layer	Continuous	$\lambda/10$	[0 , 5]	0.1
α leaky ReLU in second FC Layer	Continuous	$\lambda/10$	[0 , 5]	0.1
STD of Gaussian initialization for first FC layer	Continuous	$\lambda/10$	[0 , 5]	0.1
STD of Gaussian initialization for second FC layer	Continuous	$\lambda/10$	[0 , 5]	0.1
STD of Gaussian initialization for first Conv layer	Continuous	$\lambda/10$	[0 , 5]	0.1
STD of Gaussian initialization for second Conv layer	Continuous	$\lambda/10$	[0 , 5]	0.1

Table C.6.: List of hyperparameters for second category of 15-CNN problems with range and initial point.

Hyperparameter(λ)	**Type**	**Range**	**Initial Point**
Mini-batch size	Integer	[32 , 512]	128
Depth of first Conv layer	Integer	[16 , 128]	32
Depth of second Conv layer	Integer	[16 , 128]	64
Number of hidden nodes in first FC layer	Integer	[100 , 400]	200
Number of hidden nodes in second FC layer	Integer	[100 , 400]	256
Learning rate of SGD	Continuous	[0.005 , 0.30]	0.1
Momentum of SGD	Continuous	[0.6 , 0.999]	0.9
Weight decay rate	Continuous	[0.00 , 0.01]	0.0005
Learning rate decay	Continuous	[0.001 , 1.00]	0.0002
α leaky ReLU in first FC Layer	Continuous	[0.00 , 0.50]	0.01
α leaky ReLU in second FC Layer	Continuous	[0.00 , 0.50]	0.01
STD of Gaussian initialization for first FC layer	Continuous	[0.00 , 0.50]	0.01
STD of Gaussian initialization for second FC layer	Continuous	[0.00 , 0.50]	0.01
STD of Gaussian initialization for first Conv layer	Continuous	[0.00 , 0.50]	0.01
STD of Gaussian initialization for second Conv layer	Continuous	[0.00 , 0.50]	0.01
Dropout rate for first FC layer	Continuous	[0.00 , 0.80]	0.5
Dropout rate for second FC layer	Continuous	[0.00 , 0.80]	0.5
Dropout rate for first Conv layer	Continuous	[0.00 , 0.80]	0.5
Dropout rate for second Conv layer	Continuous	[0.00 , 0.80]	0.5

Table C.7.: List of hyperparameters for first category of 19-CNN problems with range and initial point.

Hyperparameter(λ)	**Type**	**Scaling**	**Range**	**Initial Point**
Mini-batch size	Integer	2^λ	[4 , 8]	5
Depth of first Conv layer	Integer	2^λ	[1 , 7]	2
Depth of second Conv layer	Integer	2^λ	[1 , 7]	2
Number of hidden nodes in first FC layer	Integer	2^λ	[1 , 10]	5
Number of hidden nodes in second FC layer	Integer	2^λ	[1 , 10]	5
Learning rate of SGD	Continuous	10^λ	[-3 , $log_{10}0.3$]	-1
Momentum of SGD	Continuous	$\lambda/10$	[6 , 9]	8
Weight decay rate	Continuous	10^λ	[-5 , -2]	-4
Learning rate decay	Continuous	10^λ	[-5 , -2]	-4
α leaky ReLU in first FC Layer	Continuous	$\lambda/10$	[0 , 5]	0.1
α leaky ReLU in second FC Layer	Continuous	$\lambda/10$	[0 , 5]	0.1
STD of Gaussian initialization for first FC layer	Continuous	$\lambda/10$	[0 , 5]	0.1
STD of Gaussian initialization for second FC layer	Continuous	$\lambda/10$	[0 , 5]	0.1
STD of Gaussian initialization for first Conv layer	Continuous	$\lambda/10$	[0 , 5]	0.1
STD of Gaussian initialization for second Conv layer	Continuous	$\lambda/10$	[0 , 5]	0.1
Dropout rate for first FC layer	Continuous	λ	[0.00 , 0.80]	0.5
Dropout rate for second FC layer	Continuous	λ	[0.00 , 0.80]	0.5
Dropout rate for first Conv layer	Continuous	λ	[0.00 , 0.80]	0.5
Dropout rate for second Conv layer	Continuous	λ	[0.00 , 0.80]	0.5

Table C.8.: List of hyperparameters for second category of 19-CNN problems with range and initial point.

	6-MLP	**8-CNN**	**15-CNN**	**19-CNN**
Category 1	70	200	200	200
Category 2	5	5	5	5

Table C.9.: Starting half box-dimension for *SNOWPAC* for all the hyperparameter optimization problems.

D. Hyperparameter Optimization Results

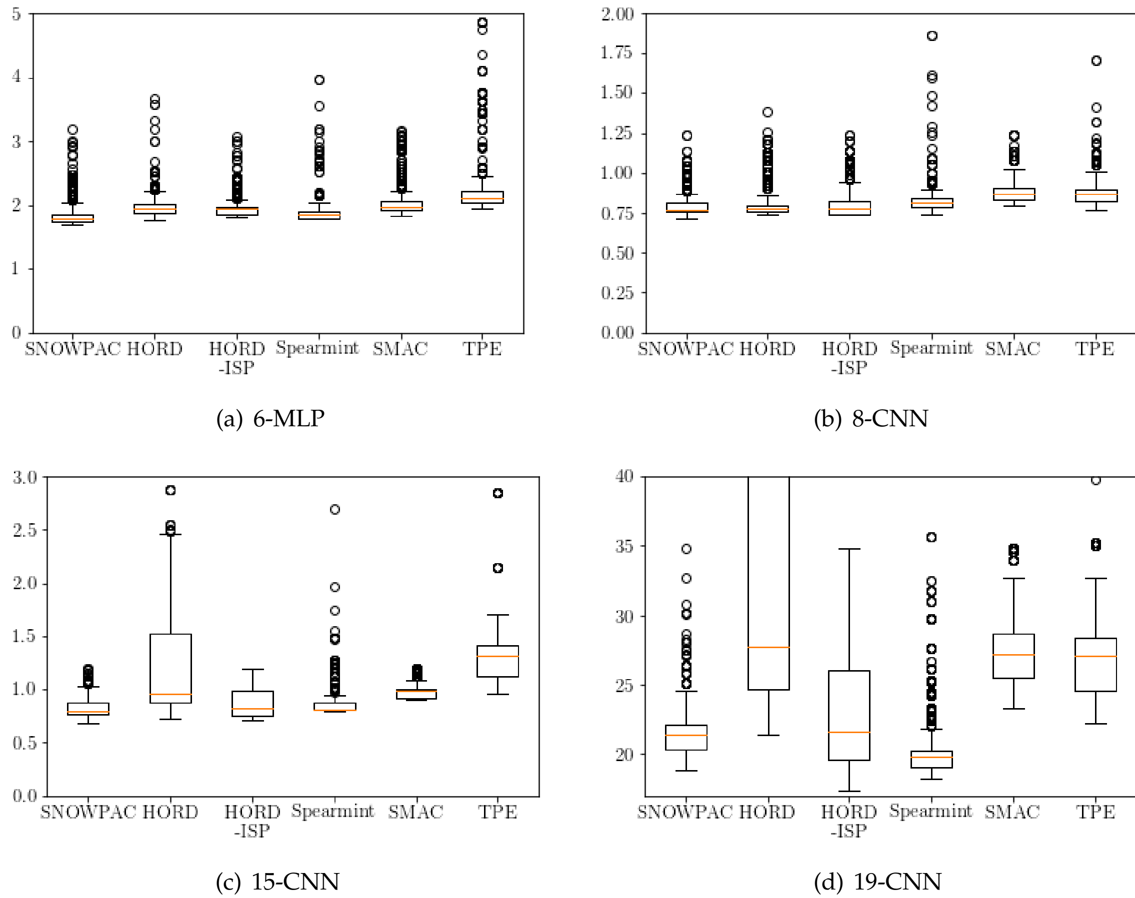
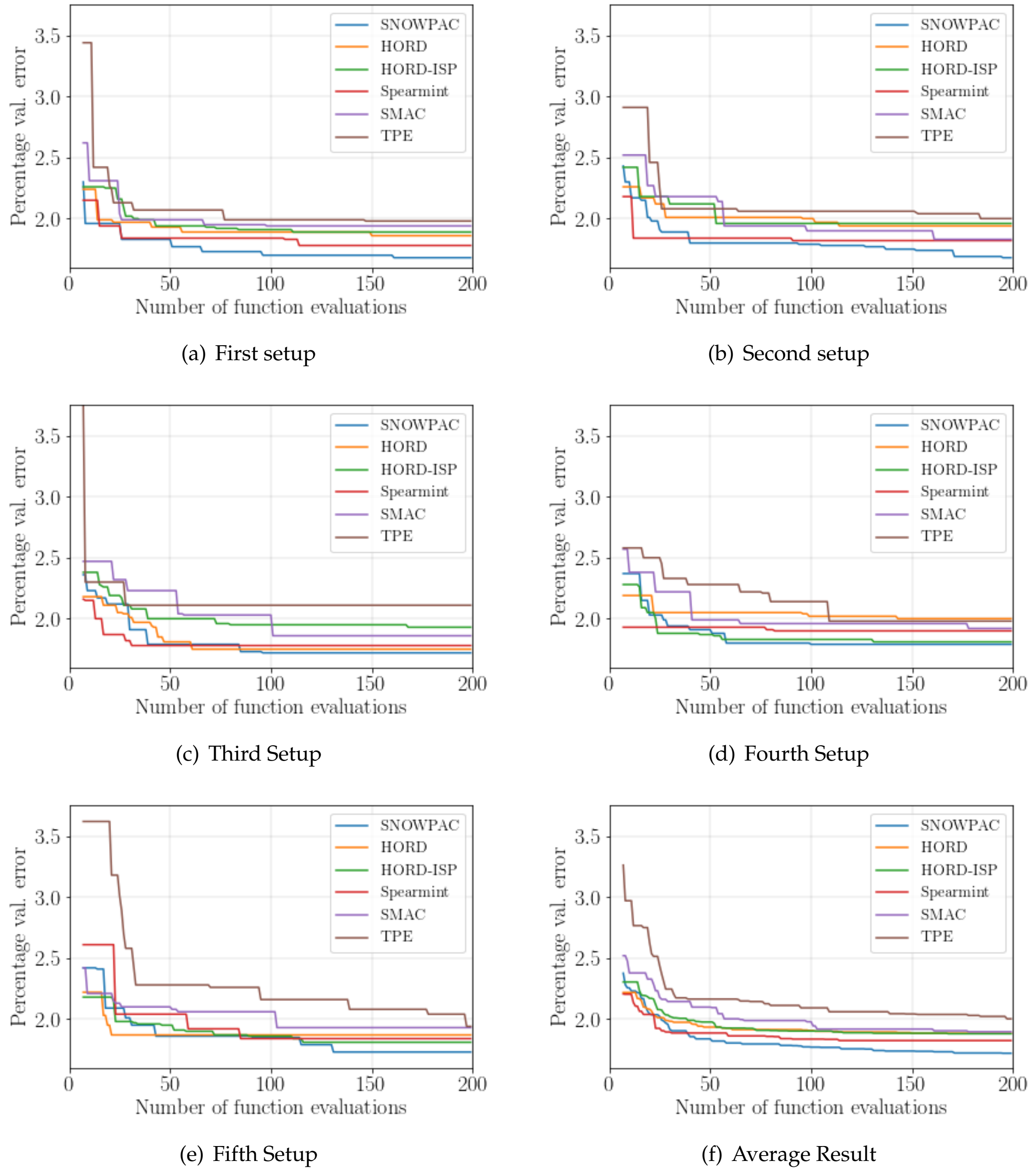


Figure D.1.: Box plot distribution of evaluation points for neural network hyperparmater optimization on unscaled setup.

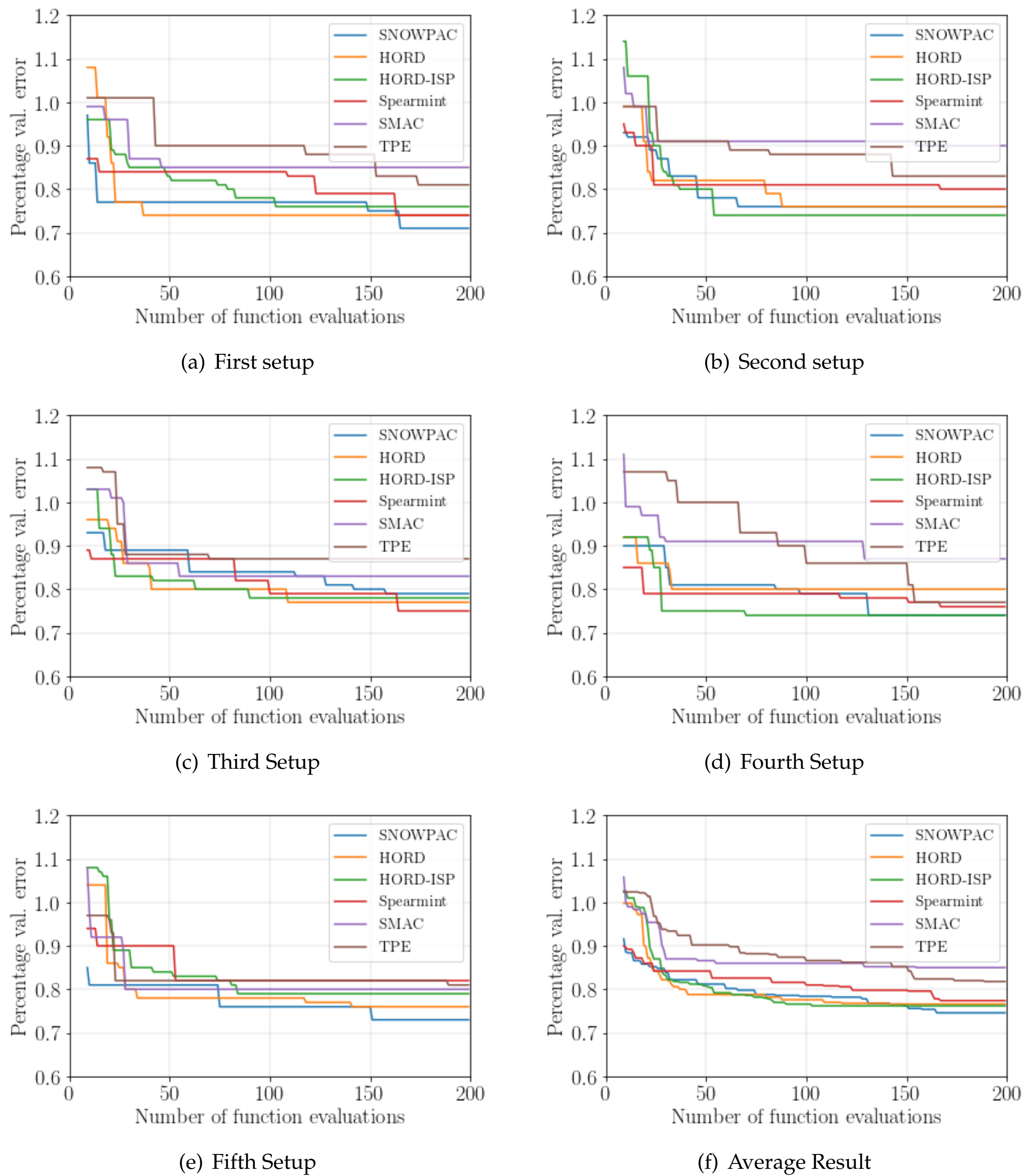


(a) First setup (b) Second setup

(c) Third Setup (d) Fourth Setup

(e) Fifth Setup (f) Average Result

Figure D.2.: Percentage validation error vs number of function evaluations for 6 hyperparamater optimization of multi-layered perceptron trained on MNIST (no scaling of hyperparameters).

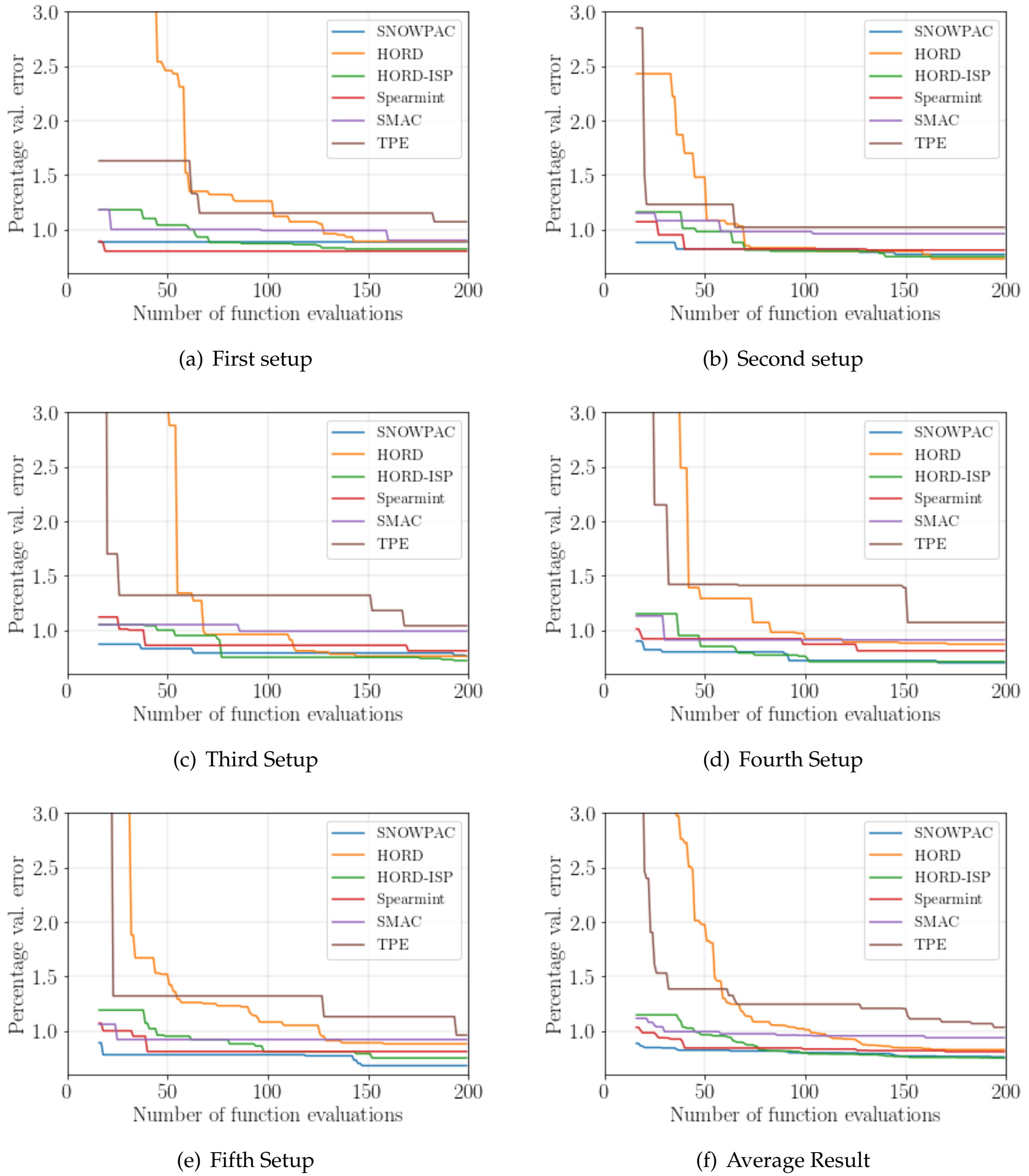


(a) First setup (b) Second setup

(c) Third Setup (d) Fourth Setup

(e) Fifth Setup (f) Average Result

Figure D.3.: Percentage validation error vs number of function evaluations for 8 hyperparmater optimization of convolutional neural network trained on MNIST (no scaling of hyperparameters).



(a) First setup

(b) Second setup

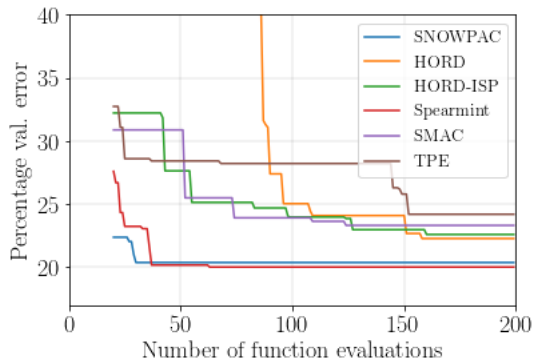
(c) Third Setup

(d) Fourth Setup

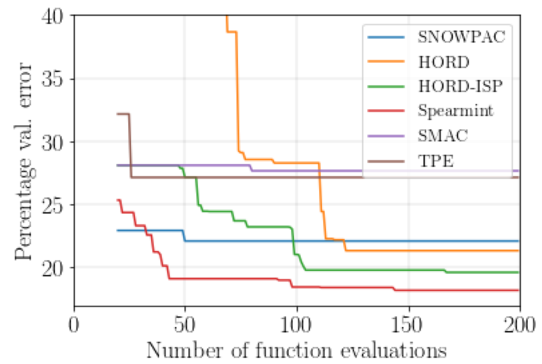
(e) Fifth Setup

(f) Average Result

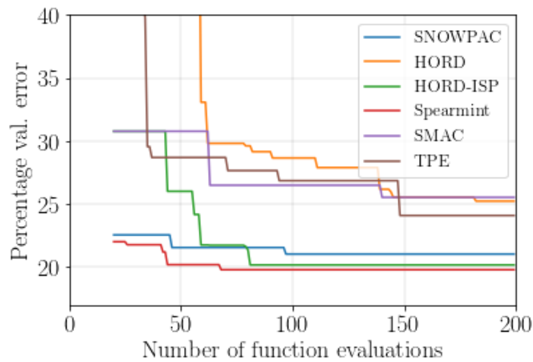
Figure D.4.: Percentage validation error vs number of function evaluations for 15 hyper-parmater optimization of convolutional neural network trained on MNIST (no scaling of hyperparameters).



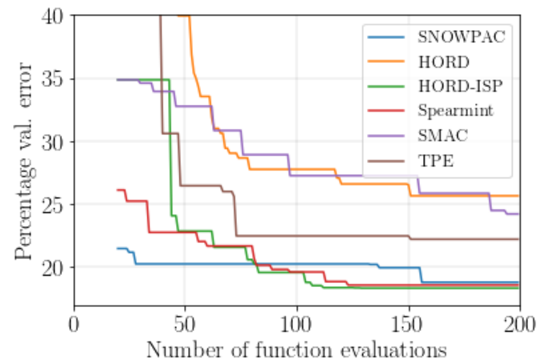
(a) First setup



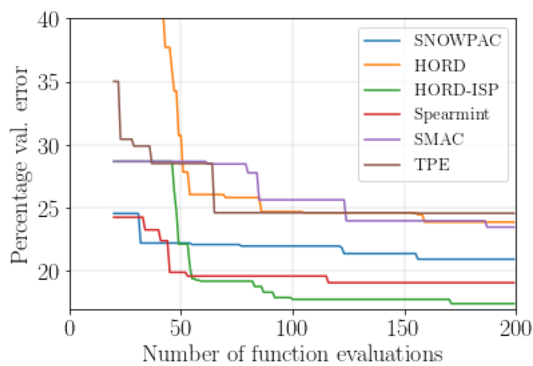
(b) Second setup



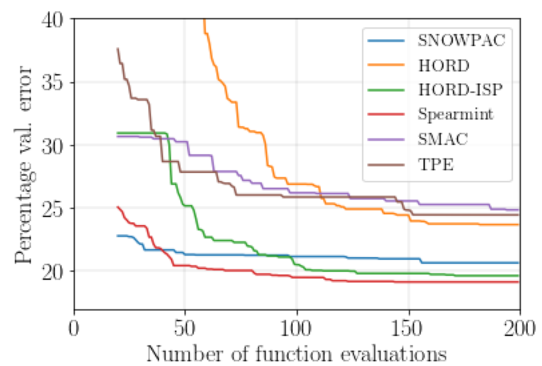
(c) Third Setup



(d) Fourth Setup

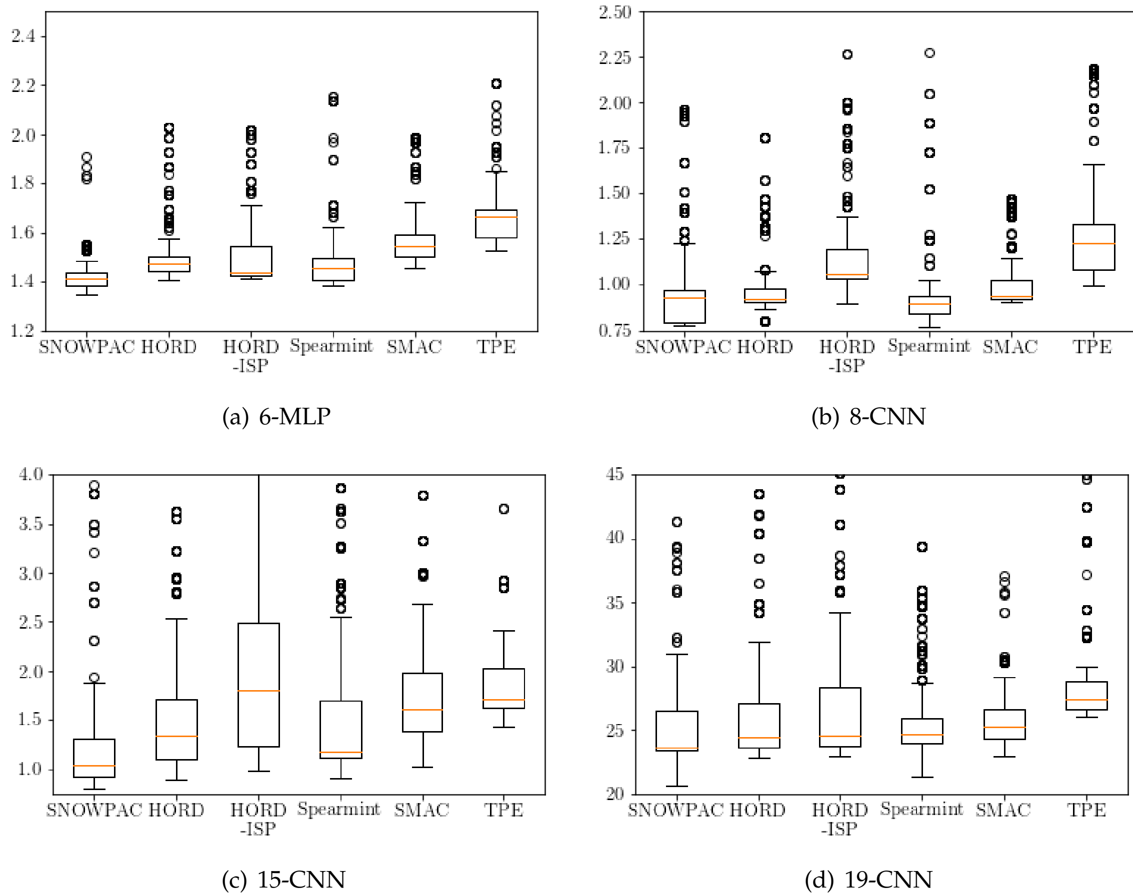


(e) Fifth Setup



(f) Average Result

Figure D.5.: Percentage validation error vs number of function evaluations for 19 hyperparmater optimization of convolutional neural network trained on CIFAR-10 (no scaling of hyperparameters).



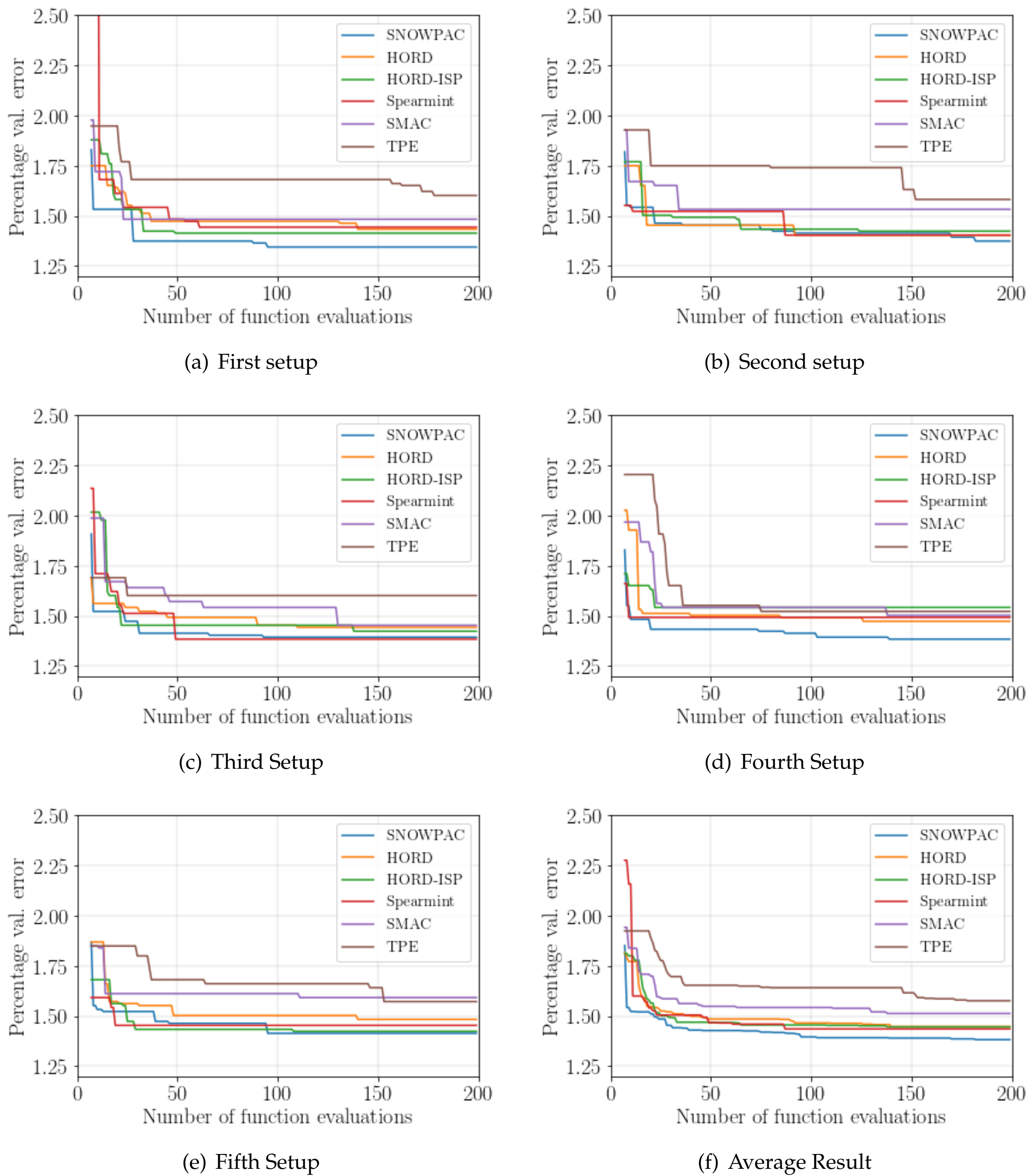
(a) 6-MLP

(b) 8-CNN

(c) 15-CNN

(d) 19-CNN

Figure D.6.: Box plot distribution of evaluation points for neural network hyperparameter optimization on scaled setup.



(a) First setup

(b) Second setup

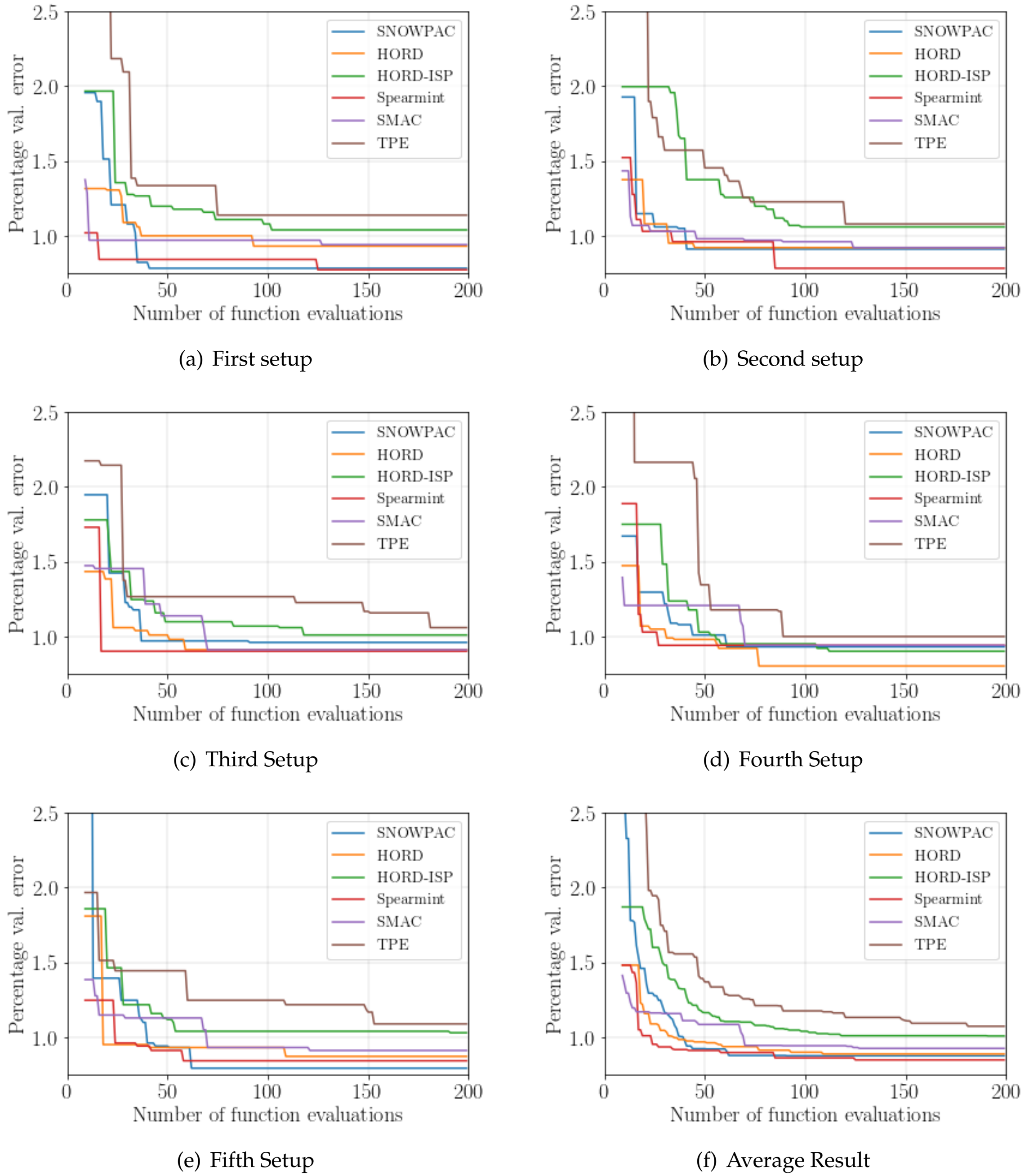
(c) Third Setup

(d) Fourth Setup

(e) Fifth Setup

(f) Average Result

Figure D.7.: Percentage validation error vs number of function evaluations for 6 hyperparmater optimization of multi-layered perceptron trained on MNIST with scaled hyperparameters.



(a) First setup

(b) Second setup

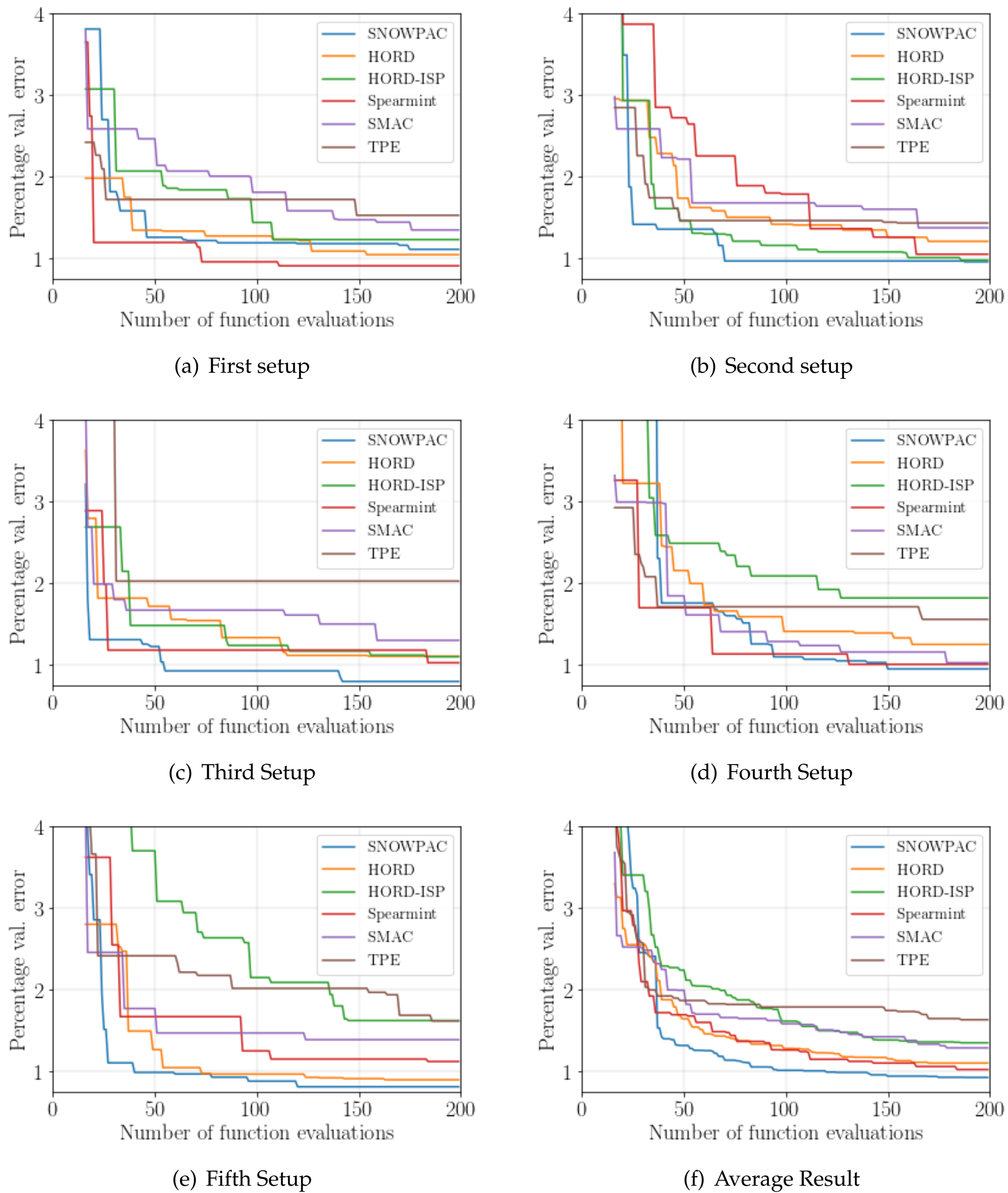
(c) Third Setup

(d) Fourth Setup

(e) Fifth Setup

(f) Average Result

Figure D.8.: Percentage validation error vs number of function evaluations for 8 hyperparamater optimization of convolutional neural network trained on MNIST with scaled hyperparameters.



(a) First setup

(b) Second setup

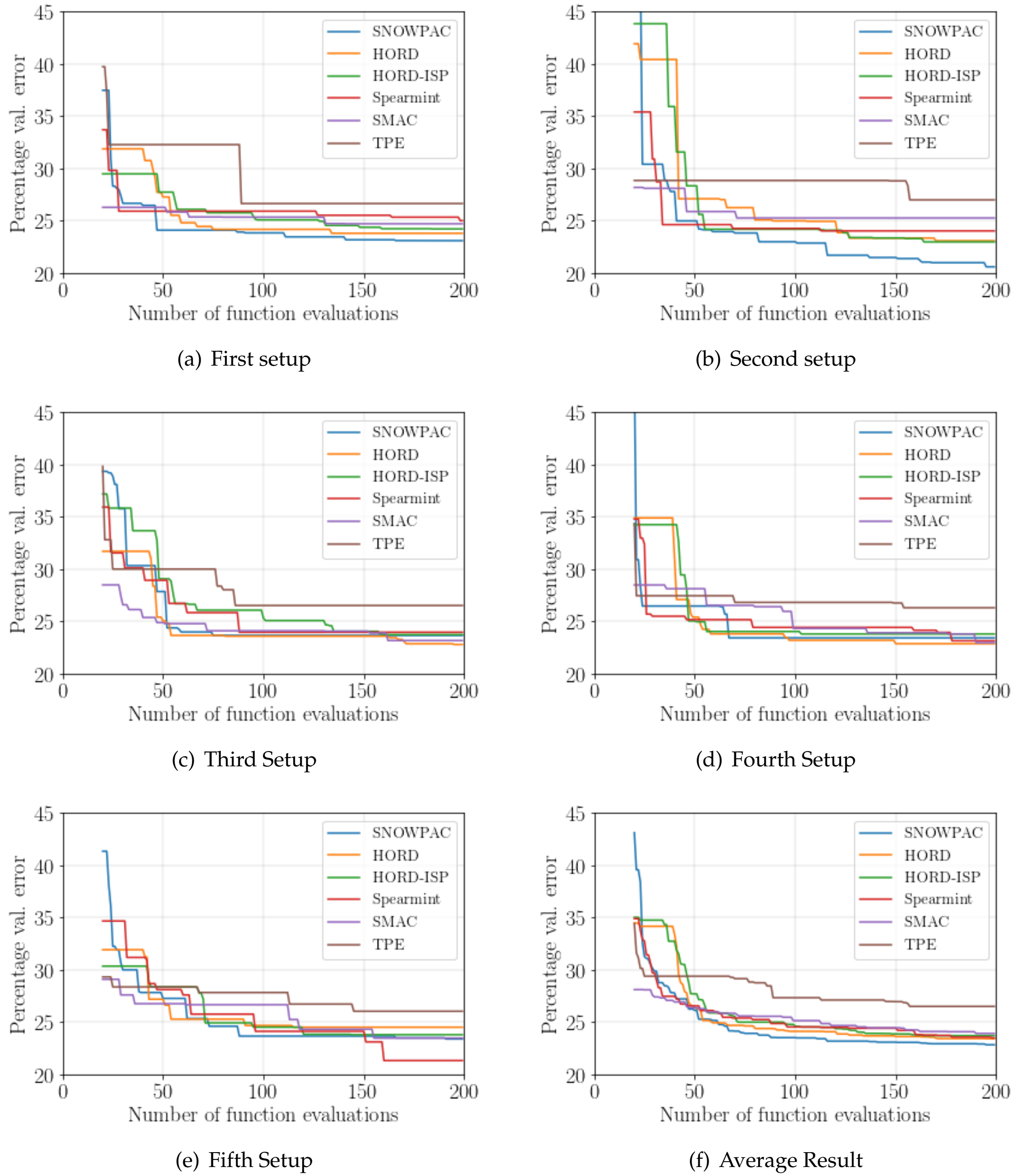
(c) Third Setup

(d) Fourth Setup

(e) Fifth Setup

(f) Average Result

Figure D.9.: Percentage validation error vs number of function evaluations for 15 hyper-parmater optimization of convolutional neural network trained on MNIST with scaled hyperparameters.



(a) First setup (b) Second setup (c) Third Setup (d) Fourth Setup (e) Fifth Setup (f) Average Result

Figure D.10.: Percentage validation error vs number of function evaluations for 19 hyperparmater optimization of convolutional neural network trained on CIFAR-10 with scaled hyperparameters.

Bibliography

[1] Carlo Acerbi and Dirk Tasche. Expected shortfall: a natural coherent alternative to value at risk. *Economic notes*, 31(2):379–388, 2002.

[2] Tobias Achterberg, Timo Berthold, and Gregor Hendel. Rounding and propagation heuristics for mixed integer programming. In *Operations research proceedings 2011*, pages 71–76. Springer, 2012.

[3] M Montaz Ali, Charoenchai Khompatraporn, and Zelda B Zabinsky. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of global optimization*, 31(4):635–672, 2005.

[4] Dave Anderson and George McNeill. Artificial neural networks technology. *Kaman Sciences Corporation*, 258(6):1–83, 1992.

[5] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

[6] Charles Audet. A survey on direct search methods for blackbox optimization and their applications. In *Mathematics without boundaries*, pages 31–56. Springer, 2014.

[7] F Augustin and YM Marzouk. Nowpac: a provably convergent derivative-free nonlinear optimizer with path-augmented constraints. *arXiv preprint arXiv:1403.1931*, 2014.

[8] F Augustin and YM Marzouk. A trust-region method for derivative-free nonlinear constrained stochastic optimization. *arXiv preprint arXiv:1703.04156*, 2017.

[9] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical programming*, 58(1-3):295–324, 1993.

[10] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[11] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[12] James Bergstra, Daniel Yamins, and David Daniel Cox. Tree Structured Parzen's Estimator. https://github.com/hyperopt/hyperopt, 2013. [Online; accessed 20-Sep-2018].

[13] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[14] Timo Berthold. Rens. *Mathematical Programming Computation*, 6(1):33–54, 2014.

[15] Shalabh Bhatnagar, HL Prasad, and LA Prashanth. *Stochastic recursive algorithms for optimization: simultaneous perturbation methods*, volume 434. Springer, 2012.

[16] Daniel Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical programming*, 74(2):121–140, 1996.

[17] Hildo Bijl, Thomas B Schön, Jan-Willem van Wingerden, and Michel Verhaegen. Online sparse gaussian process training with input noise. *stat*, 1050:29, 2016.

[18] Hildo Bijl, Jan-Willem van Wingerden, Thomas B Schön, and Michel Verhaegen. Online sparse gaussian process regression using fitc and pitc approximations. *IFAC-PapersOnLine*, 48(28):703–708, 2015.

[19] Alain Billionnet and Sourour Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Mathematical Programming*, 109(1):55–68, 2007.

[20] Alain Billionnet, Sourour Elloumi, and Amélie Lambert. Extending the qcr method to general mixed-integer programs. *Mathematical programming*, 131(1-2):381–401, 2012.

[21] Christopher M Bishop. Pattern recognition and machine learning (information science and statistics) springer-verlag new york. *Inc. Secaucus, NJ, USA*, 2006.

[22] Ihor O Bohachevsky, Mark E Johnson, and Myron L Stein. Generalized simulated annealing for function optimization. *Technometrics*, 28(3):209–217, 1986.

[23] Pierre Bonami and Jon Lee. Bonmin user manual. *Numer Math*, 4:1–32, 2007.

[24] Brian Borchers and John E Mitchell. An improved branch and bound algorithm for mixed integer nonlinear programs. *Computers & Operations Research*, 21(4):359–367, 1994.

[25] George EP Box and Kenneth B Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 13(1):1–38, 1951.

[26] Richard H Byrd, Robert B Schnabel, and Gerald A Shultz. A trust region algorithm for nonlinearly constrained optimization. *SIAM Journal on Numerical Analysis*, 24(5):1152–1170, 1987.

[27] Yanshuai Cao, Marcus A Brubaker, David J Fleet, and Aaron Hertzmann. Efficient optimization for sparse gaussian process regression. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2013.

[28] Maria Rosa Celis, JE Dennis, and RA Tapia. A trust region strategy for nonlinear equality constrained optimization. *Numerical optimization*, 1984:71–82, 1985.

[29] Krzysztof Chalupka, Christopher KI Williams, and Iain Murray. A framework for evaluating approximation methods for gaussian process regression. *Journal of Machine Learning Research*, 14(Feb):333–350, 2013.

[30] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[31] Jens Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30, 1999.

[32] Andrew R Conn, Katya Scheinberg, and Ph L Toint. On the convergence of derivative-free methods for unconstrained optimization. *Approximation theory and optimization: tributes to MJD Powell*, pages 83–108, 1997.

[33] Andrew R Conn, Katya Scheinberg, and Luís N Vicente. Global convergence of general derivative-free trust-region algorithms to first-and second-order critical points. *SIAM Journal on Optimization*, 20(1):387–415, 2009.

[34] Andrew R Conn, Katya Scheinberg, and Luis N Vicente. *Introduction to derivative-free optimization*, volume 8. Siam, 2009.

[35] C.Raffel. Bayesian Optimization. https://github.com/craffel/simple_spearmint, 2017. [Online; accessed 20-Dec-2018].

[36] Noel Cressie and Hsin-Cheng Huang. Classes of nonseparable, spatio-temporal stationary covariance functions. *Journal of the American Statistical Association*, 94(448):1330–1339, 1999.

[37] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[38] Jonathan Currie. OPTI Toolbox. https://github.com/jonathancurrie/OPTI. [Online; accessed 10-Feb-2019].

[39] D. Bindel D. Eriksson and C. Shoemaker. Surrogate Optimization Toolbox (pySOT). https://github.com/dme65/pySOT, 2015. [Online; accessed 09-Jan-2019].

[40] Robert J Dakin. A tree-search algorithm for mixed integer programming problems. *The computer journal*, 8(3):250–255, 1965.

[41] WC Davidson. Variable metric method for minimization, rep. *ANL 5990Argonne Nat. Lab., Argonne, Illinois*, 1959.

[42] Etienne de Klerk et al. Stories about maxima and minima. Technical report, Tilburg University, School of Economics and Management, 2010.

[43] Ian Dewancker, Michael McCourt, Scott Clark, Patrick Hayes, Alexandra Johnson, and George Ke. A stratified analysis of bayesian optimization methods. *arXiv preprint arXiv:1603.09441*, 2016.

[44] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[45] Oliver Exler and Klaus Schittkowski. A trust region sqp algorithm for mixed-integer nonlinear programming. *Optimization Letters*, 1(3):269–280, 2007.

[46] Richard Fitzpatrick. Euclid's elements of geometry. 2007.

[47] R Fletcher. Second order corrections for non-differentiable optimization. In *Numerical analysis*, pages 85–114. Springer, 1982.

[48] Christodoulos A Floudas. *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

[49] Yarin Gal and Richard Turner. Improving the gaussian process sparse spectrum approximation by representing uncertainty in frequency inputs. In *International Conference on Machine Learning*, pages 655–664, 2015.

[50] Carl Friedrich Gauss. Works volume ii. *Society of the Sciences, G "o ttingen, pages = 219-222, year = 1863*.

[51] Marc G Genton. Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312, 2001.

[52] Torkel Glad and Allen Goldstein. Optimization of functions whose values are subject to small errors. *BIT Numerical Mathematics*, 17(2):160–169, 1977.

[53] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[54] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[55] Charles W Groetsch. *Inverse problems: activities for undergraduates*, volume 12. Cambridge University Press, 1999.

[56] Ignacio E Grossmann and Zdravko Kravanja. Mixed-integer nonlinear programming: A survey of algorithms and applications. In *Large-scale optimization with applications*, pages 73–100. Springer, 1997.

[57] Omprakash K Gupta and A Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management science*, 31(12):1533–1546, 1985.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[59] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. https://github.com/automl/SMAC3. [Online; accessed 06-Jan-2019].

[60] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[61] Ilija Ilievski, Taimoor Akhtar, Jiashi Feng, and Christine Annette Shoemaker. Efficient hyperparameter optimization for deep learning algorithms using deterministic rbf surrogates. In *AAAI*, pages 822–829, 2017.

[62] H. Larochelle J. Snoek and R.P. Adams. Bayesian Optimization. https://github.com/HIPS/Spearmint, 2016. [Online; accessed 20-Sep-2018].

[63] Yulei Jiang, Robert M Nishikawa, Robert A Schmidt, Charles E Metz, Maryellen L Giger, and Kunio Doi. Improving breast cancer diagnosis with computer-aided diagnosis. *Academic radiology*, 6(1):22–33, 1999.

[64] SG Johnson. The nlopt nonlinear-optimization package [software], 2014.

[65] Aswin Kannan and Stefan M Wild. Obtaining quadratic models of noisy functions. *Preprint ANL/MCS-P1975-1111, Argonne National Laboratory*, 2012.

[66] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. *Neural networks*, 1, 2016.

[67] AI Kibzun and YS Kan. Stochastic programming problems with probability and quantile functions. *Journal of the Operational Research Society*, 48(8):849–849, 1997.

[68] Mitri Kitti. History of optimization, 2014.

[69] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[71] Pavlo Krokhmal, Michael Zabarankin, and Stan Uryasev. Modeling and optimization of risk. In *HANDBOOK OF THE FUNDAMENTALS OF FINANCIAL DECISION MAKING: Part II*, pages 555–600. World Scientific, 2013.

[72] Sang Gyu Kwak and Jong Hae Kim. Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology*, 70(2):144–156, 2017.

[73] Miguel Lázaro-Gredilla, Joaquin Quinonero-Candela, and Anıbal Figueiras-Vidal. Sparse spectral sampling gaussian processes. Technical report, Technical report, Microsoft Research, 2007.

[74] Sébastien Le Digabel. Algorithm 909: Nomad: Nonlinear optimization with the mads algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 37(4):44, 2011.

[75] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[76] Giampaolo Liuzzi, Stefano Lucidi, and Francesco Rinaldi. Derivative-free methods for bound constrained mixed-integer optimization. *Computational Optimization and Applications*, 53(2):505–526, 2012.

[77] Stefano Lucidi, Veronica Piccialli, and Marco Sciandrone. An algorithm model for mixed variable programming. *SIAM Journal on Optimization*, 15(4):1057–1084, 2005.

[78] Pamela McCorduck. *Machines who think: A personal inquiry into the history and prospects of artificial intelligence*. AK Peters/CRC Press, 2009.

[79] Mansfield Merriman. On the history of the method of least squares. *The Analyst*, 4(2):33–36, 1877.

[80] Jonas Mockus. *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media, 2012.

[81] Jorge J Moré and Stefan M Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[82] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[83] George L Nemhauser and Laurence A Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

[84] Eric Newby. *General solution methods for mixed integer quadratic programming and derivative free mixed integer non-linear programming problems*. PhD thesis, 2013.

[85] Eric Newby and M Montaz Ali. A trust-region-based derivative free algorithm for mixed integer programming. *Computational Optimization and Applications*, 60(1):199–229, 2015.

[86] Jorge Nocedal and Stephen J Wright. Numerical optimization 2nd, 2006.

[87] Michael JD Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*, pages 51–67. Springer, 1994.

[88] Michael JD Powell. Least frobenius norm updating of quadratic models that satisfy interpolation conditions. *Mathematical Programming*, 100(1):183–215, 2004.

[89] MJD Powell. Direct search algorithms for optimization calculations. *Acta numerica*, 7:287–336, 1998.

[90] MJD Powell and Y Yuan. A trust region algorithm for equality constrained optimization. *Mathematical Programming*, 49(1):189–211, 1990.

[91] Andras Prekopa. On probabilistic constrained programming. In *Proceedings of the Princeton symposium on mathematical programming*, volume 113, page 138. Princeton, NJ, 1970.

[92] Joaquin Quiñonero-Candela, Carl Edward Rasmussen, Aníbal R Figueiras-Vidal, et al. Sparse spectrum gaussian process regression. *Journal of Machine Learning Research*, 11(Jun):1865–1881, 2010.

[93] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian process for machine learning*. MIT press, 2006.

[94] Rommel G Regis and Christine A Shoemaker. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, 45(5):529–555, 2013.

[95] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[96] Christian Robert. Machine learning, a probabilistic perspective, 2014.

[97] Eric Roberts. Neural networks-history: The 1940's to the 1970's, 2000.

[98] R Rockafellar, Stanislav Uryasev, and Michael Zabarankin. Deviation measures in risk analysis and optimization. 2002.

[99] R Tyrrell Rockafellar and Stanislav Uryasev. Conditional value-at-risk for general loss distributions. *Journal of banking & finance*, 26(7):1443–1471, 2002.

[100] R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[101] Friedrich Sauvigny. *Partial Differential Equations 2: Functional Analytic Methods*. Springer Science & Business Media, 2012.

[102] Cristiana J Silva and Delfim FM Torres. Two-dimensional newton's problem of minimal resistance. *arXiv preprint math/0607197*, 2006.

[103] James C Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on aerospace and electronic systems*, 34(3):817–823, 1998.

[104] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[105] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[106] Robert A Stubbs and Sanjay Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical programming*, 86(3):515–532, 1999.

[107] Philippe L Toint. Non-monotone trust-region algorithms for nonlinear optimization subject to convex constraints. *Mathematical programming*, 77(3):69–94, 1997.

[108] Stanislav Uryasev. *Probabilistic constrained optimization: methodology and applications*, volume 49. Springer Science & Business Media, 2013.

[109] David Henry Winfield. *Function and functional optimization by interpolation in data tables*. PhD thesis, Harvard University, 1970.

[110] Xin-She Yang. *Engineering optimization: an introduction with metaheuristic applications*. John Wiley & Sons, 2010.