Strategies for Black-Box and Multi-Objective Optimization

Biswajit Paria

July 2022
CMU-ML-22-102

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Barnabás Póczos (co-chair)	Carnegie Mellon University
Jeff Schneider (co-chair)	Carnegie Mellon University
Zachary Lipton	Carnegie Mellon University
Abhimanyu Das	Google Research

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was funded by: Air Force Research Laboratory award FA87501720130; United States Army awards W911NF20F0015 and W911NF20F0017; grants from Lockheed Martin Corporation and the Toyota Research Institute; and research internships at Google and Snap Research. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: bayesian, black box, multi-objective, optimization, time series, neural networks, uncertainty, sparsity, retrieval, nearest neighbors, embeddings

To my family,
for their unconditional support for the last 28 years.

Abstract

Black-box optimization (BBO) problems occur frequently in many engineering and scientific disciplines, where one has access to zeroth-order evaluations of a function (black-box), that has to be optimized over a specified domain. In many situations, the function is expensive to evaluate, and hence the number of evaluations is limited by a budget. A popular class of algorithms known as Bayesian Optimization model the black-box function via surrogates, and proceed by evaluating points that are most likely to lead to the optimum. Multi-objective optimization (MOO) is another topic in optimization where the goal is to simultaneously optimize for multiple objectives defined over a common domain. Typically, these objectives do not achieve their optima for the same inputs. In such scenarios, rather than searching for a single best solution, a set of Pareto optimal solutions is desired. In this thesis, we study several optimization strategies for BBO and MOO and their applications.

The first half of this thesis is about BBO for expensive functions. First, we propose a simple and flexible approach for multi-objective black-box optimization (MOBO) based on the idea of random scalarizations. We introduce a notion of multi-objective regret and show that our strategy achieves zero regret as the budget grows. Next, we study the effectiveness of neural networks for expensive BBO. We show that a simple greedy approach can achieve a performance close to that of Gaussian process Bayesian optimization. Using recently studied connections between Gaussian processes and training dynamics of very wide neural networks, we prove upper bounds on the regret of our proposed algorithm. Lastly, we propose a cost-aware Bayesian optimization framework that takes into account the cost of each evaluation. This approach is useful in settings where the evaluation cost varies across the input domain and low cost evaluations can provide a large amount of information about the maximum.

The second half of this thesis is about the application of MOO to two differentiable MOO problems. Our first application is learning sparse embeddings for fast retrieval using neural networks. The objectives to be optimized here are retrieval accuracy and retrieval speed. We introduce a novel sparsity regularizer and demonstrate an annealing strategy that yields a better Pareto frontier of the objectives compared to other methods. For our second application, we consider the problem of hierarchical time series forecasting, where multiple related time series are organized as a hierarchy. We propose an approach that accounts for the hierarchical structure, while being scalable to large hierarchies, and show that it leads to an improved accuracy across most hierarchical levels. We also treat this as a multi-objective problem and demonstrate the performance trade-offs across various hierarchical levels.

To summarize our contributions, in this thesis we proposed strategies for optimization of various types of black-box and multi-objective functions, with experimental evaluation on synthetic or benchmark datasets.

Acknowledgments

They say that *it takes a village to raise a PhD*, and having survived this far, I cannot agree more. I could not have come this far without the constant support of my village consisting of my advisors, teachers, colleagues, friends, and family. The credits to this thesis also go to them, and I apologize in advance if I have missed anyone below.

First and foremost, I thank my advisors, Barnabàs and Jeff, for their support and guidance. I am thankful for the freedom they provided so that I could work on problems that *I* felt excited about. They have always encouraged me to work on problems that are fun and interesting, and nudged me back on the right path whenever I felt lost. They have been very inspiring and supportive, especially during the beginning of the pandemic coinciding with the loss of a family member. I will be eternally grateful to them for helping me get back on track. I also thank Abhi and Zack for being a part of the thesis committee despite their busy schedules, and for all the interesting conversations we have had about this thesis.

I feel lucky and thankful to have been mentored by $^{\alpha}$Abhi, Arun, Ian En-Hsu, Rajat, Samy, and Willie. I thoroughly enjoyed and learned a lot while working closely with them through all the paper deadlines and the brainstorming sessions. I am thankful to my close collaborators $^{\alpha}$Ady, Chih-Kuan (Jason), Kirby, Pradeep, Sachin, and Viraj, for all the research we did together. Thanks to the ML department, and special thanks to Diane for providing such a comfortable and friendly work environment.

CMU and the ML department undoubtedly provide an excellent academic environment where students can thrive. What made the journey even more pleasing was the company of amazing friends. I am grateful to my closest friends $^{\alpha}$Eyan, Giulio, Kartik, Kin, Leqi, Mihir, Sachin, Sneha, and Stef, for the countless dinners, parties, bar trips, and most importantly, for their support. My heartfelt thanks to Sneha for being so supportive of this huge endeavor despite the distance. I thank my bouldering buddies $^{\alpha}$Akash, Giulio, Neil, and Sachin for all the stress-busting sessions at Ascend and Iron City, my officemates $^{\alpha}$Che-Ping, Christina, and Jennifer for the exciting first-year energy and the fun office conversations, and special thanks to my furry feline friend Prof. Minerva for being kind enough to let me pet her.

The list is far from ending. I thank $^{\alpha}$Adarsh, Agnivo, Anurag, Arun, Charvi, Chenghui, Chih-Kuan (Jason), Chirag G, Chirag N, Chun-Liang, Cristian, Daphne, David, Devendra, Greg, Ian, Ivan, Jin, Mandy, Nicholay, Otilia, Paul, Pratyush, Pratik, Prithwish, Ritesh, Robin, Sai, Sebastian, Shaojie, Siddharth,

α: alphabetical ordering

Sunny, Tathagata, Viraj, Yang, and Young, for the countless conversations, hallway chats, research discussions, party invites, and trips together.

Getting the proper guidance at the right time during my formative years was important for me to reach this stage. I am thankful to my teachers, Maity Sir, Goutam Sir, and Karan Sir, for guiding me towards a strong foundation during my school years, and my professors at IIT Kharagpur and CMU for building a strong understanding of the fundamentals.

Next, it is the city of Pittsburgh's turn to be grateful for. My closest friends know how big of a proponent I can be, when it comes to Pittsburgh, despite so many people complaining about the weather. To me, Pittsburgh has been the optimal combination of quaint, nature, and city, with its amazing lush green summers, white snowy winters, beautiful uneven terrain making way for the hundreds of bridges - a city that offers everything, without the distractions of a large city (great for research, in my opinion). My experience at CMU could have been very different had Pittsburgh not been so close to nature. I am grateful for the access to streams and hills within walking distance, for the numerous trails at Schenley and Frick, the two rivers, great public transport, and the beautiful downtown Pittsburgh skyline.

Saving the best for the last, I am eternally indebted to my parents and my brother for always looking out for the family, for all the hard work we all put together, for the moments we shared, for being by my side throughout this journey, and the almighty for keeping us all safe. I dedicate this degree and thesis to them.

Contents

List of Figures

List of Tables

1 | Introduction

In a large variety of practical problems, we often have to deal with optimization of expensive black-box functions. The term black-box refers to the fact that such functions only offer *zeroth order* access to the function, that is, given a query point $\boldsymbol{x}$ belonging to some domain $\mathcal{X}$, the function can only yield the value of the function $f(\boldsymbol{x})$. This is in contrast to the *first order* access, where in addition to being able to compute $f(\boldsymbol{x})$, we can also compute the gradient $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. Global optimization of black-box functions is an important problem occurring in many scientific and engineering domains. Its recent popularity can be attributed to machine learning applications such as hyperparameter tuning for machine learning algorithms [119, 246], neural architecture search [118], and in general the broad topic of AutoML [99]. However, black-box optimization problems (BBO) can be frequently found in other practical applications such as experiment design [38, 171], online advertising, scientific discovery [74, 91, 116, 226, 263] and others [165, 198] to name a few. The evaluation of such functions can be expensive, for instance, the result of a machine learning experiment for a set of given input hyper-parameters, or the result of a simulation experiment for molecular discovery. As a result, optimization for such functions is often limited by a budget.

Global optimization of black-box functions have been studied over many decades. Existing approaches range from methods based on random walks [185, 208], evolutionary algorithms [14, 52, 304], estimating derivatives using finite differences (often studied under the realm of *derivative free optimization*) [143], to ones based on fitting a surrogate model or making smoothness assumptions on the black-box function [29, 130, 157, 176]. Methods based on random walks and evolutionary algorithms often require a large budget to find the optimum reliably, on the other hand smoothness assumptions are often necessary for black-box under a limited budget. Bayesian optimization [63, 96, 112, 251] is a popular class of methods used for expensive black-box optimization that assume a Gaussian process prior [218], thus essentially imposing a smoothness criterion on the function. Other kinds of surrogate models such as regression trees [98], linear models [83], and more recently neural networks [91, 125, 149, 247] have also been explored.

Several variations of the standard BBO have also gained recent interest including:

1. *Multi-Fidelity optimization [115, 287]:* where multiple levels of fidelity are available for each evaluation. For instance, a simulation experiment can be run at a lower accuracy setting requiring lesser computation resources, but producing a low fidelity approximation of the black box function at the queried point.

2. *Cost-Aware optimization [9, 144, 201]:* where each evaluation can have a variable cost. Examples include chemical experiments requiring a different quantity of materials for each experiment.

3. *Multi-Objective optimization [89, 200]:* where the goal is to optimize several objectives defined on a shared domain. For example, in neural architecture search, the user may want to maximize accuracy of a neural network classifier while minimizing computations. The objectives are often competing in nature and hence improving one of the objectives leads to worsening of the others. In such cases, a notion of optimality known as *Pareto* optimality is often desired.

Multi-objective optimization (MOO) is not limited to the black-box / zeroth-order setting. MOO problems can also be differentiable in nature, for instance, when training a classifier on an imbalanced dataset, the class-wise losses (typically differentiable) are considered as the multiple objectives that must be minimized.

In the first part of this thesis, we look at several problems in expensive black box optimization. In particular, multi-objective Bayesian optimization, cost-aware Bayesian optimization, and black-box optimization using neural networks. The second part of this thesis is about two problems with aspects of differentiable multi-objective optimization.

Contributions. We briefly discuss our main contributions below. In the first part of the thesis, we present our contributions in the domain of black-box optimization.

1. As discussed above, multi-objective optimization (MOO) problems occur in a variety of practical applications. Often, these objectives are expensive to evaluate, and are conflicting in nature – thus cannot be optimized simultaneously. Rather than optimizing for a single optimal point, most multi-objective optimization approaches aim to recover the Pareto front, defined as the set of Pareto optimal points (see Section 1.1.2 for a definition). A lot of multi-objective problems are also black-boxes, thus amenable to Bayesian optimization. Several prior approaches for multi-objective Bayesian optimization (MOBO) have been proposed, which are either conceptually complex, computationally expensive, or aim to estimate the whole Pareto front [89, 131, 207, 305].

   In Chapter 2, we propose a method for multi-objective Bayesian optimization (MOBO) based on random-scalarizations [200] in which a practitioner can encode their preferences as a prior and steer the exploration strategy towards interesting regions of the Pareto front. We propose a novel MOBO algorithm based on upper confidence bounds (UCB) and Thompson sampling (TS) [260]. We demonstrate the effectiveness of our approach on various synthetic and real world problems. We also introduce a novel measure of regret in the multi-objective setting, and theoretically show that our approach achieves a zero regret as the budget T increases to infinity. To the best of our knowledge, this is the first work to introduce a notion of regret and theoretical bounds in the multi-objective setting.

2. Gaussian processes are the most popular surrogates used in practice [73]. Neural network based approaches are gaining importance as replacements for GPs due to their ability to fit high dimensional and structured data [125, 289, 299]. Most existing approaches are either not theoretically principled or do not perform well in practice.

In Chapter 3 we show that a simple approach based on empirical risk minimization with neural networks performs at par if not better than GPs on benchmark problems. We also draw connections to the theory of wide neural networks [86, 147] and prove upper bounds on the regret.

3. In the standard black-box optimization setup, it is assumed that the cost of each query is uniform across the domain. In many situations the cost can vary based on the query point, such as scientific experiments that require varying amounts of resources in each experiments. To this end, in Chapter 4 we propose a cost-aware approach to expensive black-box optimization [201] based on the idea of information directed sampling [229]. We also derive theoretical upper bounds on the regret, and discuss the theoretical limitations of such an approach.

In the second part of the thesis, we present two problems with aspects of multi-objective optimization.

1. In Chapter 5, we propose a method for learning sparse representations for efficient nearest neighbors search [202]. In this work, we leverage sparse representations to compute fast dot products via sparse-matrix sparse-vector products. The goal here, is to learn sparse representations while optimizing for two competing objectives - to achieve a high recall, and a small retrieval time. We introduce a novel sparsity regularizer that optimally balances the sparsity and retrieval accuracy. One of our key observations is that, minimizing a standard regularized loss is not sufficient to yield the best sparsity for a given accuracy. We propose a regularization annealing schedule that helps both the objectives achieve better optima. We show the effectiveness of our approach on multiple retrieval tasks. A comparison of the Pareto frontiers of the two values shows an improved speed-vs-accuracy tradeoff for our proposed approach.

2. In Chapter 6, we consider the problem of hierarchical time series forecasting. To motivate the problem, consider the example of demand forecasting for a large ecommerce retail chain. Given a hierarchy (represented as a rooted tree) of all available products, forecasts can be made at various levels of hierarchy. Future demand forecasts can be made for a particular product occurring at the leaf of the hierarchy (fine-grained), or the total demand of a class of products occurring higher up in the hierarchy (aggregated/coarse-grained). The goal in this problem is to make accurate forecasts for each node of the hierarchy. We measure the accuracy of our predictions for each individual time series and aggregate the metrics at each hierarchical level. We propose a scalable approach that incorporates the hierarchy into the training loss, and show an improved overall performance across all hierarchical levels. We also treat it as a multi-objective problem, where the aggregated metric at each hierarchical level is an objective.

Broadly speaking, in this thesis we propose several strategies for black-box and multi-objective optimization, that are applicable to many problems in practice.

1.1 Background

In this section we provide a brief introduction to the background for each of the topics.

1.1.1 Gaussian Processes and Bayesian Optimization

Gaussian Processes. A Gaussian process (GP) defines a prior distribution over functions defined on some input space $\mathcal{X}$. GPs are characterized by a mean function $\mu : \mathcal{X} \mapsto \mathbb{R}$ and a kernel $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$. For any function $f \sim \mathcal{GP}(\mu, \kappa)$ and some finite set of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$, the function values $f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)$ follow a multivariate Gaussian distribution with mean μ and covariance Σ given by $\mu_i = \mu(\boldsymbol{x}_i)$, $\Sigma_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad \forall 1 \leq i, j \leq n$. Examples of popular kernels include the squared exponential and the Matérn kernel with smoothness parameter $\nu = 5/2$.

$$k_{\text{se}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\frac{\|x - x'\|_2^2}{2\ell^2}\right) \tag{1.1}$$

$$k_{\text{mat},5/2}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \left(1 + \frac{\sqrt{5}d}{\rho} + \frac{5d^2}{3\rho^2}\right) \exp\left(-\frac{\sqrt{5}d}{\rho}\right) \tag{1.2}$$

Duvenaud [55] summarizes a variety of kernel functions with different properties in the Kernel Cookbook[1]. The mean function μ is typically assumed to be $\mathbf{0}$ in the absence of any domain knowledge. The posterior process, given observations $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{t-1}$ where $y_i = f(\boldsymbol{x}_i) + \epsilon_i \in \mathbb{R}$, $\epsilon_i \sim \mathcal{N}(\mu, \sigma^2)$, is also a GP with the mean and kernel function given by

$$\mu_t(\boldsymbol{x}) = k^T(\Sigma + \sigma^2 I)^{-1} Y, \quad \kappa_t(\boldsymbol{x}, \boldsymbol{x}') = \kappa(\boldsymbol{x}, \boldsymbol{x}') - k^T(\Sigma + \sigma^2 I)^{-1} k'. \tag{1.3}$$

where $Y = [y_i]_{i=1}^t$ is the vector of observed values, $\Sigma = [\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^t$ is the Gram matrix, $k = [\kappa(\boldsymbol{x}, \boldsymbol{x}_i)]_{i=1}^t$, and $k' = [\kappa(\boldsymbol{x}', \boldsymbol{x}_i)]_{i=1}^t$. Further details on GPs can be found in [218].

Bayesian Optimization. We assume that the given black-box function is $f : \mathcal{X} \mapsto \mathbb{R}$, and querying any point $\boldsymbol{x} \in \mathcal{X}$ returns a noisy value of the function $y = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is often assumed to be a zero mean Gaussian random variable.

BO methods operate sequentially, using past observations $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{t-1}$ to determine the next point $\boldsymbol{x}_t$. Given $t-1$ observations, the posterior GP can be computed using Eq. (1.3). An acquisition function is optimized to compute $\boldsymbol{x}_t$. Roughly speaking, an acquisition function $\text{acq}(\boldsymbol{x}_t)$ quantifies the additional benefit of observing $\boldsymbol{x}_t$ towards estimating the optimum of the black-box function. Several aquisition functions have been proposed in the literature. Thompson Sampling (TS) [230, 260] draws a sample g_t from the posterior GP. The next candidate $\boldsymbol{x}_t$ is choosen as $\boldsymbol{x}_t = \text{argmax}_{\boldsymbol{x} \in \mathcal{X}}\, g_t(\boldsymbol{x})$. GP-UCB [252] constructs an upper confidence bound as $U_t(\boldsymbol{x}) = \mu_{t-1}(\boldsymbol{x}) + \sqrt{\beta_t}\sigma_{t-1}(\boldsymbol{x})$, where μ_{t-1} and σ_{t-1} are the posterior mean and covariances according to equations 1.3. β_t is a function of t and the dimensionality of the input domain $\mathcal{X}$. The next point is selected as $\boldsymbol{x}_t = \text{argmax}_{\boldsymbol{x} \in \mathcal{X}}\, U_t(\boldsymbol{x})$. Expected improvement is a popularly used acquisition function as it often performs better than others in practice. The expected improvement at a point is defined as $\text{EI}(\boldsymbol{x}) = \mathbb{E}_{f|\mathcal{D}}[\max(f(\boldsymbol{x}) - y_{\text{best}}, 0)]$ where the expectation is over f conditioned on the observations $\mathcal{D}$. The next query $\boldsymbol{x}_t$ is chosen as $\boldsymbol{x}_t = \text{argmax}_{\boldsymbol{x} \in \mathcal{X}}\, \text{EI}(\boldsymbol{x})$. A typical BO loop is shown in Algorithm 1.1 and Fig. 1.1. Fig. 1.2 illustrates a single iteration for a randomly sampled black-box function.

[1] `https://www.cs.toronto.edu/~duvenaud/cookbook/`

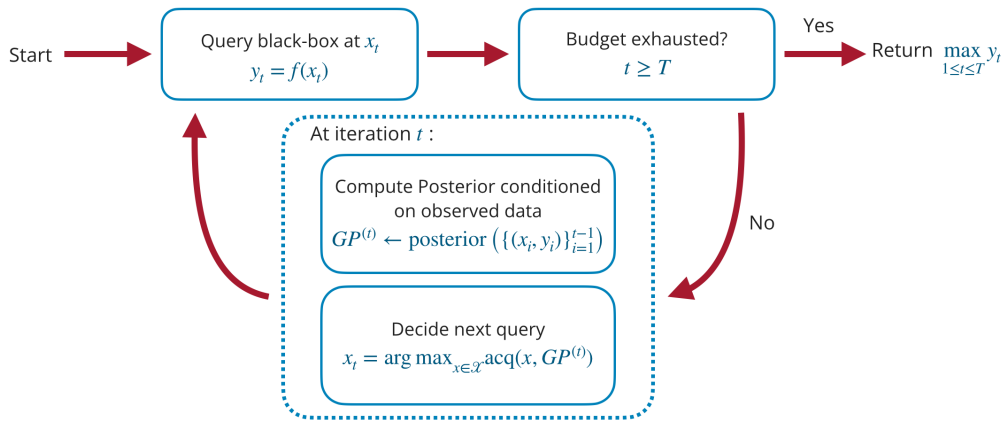


Figure 1.1: A typical BO loop. The dotted box denotes the decision making step which includes computing the posterior and acquisition maximization.

Algorithm 1.1 Bayesian optimization loop

Input: Black-box function f, kernel κ, exploration budget T_e, total budget T.

for $t = 1 \to T_e$ **do**
  Sample $\boldsymbol{x}_t \sim \mathcal{X}$
  Observe $y_t = f(\boldsymbol{x}_t)$
for $t = T_e + 1 \to T$ **do**
  $\mathcal{D}_t \leftarrow \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{t-1}$
  $\mathcal{GP}^{(t)} \leftarrow \text{posterior}\left(\mathcal{GP}(\mathbf{0}, \kappa) \mid \mathcal{D}_t\right)$
  $\boldsymbol{x}_t \leftarrow \text{argmax}_{\boldsymbol{x} \in \mathcal{X}}\ \text{acq}(\boldsymbol{x},\ \mathcal{GP}^{(t)})$
  Evaluate $\boldsymbol{y}_t = f(\boldsymbol{x}_t)$
$b = \text{argmax}_{t=1}^{T}\, y_t$
return $\boldsymbol{x}_b, y_b$

Regret. The performance of a BBO algorithm at iteration T is measured using one of the following criteria:

$$\text{Simple Regret:} \quad \min_{t=1}^{T} f(\boldsymbol{x}_t) - \min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}),$$

$$\text{Cumulative Regret:} \quad \sum_{t=1}^{T} f(\boldsymbol{x}_t) - \min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}),$$

where $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T$ are the sequence of points queried by the algorithm. Intuitively, the regret measures how sub-optimal are the queried points compared to the true optimum value.

1.1.2 Multi-Objective Optimization

Pareto optimality. Multiple objectives are often conflicting in nature and hence, cannot be optimized simultaneously. Rather than optimizing for a single optimal point, most

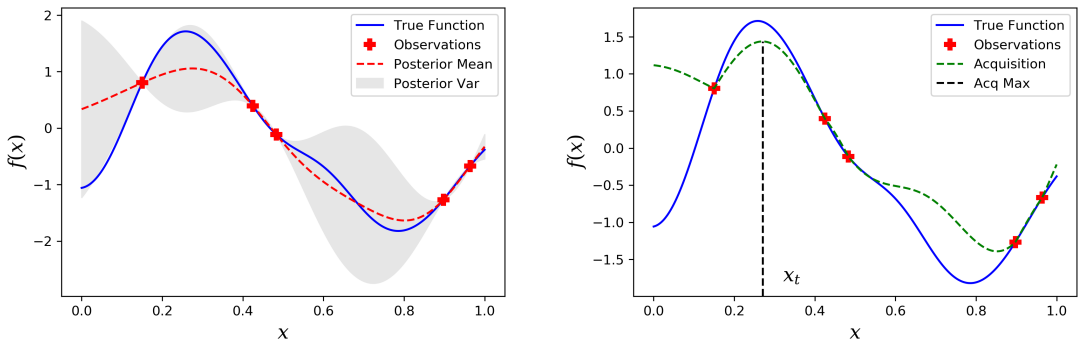


Figure 1.2: A visual illustration of a typical BO iteration. We consider a randomly sampled one-dimensional function (blue) as our black-box. Given observations (red crosses), the dashed red curve denotes the posterior mean and the grey region denotes the posterior variance. The dashed green curve (right) denotes the UCB acquisition function which is maximized to yield the next point $\boldsymbol{x}_t$ denoted by the dashed black line.

multi-objective optimization approaches aim to recover the Pareto front, defined as the set of Pareto optimal points. This is depicted in Fig. 1.3a for a two objective function f. A formal definition of Pareto optimality is given below.

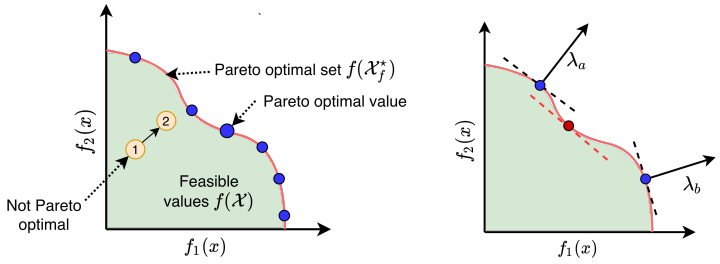
Pareto optimality. A point is Pareto optimal if it cannot be improved in any of the objectives without degrading some other objective. More formally, given K objectives $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})) : \mathcal{X} \to \mathbb{R}^K$ over a compact domain $\mathcal{X} \subset \mathbb{R}^d$, a point $\boldsymbol{x}_1 \in \mathcal{X}$ is Pareto dominated by another point $\boldsymbol{x}_2 \in \mathcal{X}$ iff $f_k(\boldsymbol{x}_1) \leq f_k(\boldsymbol{x}_2)\ \forall k \in [K]$[2] (pointwise dominance) and $\exists k \in [K]$ s.t. $f_k(\boldsymbol{x}_1) < f_k(\boldsymbol{x}_2)$ (unequal points). We denote this by $f(\boldsymbol{x}_1) \prec f(\boldsymbol{x}_2)$. A point is Pareto optimal if it is not Pareto dominated by any other point. The Pareto frontier is defined as the set of all Pareto optimal values. We use $\mathcal{X}_f^\star$ to denote the Pareto frontier for a multi-objective function f, and $f(\mathcal{X}_f^\star)$ to denote the set of Pareto optimal values, where function of a set is defined as the element-wise functions of the elements of the set.[3]

Scalarizations. One of the standard approaches to multi-objective optimization is to convert the vector valued function outputs to a scalar and optimize it. A linear scalarization, for instance, is essentially a weighted sum of the elements of the vector. Given a vector output $\boldsymbol{y} = f(\boldsymbol{x})$, and a weight vector $\boldsymbol{\lambda} \in \mathbb{R}^K$, a linear scalarization is given by,

$$\mathfrak{s}_{\boldsymbol{\lambda}}^{\text{lin}}(\boldsymbol{y}) = \sum_{k=1}^{K} \boldsymbol{\lambda}_k \boldsymbol{y}_k. \tag{1.4}$$

As depicted in Fig. 1.3b, maximizing $\mathfrak{s}_{\boldsymbol{\lambda}}^{\text{lin}}(f(\boldsymbol{x}))$ yields a point on the Pareto front where the tangent is orthogonal to the $\boldsymbol{\lambda}$, provided that the tangent does not pass through the interior of the feasible region. Linear scalarizations can be used to yield Pareto optimal

[2] We use $[K]$ to denote the set $\{1, \ldots, K\}$.
[3] Define $f(\boldsymbol{X}) = \{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \boldsymbol{X}\}$ for any set $\boldsymbol{X}$.



(a) (b)

Figure 1.3: **(a)** f_1, f_2 denote the two objectives. The green region denotes the feasible set of possible function values. The red boundary denotes the set of Pareto optimal values and the blue points denote samples of Pareto optimal values. Point 1 is example of a non-Pareto optimal value, as it is dominated by point 2. **(b)** The blue points represent Pareto optimal points that can be obtained using linear scalarizations $\boldsymbol{\lambda}_a, \boldsymbol{\lambda}_b$. No linear scalarization can yield the red point in the non-'convex' region, as the projection of the blue points on any $\boldsymbol{\lambda}$ will result in a higher value than the red point.

points in the 'convex'[4] regions of the Pareto front only. A different scalarization known as the Tchebyshev scalarization, allows us to obtain points from all the regions of the Pareto front including the non-'convex' regions as shown in Fig. 1.3b.

$$\mathfrak{s}_{\boldsymbol{\lambda}}^{\text{tch}}(\boldsymbol{y}) = \min_{k=1}^{K} \boldsymbol{\lambda}_k(\boldsymbol{y}_k - \boldsymbol{z}_k), \quad \text{where } \boldsymbol{z} \text{ is a given reference point.} \tag{1.5}$$

Constrained optimization. Multi-objective optimization is also closely related to constrained optimization. Consider *convex* functions $f, g_1, \cdots, g_k : \mathcal{X} \to \mathbb{R}$ for some convex set $\mathcal{X}$, and the following optimization problem.

$$\text{Let } x^* \in \operatorname*{argmax}_{x \in \mathcal{X}} f(x) \quad \text{subject to} \quad g_1(x) \geq c_1, \cdots, g_k(x) \geq c_k. \tag{1.6}$$

Then provided that the KKT conditions [25, 221] hold true, there exist

$$\lambda_1 \geq 0, \cdots, \lambda_k \geq 0, \quad \text{such that} \quad x^* \in \operatorname*{argmax}_{x \in \mathcal{X}} f(x) + \lambda_1 g_1(x) + \cdots + \lambda_k g_k(x). \tag{1.7}$$

The converse can also be shown to hold true, that is, for any $\lambda_1, \cdots, \lambda_k \geq 0$, there exist constants $c_1, \cdots, c_k$ such that the optimum in Eq. (1.7) is also the optimum in Eq. (1.6).

[4]Refers to the convexity of the Pareto front, not the objectives themselves.

Part I

Black Box Optimization

2 | Multi-Objective Bayesian Optimization

2.1 Introduction

Bayesian optimization (BO) is a popular recipe for optimizing expensive black-box functions where the goal is to find a global maximizer of the function. In many practical applications however, we are required to optimize multiple objectives, and moreover, these objectives tend to be competing in nature. For instance, consider drug discovery, where each evaluation of the functions is an in-vitro experiment and as the output of the experiment, we measure the solubility, toxicity and potency of a candidate example. A chemist wishes to find a molecule that has high solubility and potency, but low toxicity. This is an archetypal example for Bayesian optimization as the lab experiment is expensive. Further, drugs that are very potent are also likely to be toxic, so these two objectives are typically competing. Other problems include creating fast but accurate neural networks. While smaller neural networks are faster to evaluate, they suffer in terms of accuracy.

Due to their conflicting nature, all the objectives cannot be optimized simultaneously. As a result, most multi-objective optimization (MOO) approaches aim to recover the Pareto front, defined as the set of Pareto optimal points, as introduced in Section 1.1.2. The traditional goal in the MOO optimization regime is to approximate the set of Pareto optimal points [89, 131, 207, 305].

However, in certain scenarios, it is preferable to explore only a part of the Pareto front. For example, consider the drug discovery application described above. A method which aims to find the Pareto front, might also invest its budget to discover drugs that are potent, but too toxic to administer to a human. Such scenarios arise commonly in many practical applications. Therefore, we need flexible methods for MOO that can steer the sampling strategy towards regions of the Pareto front that a domain expert may be interested in. Towards this end, we propose a Bayesian approach based on random-scalarizations in which the practitioner encodes their preferences as a prior on a set of scalarization functions.

A common approach to multi-objective optimization is to use *scalarization functions*[1] $\mathfrak{s}_{\boldsymbol{\lambda}}(\boldsymbol{y}) : \mathbb{R}^K \to \mathbb{R}$ [222], parameterized by $\boldsymbol{\lambda}$ belonging to a set Λ, and $\boldsymbol{y} \in \mathbb{R}^K$ denoting K-dimensional objective values. Scalarizations are often used to convert multi-objective values

[1] Also known as utility functions in decision theory literature. To avoid confusion with acquisition functions which are sometimes referred to as utility functions in BO, we use the term scalarization function.

to scalars, and standard Bayesian optimization methods for scalar functions are applied. Since our goal is to sample points from the Pareto front, we need additional assumptions to ensure that the utility functions are maximized for $\boldsymbol{y} \in f(\mathcal{X}_f^\star)$. Following Roijers et al. [222] and Zintgraf et al. [302] we assume that $\mathfrak{s}_{\boldsymbol{\lambda}}(\boldsymbol{y})$ are monotonically increasing in all coordinates. Optimizing for a single fixed scalarization amounts to the following maximization problem, which returns a single optimal point lying on the Pareto front.

$$\boldsymbol{x}_{\boldsymbol{\lambda}}^\star = \underset{\boldsymbol{x} \in \mathcal{X}}{\operatorname{argmax}} \quad \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x})) \tag{2.1}$$

One can verify that $\boldsymbol{x}_{\boldsymbol{\lambda}}^\star \in \mathcal{P}_f$ follows from the monotonicity of the scalarization. In this problem, we are interested in a set of points $\boldsymbol{X}^\star = \{\boldsymbol{x}_i^\star\}_{i=1}^T$ of size at most T, spanning a specified region of the Pareto front rather than a single point. To achieve this we take a Bayesian approach and assume a prior $p(\boldsymbol{\lambda})$ with support on Λ, which intuitively translates to a prior on the set of scalarizations $\mathcal{S}_\Lambda = \{\mathfrak{s}_{\boldsymbol{\lambda}} \mid \boldsymbol{\lambda} \in \Lambda\}$. Thus, in place of optimizing a single scalarization, we aim to optimize over a set of scalarizations weighted by the prior $p(\boldsymbol{\lambda})$. Each $\boldsymbol{\lambda} \in \Lambda$ maps to a pareto optimal value $f(\boldsymbol{x}_{\boldsymbol{\lambda}}^\star) \in f(\mathcal{X}_f^\star)$. Thus, the prior $p(\boldsymbol{\lambda})$ defines a probability distribution over the set of Pareto optimal values, and hence encodes user preference, which is depicted in Fig. 2.1.

We propose to minimize a Bayes regret which incorporates user preference through the prior and scalarization specified by the user. We propose multi-objective extensions of classical BO algorithms: upper confidence bound (UCB) [10], and Thompson sampling (TS) [260] to minimize our proposed regret. At each step the algorithm computes the next point to evaluate by randomly sampling a scalarization $\mathfrak{s}_{\boldsymbol{\lambda}}$ using the prior $p(\boldsymbol{\lambda})$, and optimizes it to get $\boldsymbol{x}_{\boldsymbol{\lambda}}^\star$. Our algorithm is fully amenable to changing priors in an interactive setting, and hence can also be used with other interactive strategies in the literature. The complete algorithm is presented in Algorithm 2.1 and discussed in detail in Section 2.2. While random scalarizations have been previously explored by Knowles [131] and Zhang et al. [294], our approach is different in terms of the underlying algorithm. Furthermore, we study a more general class of scalarizations and also prove regret bounds. As we shall see, this formulation fortunately gives rise to an extremely flexible framework that is much simpler than the existing work for MOO and computationally less expensive. Our contributions can summarized as follows:

1. We propose a flexible framework for MOO using the notion of random scalarizations. Our algorithm is flexible enough to sample from the entire Pareto front or an arbitrary region specified by the user. It is also naturally capable of sampling from non-convex regions of the Pareto front. While other competing approaches can be modified to sample from such complex regions, this seamlessly fits into our framework. In contrast to the prior work on MOBO, we consider more general scalarizations that are only required to be Lipschitz and monotonic.
2. We prove sublinear regret bounds making only assumptions of Lipschitzness and monotonicity of the scalarization function. To our knowledge the only prior work discussing theoretical guarantees for MOO algorithms is Pareto Active Learning [305] with sample complexity bounds.
3. We compare our algorithm to other existing MOO approaches on synthetic and real-life tasks. We demonstrate that our algorithm achieves the said flexibility and superior performance in terms of the proposed regret, while being computationally inexpensive.

Related Work

Most multi-objective bayesian optimization approaches aim at approximating the whole Pareto front. Predictive Entropy Search (PESMO) by Hernández-Lobato et al. [89] is based on reducing the posterior entropy of the Pareto front. SMSego by Ponweiser et al. [207] uses an optimistic estimate of the function in an UCB fashion, and chooses the point with the maximum hypervolume improvement. Pareto Active Learning (PAL) [305] and ε-PAL [306] are similar to SMSego, and with theoretical guarantees. [32] introduce another active learning approach that approximates the surface of the Pareto front. Expected hypervolume improvement (EHI) [58] and Sequential uncertainty reduction (SUR) [205] are two similar approaches based on maximizing the expected hypervolume. Computing the expected hypervolume is an expensive process that renders EHI and SUR computationally intractable in practice when there are several objectives.

The idea of random scalarizations has been previously explored in the following works aimed at recovering the whole Pareto front: ParEGO [131] which uses random scalarizations to explore the whole Pareto front; MOEA/D [293], an evolutionary computing approach to MOO; and MOEA/D-EGO [294], an extension of MOEA/D using Gaussian processes that evaluates batches of points at a time instead of a single point. At each iteration, both ParEGO and MOEA/D-EGO sample a weight vector uniformly from the $K-1$ simplex, which is used to compute a scalar objective. The next candidate point is chosen by maximizing an *off-the-shelf* acquisition function over the GP fitted on the scalar objective. Our algorithm on the other hand, maintains K different GPs, one for each objective. Furthermore, our approach *necessitates* using acquisitions specially designed for the multi-objective setting for any general scalarization; more specifically, they are generalizations of single-objective acquisitions for multiple objectives (see Table 2.1). These differences with ParEGO are not merely superficial – our approach gives rise to a theoretical regret bound, while no such bound exists for the above methods.

Another line of work involving scalarizations include utility function based approaches. Roijers et al. [222], Zintgraf et al. [302] propose scalar utility functions as an evaluation criteria. Roijers et al. [223, 224], Zintgraf et al. [303] propose interactive strategies to maximize an unknown utility. In contrast to our approach the utility in these works is assumed to be fixed.

While there has been ample work on incorporating preferences in multi-objective optimization using evolutionary techniques [27, 28, 51, 124, 259], there has been fewer on using preferences for optimization, when using surrogate functions. Surrogate functions are essential for expensive black-box optimization. PESC [64] is an extension of PESM allowing to specify preferences as constraints. Hakanen and Knowles [80] propose an extension of ParEGO in an interactive setting, where users provide feedback on the observations by specifying constraints on the objectives in an online fashion. Yang et al. [291] propose another way to take preferences into account by using truncated functions. An interesting idea proposed by Sato et al. [237] uses a modified notion of Pareto dominance to prevent one or more objectives from being too small. The survey by Ishibuchi et al. [105] on evolutionary approaches to MOO can be referred for a more extensive review.

When compared to existing work for MOO, our approach enjoys the following advantages.

1. *Flexibility:* Our approach allows the flexibility to specify any region of the Pareto front including non-connected regions of the Pareto front, which is not an advantage enjoyed by other methods. Furthermore, the approach is flexible enough to recover the entire Pareto front when necessary. Our approach is not restricted to linear scalarization and extends to a much larger class of scalarizations.
2. *Theoretical guarantees:* Our approach seamlessly lends itself to analysis using our proposed notion of regret, and achieves sub-linear regret bounds.
3. *Computational simplicity:* The computational complexity of our approach scales linearly with the number of objectives K. This is in contrast to EHI and SUR, whose complexity scales exponentially with K. Our method is also computationally cheaper than other entropy based methods such as PESMO.

Background. A brief background on Gaussian processes and Bayesian optimization can be found in Section 1.1.1.

2.2 Our Approach

We first provide a formal description of random scalarizations, then we formulate a regret minimization problem, and finally propose multi-objective extensions of the classical UCB and TS algorithms to optimize it.

2.2.1 Random Scalarizations

As discussed earlier in Section 2.1, we consider a set of scalarizations $\mathfrak{s}_{\boldsymbol{\lambda}}$ parameterized by $\boldsymbol{\lambda} \in \Lambda$. We assume a prior $p(\boldsymbol{\lambda})$ with support Λ. We further assumed that, for all $\boldsymbol{\lambda} \in \Lambda$, $\mathfrak{s}_{\boldsymbol{\lambda}}$ is $L_{\boldsymbol{\lambda}}$-Lipschitz in the ℓ_1-norm and monotonically increasing in all the coordinates. More formally,

$$\begin{gathered} \mathfrak{s}_{\boldsymbol{\lambda}}(\boldsymbol{y}_1) - \mathfrak{s}_{\boldsymbol{\lambda}}(\boldsymbol{y}_2) \leq L_{\boldsymbol{\lambda}} \|\boldsymbol{y}_1 - \boldsymbol{y}_2\|_1, \quad \forall \boldsymbol{\lambda} \in \Lambda, \quad \boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{R}^d, \\ \text{and,} \quad \mathfrak{s}_{\boldsymbol{\lambda}}(\boldsymbol{y}_1) < \mathfrak{s}_{\boldsymbol{\lambda}}(\boldsymbol{y}_2) \text{ whenever } \boldsymbol{y}_1 \prec \boldsymbol{y}_2. \end{gathered} \tag{2.2}$$

The Lipschitz condition can also be generalized to ℓ_p-norms using the fact that $\|\boldsymbol{y}\|_1 \geq K^{1-\frac{1}{p}} \|\boldsymbol{y}\|_p$ for any $p \in [1, \infty]$ and $\boldsymbol{y} \in \mathbb{R}^K$. Monotonicity ensures that

$$\boldsymbol{x}^\star_{\boldsymbol{\lambda}} = \operatorname*{argmax}_{\boldsymbol{x} \in \mathcal{X}} \quad \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x})) \in \mathcal{X}^\star_f,$$

since otherwise, if $f(\boldsymbol{x}^\star_{\boldsymbol{\lambda}}) \prec f(\boldsymbol{x})$ for some $\boldsymbol{x} \neq \boldsymbol{x}^\star_{\boldsymbol{\lambda}}$, then we have $\mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}^\star_{\boldsymbol{\lambda}})) < \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))$, leading to a contradiction. Each $\boldsymbol{\lambda} \in \Lambda$ maps to an $\boldsymbol{x}^\star_{\boldsymbol{\lambda}} \in \mathcal{X}^\star_f$ and a $\boldsymbol{y}^\star = f(\boldsymbol{x}^\star_{\boldsymbol{\lambda}}) \in f(\mathcal{X}^\star_f)$. Assuming the required measure theoretic regularity conditions hold, the prior $p(\boldsymbol{\lambda})$ imposes a probability distribution on $f(\mathcal{X}^\star_f)$ through the above mapping as depicted in Fig. 2.1.

2.2.2 Bayes Regret

In contrast to Eq. (2.1), which returns a single optimal point, we aim to return a set of points from the user specified region. Our goal is to compute a subset $\boldsymbol{X} \subset \mathcal{X}$ such that $f(\boldsymbol{X})$ spans the high probability region of $f(\mathcal{X}^\star_f)$. This can be achieved by minimizing the

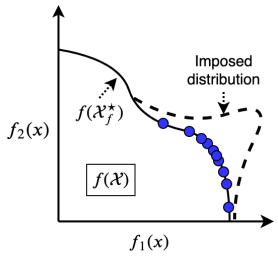


Figure 2.1: A prior $p(\boldsymbol{\lambda})$ imposes a distribution on the set of Pareto optimal values. The imposed probability density is illustrated using the dotted lines. The imposed distribution leads to a concentration of the sampled values (blue circles) in the high probability region.

following *Bayes regret* denoted by $\mathcal{R}_B$,

$$\begin{aligned} \mathcal{R}_B(\boldsymbol{X}) &= \mathbb{E}_{\boldsymbol{\lambda}\sim p(\boldsymbol{\lambda})}\Big(\underbrace{\max_{\boldsymbol{x}\in\mathcal{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x})) - \max_{\boldsymbol{x}\in\boldsymbol{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))}_{\text{Pointwise regret}}\Big), \\ \boldsymbol{X}^\star &= \operatorname*{argmin}_{\boldsymbol{X}\subset\mathcal{X},|\boldsymbol{X}|\leq T} \quad \mathcal{R}_B(\boldsymbol{X}) \end{aligned} \tag{2.3}$$

We now elaborate on the above expression. The pointwise regret $\max_{\boldsymbol{x}\in\mathcal{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x})) - \max_{\boldsymbol{x}\in\boldsymbol{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))$ quantifies the regret for a particular $\boldsymbol{\lambda}$ and is analogous to the simple regret in the standard bandit setting [30]. $\mathcal{R}_B(T)$ similarly corresponds to the Bayes simple regret in a bandit setting. The pointwise is minimized when $\boldsymbol{x}^\star_{\boldsymbol{\lambda}} = \operatorname{argmax}_{\boldsymbol{x}\in\mathcal{X}} \quad \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))$ belongs to $\boldsymbol{X}$. Since $\boldsymbol{X}$ is finite, the minimum may not be achieved for all $\boldsymbol{\lambda}$, as the set of optimial points can be potentially infinite. However, the regret can be small when $\exists \boldsymbol{x} \in \boldsymbol{X}$ such that $f(\boldsymbol{x}),\ f(\boldsymbol{x}^\star_{\boldsymbol{\lambda}})$ are close, from which it follows using the Lipschitz assumption that $\mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}^\star_{\boldsymbol{\lambda}})) - \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))$ is small. Therefore, roughly speaking, the Bayes regret is minimized when the points in $\boldsymbol{X}$ are Pareto optimal and $f(\boldsymbol{X})$ well approximates the high probability regions of $f(\mathcal{X}^\star_f)$. In this case, $\mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}^\star_{\boldsymbol{\lambda}})) - \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))$ is small for $\boldsymbol{\lambda}$s with high probabilities. Even though the rest of the regions are not well approximated, it does not affect the Bayes regret since those regions do not dominate the expectation by virtue of their low probability. This is what was desired from the beginning, that is, to compute a set of points with the majority of them spanning the desired region of interest. This is also illustrated in Fig. 2.2 showing three scenarios which can incur a high regret.

It is interesting to ask, why cannot one simply maximize

$$\max_{\boldsymbol{x}\in\mathcal{X}} \mathbb{E}_{\boldsymbol{\lambda}\sim p(\boldsymbol{\lambda})} \left[\mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))\right].$$

The above expression can be maximized using a single point $\boldsymbol{x}$ which is not the purpose of our approach. On the other hand, our proposed Bayes regret is not minimized by a single point or multiple points clustered in a small region of the Pareto front. Minimizing the pointwise regret for a single $\boldsymbol{\lambda}$ does not minimize the Bayes regret, as illustrated in Fig. 2.2. Our proposed regret has some resemblance to the expected utility metric in Zintgraf et al. [302]. However, the authors present it as an evaluation criteria, whereas we propose an optimization algorithm for minimizing it and also prove regret bounds on it.

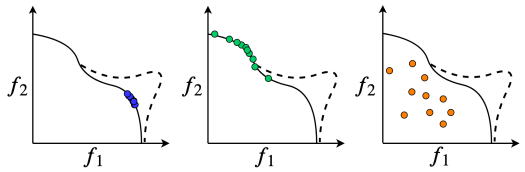


Figure 2.2: Three scenarios which incur a high regret: (1) The points are clustered in a small region. (2) The points are not from the desired distribution. (3) The points are not Pareto optimal.

2.2.3 Scalarized Thompson Sampling and UCB

In this section we introduce Thompson Sampling and UCB based algorithms for minimizing the Bayes regret. In contrast to other methods based on random scalarizations [131, 182], our algorithm does not convert each observation to a scalar value and fit a GP on them, but instead models them separately by maintaining a GP for each objective separately. In each iteration, we first fit a GP for each objective using the previous observations. Then we sample a $\boldsymbol{\lambda} \sim p(\boldsymbol{\lambda})$, which is used to compute a multi-objective acquisition function based on the scalarization $\mathfrak{s}_{\boldsymbol{\lambda}}$. The next candidate point is chosen to be the maximizer of the acquisition function. The complete algorithm is presented in Algorithm 2.1 and the acquisition functions are presented in Table 2.1. The acquisition function for UCB is a scalarization of the individual upper bounds of each of the objectives. Similarly, the acquisition function for TS is a scalarization of posterior samples of the K objectives.

The intuition behind our approach is to choose the $\boldsymbol{x}_t$ that minimizes the pointwise regret for the particular $\boldsymbol{\lambda}_t$ sampled in that iteration. Looking at the expression of the Bayes regret, at a high level, it seems that it can be minimized by sampling a $\boldsymbol{\lambda}$ from the prior and choosing an $\boldsymbol{x}_t$ that minimizes the regret for the sampled $\boldsymbol{\lambda}$. We prove regret bounds for both TS and UCB in Section 2.3 and show that this idea is indeed true.

Practical Considerations. In practice, our method requires the prior and class of scalarization functions to be specified by the user. These would typically be domain dependent. In practice, a user would also interactively update their prior based on the observations, as done in Hakanen and Knowles [80], Roijers et al. [223, 224]. Our approach is fully amenable to changing the prior interactively, and changing regions of interest. We do not propose any general methods for choosing or updating the prior, as it is not possible to do so for any general class of scalarizations. The interested readers can refer to the literature on interactive methods for MOBO. However, for the sake of demonstration we propose a simple heuristic in the experimental section.

2.2.4 Computational Complexity

At each step all algorithms incur a cost of at most $\mathcal{O}(KT^3)$, for fitting K GPs, except for ParEGO, which fits a single GP at each time step with a cost of $\mathcal{O}(T^3)$. The next step of maximizing the acquisition function differs widely across the algorithms. Computing the acquisition function at each point $\boldsymbol{x}$ costs $\mathcal{O}(T)$ for ParEGO, and $\mathcal{O}(KT)$ for our approach. The additional factor K is the price one must pay when maintaining K GPs.

Apart from fitting the K GPs, SMSEgo requires computing the expected hypervolume gain

Algorithm 2.1 MOBO using Random Scalarizations (MOBO-RS)

Init $\mathcal{D}^{(0)} \leftarrow \emptyset,\ \mathcal{GP}_k^{(0)} \leftarrow \mathcal{GP}(\mathbf{0}, \kappa),\ \forall k \in [K]$
for $t = 1 \rightarrow T$ **do**
  Sample $\boldsymbol{\lambda}_t \sim p(\boldsymbol{\lambda})$
  $\boldsymbol{x}_t \leftarrow \text{argmax}_{\boldsymbol{x} \in \mathcal{X}}\ \text{acq}(\boldsymbol{x}, \boldsymbol{\lambda}_t)$ ▷ See Table 2.1 for acquisition functions
  Evaluate $\boldsymbol{y} = f(\boldsymbol{x}_t)$
  $\mathcal{D}^{(t)} = \mathcal{D}^{(t-1)} \cup \{(\boldsymbol{x}_t, \boldsymbol{y})\}$
  $\mathcal{GP}_k^{(t)} \leftarrow \text{post}\left(\mathcal{GP}_k^{(t-1)} \mid (\boldsymbol{x}_t, \boldsymbol{y}_k)\right), \forall k \in [K]$
return $\mathcal{D}^{(T)}$

Table 2.1: Acquisition functions used in Algorithm 2.1. $\mu^{(t)}(\boldsymbol{x})$, $\sigma^{(t)}(\boldsymbol{x})$ are K dimensional vectors denoting the posterior means and variances at $\boldsymbol{x}$ of the K objectives respectively, in step t. c is a hyperparameter and d is dimension of the input space $\mathcal{X}$. f'_k is randomly sampled from the posterior of the kth objective function.

	$\text{acq}(\boldsymbol{x}, \boldsymbol{\lambda})$
UCB	$\mathfrak{s}_{\boldsymbol{\lambda}}\left(\mu^{(t)}(\boldsymbol{x}) + \sqrt{\beta_t}\sigma^{(t)}(\boldsymbol{x})\right),\ \beta_t = cd \ln t$
TS	$\mathfrak{s}_{\boldsymbol{\lambda}}(f'(\boldsymbol{x}))$, where $f'_k \sim \mathcal{GP}_k^{(t)},\ k \in [K]$

at each point which is much more expensive than computing the acquisitions for UCB or TS. Computing the expected hypervolume improvement in EHI is expensive and grows exponentially with K. PESM has a cost that is linear in K. However the computation involves performing expensive steps of expectation-propagation and MC estimates, which results in a large constant factor.

2.3 Regret Bounds

In this section we provide formal guarantees to prove upper bounds on the Bayes regret $\mathcal{R}_B$ which goes to zero as $T \rightarrow \infty$. We also show that our upper bound is able to recover regret bounds for single objectives when $K = 1$.

Analogous to the notion of regret in the single-objective setting [30], we first define the instantaneous and cumulative regrets for the multi-objective optimization. The *instantaneous regret* incurred by our algorithm in step t is defined as,

$$r(\boldsymbol{x}_t, \boldsymbol{\lambda}_t) = \max_{\boldsymbol{x} \in \mathcal{X}} \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x})) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t)), \tag{2.4}$$

where $\boldsymbol{\lambda}_t$ and $\boldsymbol{x}_t$ are the same as in Algorithm 2.1. The *cumulative regret* till step T is defined as,

$$\mathcal{R}_C(T) = \sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{\lambda}_t). \tag{2.5}$$

For convenience, we do not explicitly mention the dependency of $\mathcal{R}_C(T)$ on $\{\boldsymbol{x}_t\}_{t=1}^T$ and

$\{\boldsymbol{\lambda}_t\}_{t=1}^T$. Next, we will make a slight abuse of notation here and define $\mathcal{R}_B(T)$, the Bayes regret incurred till step T, as $\mathcal{R}_B(\boldsymbol{X}_T)$ (See Eq. (2.3)), where $\boldsymbol{X}_T = \{\boldsymbol{x}_t\}_{t=1}^T$.

We further define the *expected Bayes regret* as $\mathbb{E}\mathcal{R}_B(T)$, where the expectation is taken over the random process f, noise ε and any other randomness occurring in the algorithm. Similarly, we also define the *expected cumulative regret* as $\mathbb{E}\mathcal{R}_C(T)$, where the expectation is taken over all the aforementioned random variables and additionally $\{\boldsymbol{\lambda}_t\}_{t=1}^T$. We will show that the expected Bayes regret can be upper bounded by the expected cumulative regret, which can be further upper bounded using the maximum information gain.

Maximum Information Gain. The *maximum information gain (MIG)* captures the notion of information gained about a random process f given a set of observations. For any subset $A \subset \mathcal{X}$ define $\boldsymbol{y}_A = \{y_a = f(a) + \varepsilon_a | a \in A\}$. The reduction in uncertainty about a random process can be quantified using the notion of information gain given by $\mathbf{I}(\boldsymbol{y}_A; f) = \mathbf{H}(\boldsymbol{y}_A) - \mathbf{H}(\boldsymbol{y}_A|f)$, where $\mathbf{H}$ denotes the Shannon entropy. The maximum information gain after T observations is defined as

$$\gamma_T = \max_{A \subset \mathcal{X}: |A| = T} \mathrm{I}(\boldsymbol{y}_A; f). \tag{2.6}$$

Regret Bounds. We assume that $\forall k \in [K],\ t \in [T],\ \boldsymbol{x} \in \mathcal{X},\ f_k(\boldsymbol{x})$ follows a Gaussian distribution with marginal variances upper bounded by 1, and the observation noise $\varepsilon_{tk} \sim \mathcal{N}(0, \sigma_k^2)$ is drawn independently of everything else. Assume upper bounds $L_{\boldsymbol{\lambda}} \leq L,\ \sigma_k^2 \leq \sigma^2,\ \gamma_{Tk} \leq \gamma_T$, where γ_{Tk} is the MIG for the k th objective. When $\mathcal{X} \subseteq [0,1]^d$, the cumulative regret after T observations can be bounded as follows.

Theorem 2.1. *The expected cumulative regret for MOBO-RS after T observations can be upper bounded for both UCB and TS as,*

$$\mathbb{E}\mathcal{R}_C(T) = \mathcal{O}\left(L\left[\frac{K^2 T d \gamma_T \ln T}{\ln\left(1+\sigma^{-2}\right)}\right]^{1/2}\right). \tag{2.7}$$

The proof follows from Theorem 2.2 in the appendix. The bound for single-objective BO can be recovered by setting $K = 1$, which matches the bound of $\mathcal{O}(\sqrt{T d \gamma_T \ln T})$ shown in Russo and Van Roy [229], Srinivas et al. [252]. The proof is build on ideas for single objective analyses for TS and UCB [117, 229].

Under further assumption of the space Λ being a bounded subset of a normed linear space, and the scalarizations $\mathfrak{s}_{\boldsymbol{\lambda}}$ being Lipschitz in $\boldsymbol{\lambda}$, it can be shown that $\mathbb{E}\mathcal{R}_B(T) \leq \frac{1}{T}\mathbb{E}\mathcal{R}_C(T) + o(1)$, which combined with Theorem 2.1 shows that the Bayes regret converges to zero as $T \to \infty$. A complete proof can be found in Section 2.5.3.

2.4 Experimental Results

We experiment with both synthetic and real world problems. We compare our methods to the other existing MOO approaches in the literature: PESM, EHI, SMSego, ParEGO, and MOEA/D-EGO. EHI being computationally expensive is not feasible for more than two objectives. Other than visually comparing the results for three or lesser objectives we also compare them in terms of the Bayes regret defined in Eq. (2.3).

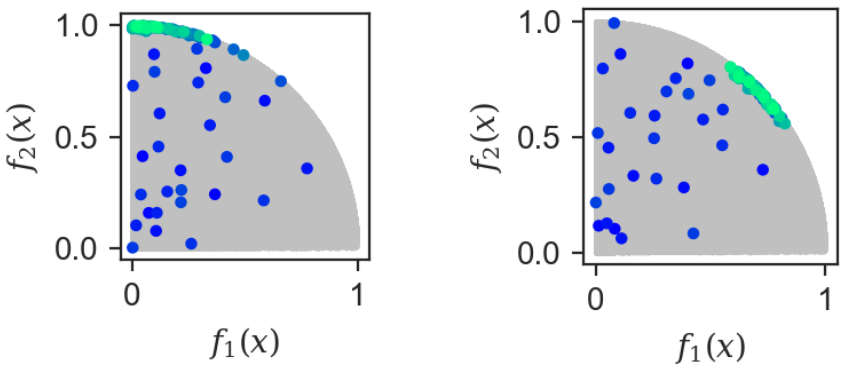


Figure 2.3: The feasible region is shown in grey. The color of the sampled points corresponds to the iteration they were sampled in, with brighter colors being sampled in the later iterations.

While our method is valid for any scalarization satisfying the Lipschitz and monotonicity conditions, we demonstrate the performance of our algorithm on two commonly used scalarizations, the linear and the Tchebyshev scalarizations [182] defined as,

$$\begin{aligned} \mathfrak{s}_{\boldsymbol{\lambda}}^{\text{lin}}(\boldsymbol{y}) &= \sum_{k=1}^{K} \boldsymbol{\lambda}_k \boldsymbol{y}_k, \\ \mathfrak{s}_{\boldsymbol{\lambda}}^{\text{tch}}(\boldsymbol{y}) &= \min_{k=1}^{K} \boldsymbol{\lambda}_k (\boldsymbol{y}_k - \boldsymbol{z}_k), \end{aligned} \tag{2.8}$$

where $\boldsymbol{z}$ is some reference point. In both cases, $\Lambda = \{\boldsymbol{\lambda} \succ \mathbf{0} \mid \|\boldsymbol{\lambda}\|_1 = 1\}$. It can be verified that the Lipschitz constant in both cases is upper bounded by 1.

Choosing the weight distribution $p(\boldsymbol{\lambda})$. While the user has the liberty to choose any distribution best suited for the application at hand, for demonstration we show one possible way. A popular way of specifying user preferences is by using bounding boxes [80], where the goal is to satisfy $f_k(\boldsymbol{x}) \in [a_k, b_k]$, $\forall 1 \leq k \leq K$. We convert bounding boxes to a weight distribution using a heuristic described below.

For the linear scalarization, it can be verified that the regret is minimized when $\boldsymbol{y}$ is pareto optimal, and the normal vector at the surface of the Pareto front at $\boldsymbol{y}$ has the same direction as $\boldsymbol{\lambda}$. This is illustrated using a simple example in Fig. 2.3. Consider two simple objectives $f_1(x, y) = xy, f_2(x, y) = y\sqrt{1 - x^2}$. Sampling $\boldsymbol{\lambda} = \left[\frac{u}{u+1}, \frac{1}{u+1}\right]$ where $u \sim \text{Unif}\,(0, 0.3)$, results in the first figure. In this example we have $\boldsymbol{\lambda}_1$ smaller than $\boldsymbol{\lambda}_2$, resulting in exploration of the region where $f_2(x, y)$ is high. Whereas sampling $\boldsymbol{\lambda} = \left[\frac{u}{u+v}, \frac{v}{u+v}\right]$ where $u, v \sim \text{Unif}\,(0.5, 0.7)$ results in the second figure since both components of $\boldsymbol{\lambda}$ have similar magnitudes. This idea leads to the following heuristic to convert bounding boxes to a sampling strategy. We sample as $\boldsymbol{\lambda} = \boldsymbol{u}/\|\boldsymbol{u}\|_1$ where $\boldsymbol{u}_k \sim \text{Unif}\,(a_k, b_k)\,,\ k \in [K]$. The intuition behind this approach is shown in Fig. 2.4. Such a weight distribution roughly samples points from inside the bounding box.

For the Tchebychev scalarization, at the optimum, $\boldsymbol{y} - \boldsymbol{z}$ is inversely proportional to $\boldsymbol{\lambda}$. For the purpose of demonstration and comparison we would like both the scalarization to obtain

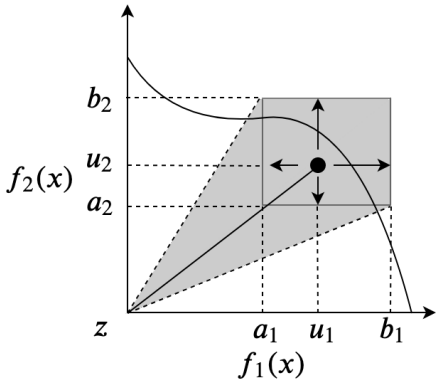


Figure 2.4: Weight distribution from bounding box.

similar objective values. Therefore, we reuse the $\boldsymbol{\lambda}$ sampled for the linear scalarization to get $\boldsymbol{\lambda}_{\text{tch}} = \boldsymbol{\lambda}'/\|\boldsymbol{\lambda}'\|_1$ where $\boldsymbol{\lambda}' = (1/\boldsymbol{\lambda}_1, \ldots, 1/\boldsymbol{\lambda}_K)$. We have normalized the vector so that it lies in Λ.

In order to explore the whole Pareto front, one can also specify a *flat* distribution. For instance consider the Dirichlet distribution on the simplex $\{\boldsymbol{x} \in \mathbb{R}^K \mid \sum_{k=1}^K \boldsymbol{x}_k = 1, \boldsymbol{x} \succ 0\}$. One can sample from the Dirichlet distribution as $\boldsymbol{\lambda} \sim \text{Dir}(1, \ldots, 1)$, which roughly provides equal weight to all the objectives leading to exploration of the whole Pareto front. Other strategies include $\boldsymbol{\lambda} = |\boldsymbol{\lambda}'|/\|\boldsymbol{\lambda}'\|_1$ where $\boldsymbol{\lambda}' \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

Other possible ways of choosing the weight vector includes learning the distribution of the weight vector from interactive user feedback. In fact, our framework also allows us to perform a joint posterior inference on the GP model and the weight distribution, thus learning the weight distribution in a more principled manner. While we leave these methods to future work, this demonstrates the flexibility of our framework.

Experimental Setup. For all our experiments, we use the squared exponential function as the GP kernel (in practice, this is a hyperparameter that must be specified by the user), given by $\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = s \exp\left(-\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2/(2\sigma^2)\right)$, where s and σ are parameters that are estimated during optimization. We perform experiments with both TS and UCB using both kinds of scalarizations. In Eq. (2.3), we observe that the term $\mathbb{E}_{\boldsymbol{\lambda}} \max_{\boldsymbol{x} \in \mathcal{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))$ is independent of the algorithm, hence it is sufficient to plot $-\mathbb{E}_{\boldsymbol{\lambda}} \max_{\boldsymbol{x} \in \boldsymbol{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))$. In all our experiments, we plot this expression, thus avoiding computing the global maximum of an unknown function. For the purposes of computing the Bayes simple regret, we linearly map the objective values to $[0, 1]$ so that the values are of reasonable magnitude. This however is not a requirement of our algorithm. Further experimental details can be found in Section 2.7.

Synthetic two-objective function. We construct a synthetic two-objective optimization problem using the Branin-4 and CurrinExp-4 functions as the two objectives respectively. These are the 4-dimensional counterparts of the Branin and CurrinExp functions [156], each mapping $[0, 1]^4 \to \mathbb{R}$. For this experiment we specify the bounding boxes $[(a_1, b_1), (a_2, b_2)]$. We sample from three different regions, which we label as *top*: $[(-110, -95), (23, 27)]$, *mid*:$[(-80, -70), (16, 22)]$, and *flat*: where we sample from a flat distribution. We also sample from a mixture of the *top* and *mid* distributions denoted by *top/mid*, thus demon-

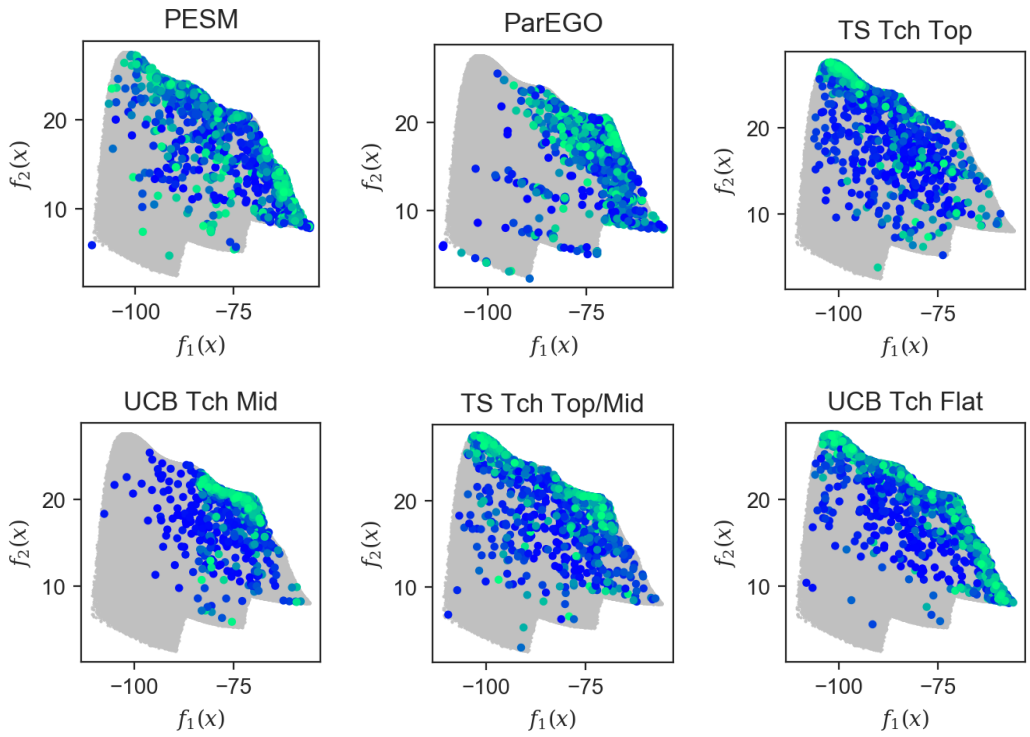


Figure 2.5: The feasible region is shown in grey. The color of the sampled points corresponds to the iteration they were sampled in, with brighter colors being sampled in the later iterations. The figure titles denote the method used and the region sampled. A complete set of results is presented in the Appendix.

strating sampling from non-connected regions in the Pareto front. Fig. 2.5 shows a scatter plot of the sampled values for the various methods. The simple regret plots are shown in Fig. 2.6.

Synthetic six-objective function. To show the viability of our method in high-dimensions, we sample six random functions $f_k : \mathbb{R}^6 \to \mathbb{R},\ f_k \sim \mathcal{GP}(\mathbf{0}, \kappa),\ k \in [6]$ where κ is the squared exponential kernel. Devoid of any domain knowledge about this random function, we linearly transform the objectives values to $[0, 1]$ for simplicity. We specify the bounding box as $[a_k, b_k] = [2/3, 1],\ \forall k \in [6]$ and denote it as the *mid* region, as the weight samples are of similar magnitude. The simple regret plot for this experiment is shown in Fig. 2.7.

Locality Sensitive Hashing. Locality Sensitive Hashing (LSH) [5] is a randomized algorithm for computing the k-nearest neighbours. LSH involves a number of tunable parameters: the number of hash tables, number of hash bits, and the number of probes to make for each query. The parameters affect the average query time, precision and memory usage. While increasing the number of hash tables results in smaller query times, it leads to an increase in the memory footprint. Similarly, while increasing the number of probes leads to a higher precision, it increases the query time. We explore the trade-offs between these three objectives.

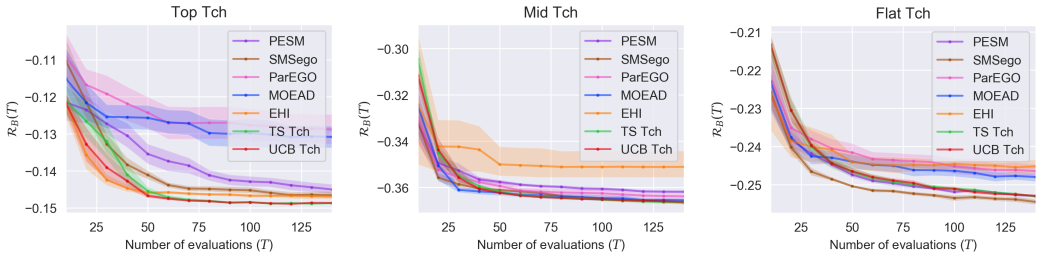
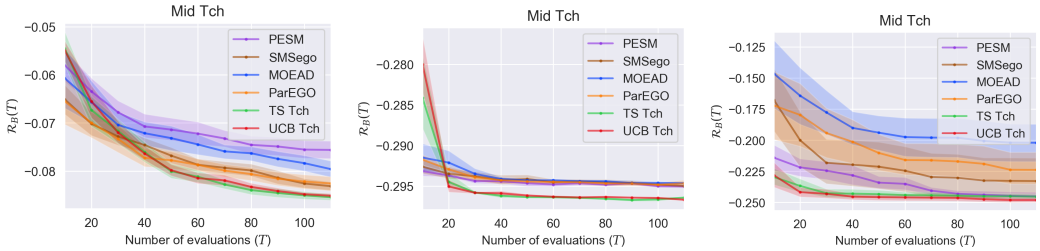


Figure 2.6: Bayes regret plots for the synthetic two-objective function. The mean and the 90% confidence interval were computed over 10 independent runs. The figure titles denote the sampling region and the scalarization used. We refer the reader to the appendix for results on linear scalarization.



(a) Synthetic 6x6 function (b) LSH Glove (c) Viola Jones

Figure 2.7: Bayes regret plots. The mean and the 90% confidence interval were computed over 5 runs. The figure titles denote the region sampled and the scalarization used. A complete set of plots can be found in the appendix.

We run LSH using the publicly available FALCONN library[2] on Glove word embeddings [204]. We use the Glove Wikipedia-Gigaword dataset trained on 6B tokens with a vocabulary size of 400K and 300-d embeddings. Given a word embedding, finding the nearest word embedding from a dictionary of word embeddings is a common task in NLP applications. We consider the following three objectives to minimize with their respective bounding boxes: Time (s) $[0.0, 0.65]$, 1−Precision $[0.0, 0.35]$, and the Memory (MB) $[0, 1600]$. The SR plots are shown in Fig. 2.7 and the sampled objective values in Fig. 2.8.

Viola Jones. The Viola Jones algorithm [272] is a fast stagewise face detection algorithm. At each stage a simple feature detector is run over the image producing a real value. If the value is smaller than a threshold the algorithm exits with a decision, otherwise the image is processed by the next stage and so on. The Viola Jones pipeline has 27 tunable thresholds. We treat these thresholds as inputs and optimize for Sensitivity, Specificity, and the Time per query. We consider the following three objectives to minimize with their bounding boxes: 1−Sensitivity $[0, 0.3]$, 1−Specificity $[0, 0.13]$, and Time per query $[0, 0.07]$. Fig. 2.7 shows the regret plot for this experiment.

Results and Discussion. Figs. 2.5 and 2.8 show the sampling patterns of our proposed approach for the synthetic 2-d and the LSH glove experiment. We observe that our approach

[2] `https://github.com/falconn-lib/falconn`

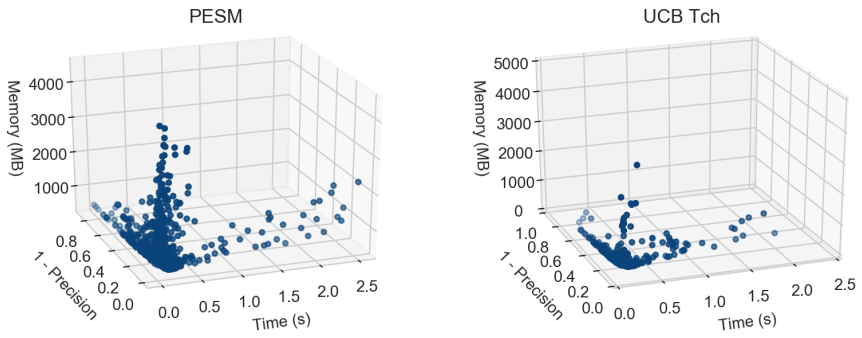


Figure 2.8: Sampled values for the LSH-Glove experiment over 5 independent runs. The figure titles denote the method used. A complete set of plots can be found in the appendix.

successfully samples from the specified region after some initial exploration, leading to a high concentration of points in the desired part of the Pareto front in the later iterations.

In Figs. 2.6 and 2.7 we observe that the proposed approach achieves a smaller or comparable regret compared to the other baselines. We notice that the improvement is most significant for the high dimensional experiments. A plausible explanation for this could be that learning high dimensional surfaces have a much higher sample complexity. However, our since our approach learns only a part of the Pareto front, it is able to achieve a small regret in a few number of samples, thus demonstrating the effectiveness of our approach.

2.5 Proofs

Russo and Van Roy [229] introduce a general approach to proving bounds on posterior sampling by decomposing the regret into two sums, one capturing the fact that the UCB upper bounds uniformly with high probability and the other that the UCB is not a loose bound. We begin by making a similar decomposition and bounding each of the other terms. Our proof for TS needs the assumption that the objectives are sampled independently from their respective priors. However, no such assumption is needed for UCB.

Denote by $\mathcal{H}_T$ the history until the $T-1$th round $\{(\boldsymbol{x}_t, \boldsymbol{y}_t, \boldsymbol{\lambda}_t)\}_{t=1}^{T-1}$. We assume $f_k \sim \mathcal{GP}(\mathbf{0}, \kappa_k)$ have marginal variances upper bounded by 1 for all $\boldsymbol{x} \in \mathcal{X}$ and $1 \leq k \leq K$. Let $\boldsymbol{x}_t^\star = \text{argmax}_{\boldsymbol{x}\in\mathcal{X}} \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}))$. Denote by $U_t(\boldsymbol{\lambda}, \boldsymbol{x}) = \mathfrak{s}_{\boldsymbol{\lambda}}(\mu^{(t)}(\boldsymbol{x}) + \sqrt{\beta_t}\sigma^{(t)}(\boldsymbol{x}))$, the UCB as defined in Table 2.1 where $\mu^{(t)}(\boldsymbol{x})$, $\sigma^{(t)}(\boldsymbol{x}) \in \mathbb{R}^K$ are the posterior means are variances of the K objectives at $\boldsymbol{x}$ in step t.

We first compute regret bounds for a finite domain $\mathcal{X}$, and then use a discretization argument to extend to continuous spaces. We begin by first proving the following decomposition of $\mathbb{E}\mathcal{R}_C(T)$, and then bound each of the decomposed terms.

Lemma 2.1. *For U_t as defined above, the following holds for both UCB and TS.*

$$\begin{aligned}\mathbb{E}\mathcal{R}_C(T) &= \mathbb{E}\left[\sum_{t=1}^{T}\left(\max_{\boldsymbol{x}\in\mathcal{X}}\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x})) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t))\right)\right] \\ &\leq \mathbb{E}\left[\sum_{t=1}^{T} U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t))\right] + \mathbb{E}\left[\sum_{t=1}^{T}\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t^\star)) - U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t^\star)\right]\end{aligned} \tag{2.9}$$

where $\boldsymbol{x}_t^\star = \operatorname{argmax}_{\boldsymbol{x}\in\mathcal{X}}\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}))$.

Proof. For UCB, we use the fact that at each step the next point to evaluate is chosen as $\boldsymbol{x}_t = \operatorname{argmax}_{\boldsymbol{x}\in\mathcal{X}} U_t(\boldsymbol{\lambda}_t, \boldsymbol{x})$. Thus, conditioned on the history $\mathcal{H}_t$, $U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t) \geq U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t^\star)$. The lemma follows using the tower property of expectation.

Thompson Sampling samples $f_1', \ldots, f_k'$ independently from the posterior in each iteration and produces an $\boldsymbol{x}_t$ maximizing $\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}))$. Making use of the independence assumption of the GP priors for TS, we observe that conditioned on the history $\mathcal{H}_t$, $\boldsymbol{x}_t$ has the same distribution as $\boldsymbol{x}_t^\star$, resulting in $\mathbb{E}[U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t)|\mathcal{H}_t] = \mathbb{E}[U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t^\star)|\mathcal{H}_t]$. We use the independence assumption only at this point in the proof and specifically for TS. □

Next we bound both the terms in the decomposition for finite $|\mathcal{X}|$, and then use a discretization based argument to prove for continuous sets $\mathcal{X}$.

2.5.1 Upper Bounds for Finite $|\mathcal{X}|$

Lemma 2.2. *For $\beta_t = 2\ln\left(\frac{t^2|\mathcal{X}|}{\sqrt{2\pi}}\right)$, and U_t as defined earlier, the following can be bounded as,*

$$\mathbb{E}\left[\sum_{t=1}^{T}\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t^\star)) - U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t^\star)\right] \leq \frac{\pi^2}{6}\mathbb{E}[L_{\boldsymbol{\lambda}}]K \tag{2.10}$$

Proof. We first see that,

$$\begin{aligned}\mathbb{E}\left[\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t^\star)) - U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t^\star)\right] &\leq \mathbb{E}\big(\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t^\star)) - U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t^\star)\big)_+ \\ &\leq \sum_{\boldsymbol{x}\in\mathcal{X}}\mathbb{E}\big(\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x})) - U_t(\boldsymbol{\lambda}_t, \boldsymbol{x})\big)_+\end{aligned}$$

where $(x)_+$ is defined as $\max(0, x)$. Using Lemma 2.6 and the definition of U_t we can further bound,

$$\mathbb{E}\big(\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x})) - U_t(\boldsymbol{\lambda}_t, \boldsymbol{x})\big)_+ \leq \ \mathbb{E}[L_{\boldsymbol{\lambda}}]\sum_{k=1}^{K}\mathbb{E}\Big[\big(f(\boldsymbol{x})_k - \mu_k^{(t)}(\boldsymbol{x}) - \sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x})\big)_+\Big].$$

Conditioned on $\mathcal{H}_t$, $f(\boldsymbol{x})_k - \mu_k^{(t)}(\boldsymbol{x}) - \sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x})$ follows a normal distribution $\mathcal{N}\left(-\sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x}),\ {\sigma_k^{(t)}}^2(\boldsymbol{x})\right)$.

Next we use the fact that for $X \sim \mathcal{N}(\mu, \sigma^2)$ and $\mu \leq 0$,

$$\mathbb{E}[X_+] \leq \frac{\sigma}{\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right). \tag{2.11}$$

Using the above,

$$\mathbb{E}\Big[\big(f(\boldsymbol{x})_k - \mu_k^{(t)}(\boldsymbol{x}) - \sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x})\big)_+ \mid \mathcal{H}_t\Big] \leq \frac{\sigma_k^{(t)}(\boldsymbol{x})}{\sqrt{2\pi}} \exp\left(-\frac{\beta_t}{2}\right) \leq \frac{1}{t^2|\mathcal{X}|}$$

Using the tower property of expectation, it follows that,

$$\mathbb{E}\big[\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t^\star)) - U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t^\star)\big] \leq \mathbb{E}[L_{\boldsymbol{\lambda}}]\frac{K}{t^2}$$

Summing over t, we get

$$\mathbb{E}\left[\sum_{t=1}^{T} \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t^\star)) - U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t^\star)\right] \leq \mathbb{E}[L_{\boldsymbol{\lambda}}]K\sum_{t=1}^{T}\frac{1}{t^2} \leq \frac{\pi^2}{6}\mathbb{E}[L_{\boldsymbol{\lambda}}]K$$

completing the proof. □

Lemma 2.3. *With the same conditions as in Lemma 2.2, it holds that*

$$\bar{L}_{\boldsymbol{\lambda}}\left(KT\beta_T\sum_{k=1}^{K}\frac{\gamma_{Tk}}{\ln(1+\sigma_k^{-2})}\right)^{1/2} + \frac{\pi^2}{6}\frac{K\mathbb{E}[L_{\boldsymbol{\lambda}}]}{|\mathcal{X}|} \tag{2.12}$$

where $\bar{L}_{\boldsymbol{\lambda}} = \mathbb{E}\left[\sqrt{\frac{1}{T}\sum_{t=1}^{T} L_{\lambda_t}^2}\right]$.

Proof. Conditioned on $\mathcal{C} = (\mathcal{H}_t, \boldsymbol{\lambda}_t, \boldsymbol{x}_t)$, the following holds using Lemma 2.6,

$$\begin{aligned}
&\mathbb{E}\big[U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t)) \mid \mathcal{C}\big] \\
&\leq \mathbb{E}\left[L_{\boldsymbol{\lambda}_t}\sum_{k=1}^{K}\Big(\mu_k^{(t)}(\boldsymbol{x}_t) + \sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x}_t) - f(\boldsymbol{x}_t)\Big)_+ \;\middle|\; \mathcal{C}\right] \\
&\leq \mathbb{E}\Bigg[L_{\boldsymbol{\lambda}_t}\sum_{k=1}^{K}\Big(\mu_k^{(t)}(\boldsymbol{x}_t) + \sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x}_t) - f(\boldsymbol{x}_t)\Big) \\
&\quad + L_{\boldsymbol{\lambda}_t}\sum_{k=1}^{K}\Big(f(\boldsymbol{x}_t) - \mu_k^{(t)}(\boldsymbol{x}_t) - \sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x}_t)\Big)_+ \;\Bigg|\; \mathcal{C}\Bigg] \\
&\leq \mathbb{E}\left[L_{\boldsymbol{\lambda}_t}\sum_{k=1}^{K}\sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x}_t) + \frac{KL_{\boldsymbol{\lambda}_t}}{t^2|\mathcal{X}|} \;\middle|\; \mathcal{C}\right]
\end{aligned}$$

where the last inequality follows from Eq. (2.11). Using the tower property of expectation,

$$\mathbb{E}\big[U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t))\big] \leq \mathbb{E}\left[L_{\boldsymbol{\lambda}_t}\sqrt{\beta_t}\sum_{k=1}^{K}\sigma_k^{(t)}(\boldsymbol{x}_t) + \frac{KL_{\boldsymbol{\lambda}_t}}{t^2|\mathcal{X}|}\right] \tag{2.13}$$

Summing over t, we get,

$$\begin{aligned}
&\mathbb{E}\left[\sum_{t=1}^{T} U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t))\right] \\
&\leq \mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{T} L_{\boldsymbol{\lambda}_t}\sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x}_t) + \sum_{t=1}^{T}\frac{KL_{\boldsymbol{\lambda}_t}}{t^2|\mathcal{X}|}\right] \\
&\leq \mathbb{E}\left[\sum_{k=1}^{K}\sum_{t=1}^{T} L_{\boldsymbol{\lambda}_t}\sqrt{\beta_t}\sigma_k^{(t)}(\boldsymbol{x}_t)\right] + \frac{\pi^2}{6}\frac{K\mathbb{E}[L_{\boldsymbol{\lambda}}]}{|\mathcal{X}|} \\
&\leq \mathbb{E}\left[\left(K\beta_T\sum_{t=1}^{T} L_{\boldsymbol{\lambda}_t}^2\right)^{1/2}\left(\sum_{k=1}^{K}\sum_{t=1}^{T}{\sigma_k^{(t)}}^2(\boldsymbol{x}_t)\right)^{1/2}\right] + \frac{\pi^2}{6}\frac{K\mathbb{E}[L_{\boldsymbol{\lambda}}]}{|\mathcal{X}|} \\
&\leq \mathbb{E}\left[\left(K\beta_T\sum_{t=1}^{T} L_{\boldsymbol{\lambda}_t}^2\right)^{1/2}\left(\sum_{k=1}^{K}\frac{\gamma_{Tk}}{\ln(1+\sigma_k^{-2})}\right)^{1/2}\right] + \frac{\pi^2}{6}\frac{K\mathbb{E}[L_{\boldsymbol{\lambda}}]}{|\mathcal{X}|}
\end{aligned}$$

where the second last step follows using Cauchy-Schwartz inequality, and the last step used the upper bound in terms of the MIG as shown in Srinivas et al. [252]. Substituting $\bar{L}_{\boldsymbol{\lambda}}$ gives us the desired result. □

Proposition 2.1. *The cumulative regret incurred in Algorithm 2.1 for both UCB and TS is upper bounded as,*

$$\bar{L}_{\boldsymbol{\lambda}}\left(KT\beta_T\sum_{k=1}^{K}\frac{\gamma_{Tk}}{\ln(1+\sigma_k^{-2})}\right)^{1/2} + \frac{\pi^2}{3}K\mathbb{E}[L_{\boldsymbol{\lambda}}] \tag{2.14}$$

Proof. The proof follows directly using Lemmas 2.1 to 2.3. □

2.5.2 Extending to continuous $\mathcal{X}$

We begin with the following result due to Ghosal and Roy [68]. For any differentiable stationary kernel κ with 4th order derivatives and $f \sim \mathcal{GP}(\mathbf{0}, \kappa)$, we have the following bound holds for some $a, b > 0$ such that for all $J > 0$, and for all $i \in \{1, \ldots, d\}$,

$$\mathbb{P}\left(\sup_x \left|\frac{\partial f(x)}{\partial x_i}\right| > J\right) \leq ae^{-(J/b)^2}. \tag{2.15}$$

Consider a continuous set $\mathcal{X}$ where $\mathcal{X} \subset \mathbb{R}^d$. For the sake of analysis, at each time step t we construct a finite discretization $\mathcal{X}_t$ of $\mathcal{X}$. $\mathcal{X}_t$ is constructed using a grid of uniformly spaced points with a distance of τ_j^{-1} between adjacent points in each coordinate. Therefore $|\mathcal{X}_t| = \tau_j^d$. Let $[\boldsymbol{x}]_t$ denote the point closest to $\boldsymbol{x}$ in $\mathcal{X}_t$. Let $M = \sup_{i\in\{1,\ldots,d\}}\sup_{\boldsymbol{x}\in\mathcal{X}}\left|\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|$. $\mathbb{E}\big[|f(\boldsymbol{x}) - f([\boldsymbol{x}]_t)|\big]$ can be bounded as

$$\begin{aligned}
\mathbb{E}\big[|f(\boldsymbol{x}) - f([\boldsymbol{x}]_t)|\big] &\leq \frac{d}{\tau_t}\mathbb{E}[M] \leq \frac{d}{\tau_t}\int_0^\infty \mathbb{P}(M \geq t)dt \\
&\leq \frac{d}{\tau_t}\int_0^\infty ae^{-(t/b)^2}dt = \frac{dab\sqrt{\pi}}{2\tau_t}
\end{aligned}$$

Let $A = \sup_{k=1}^{K} a_k,\ B = \sup_{k=1}^{K} b_k$ where $a_k,\ b_k$ correspond to the above constants for the kth objective. It follows that,

$$\mathbb{E}\big[|\mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x})) - \mathfrak{s}_{\boldsymbol{\lambda}}(f([\boldsymbol{x}]_t))|\big] \leq K\mathbb{E}[L_{\boldsymbol{\lambda}}]\frac{dAB\sqrt{\pi}}{2\tau_t} \tag{2.16}$$

We choose $\tau_t = t^2 dAB\sqrt{\pi}/2$ which gives us

$$\mathbb{E}\big[|\mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x})) - \mathfrak{s}_{\boldsymbol{\lambda}}(f([\boldsymbol{x}]_t))|\big] \leq K\mathbb{E}[L_{\boldsymbol{\lambda}}]\frac{1}{t^2} \tag{2.17}$$

Having bounded the approximation errors due to the discretization, we are in a position to use the framework developed for the finite case. We begin with a similar decomposition as Lemma 2.1, which includes the approximation factors [117]. For the continuous case, our treatment differs slightly for TS and UCB. We first look at the decomposition for UCB,

Lemma 2.4. *For the same conditions as in Lemma 2.1, and $[\boldsymbol{x}]_t$ as defined above, we have the following decomposition for TS,*

TS:

$$\mathbb{E}\mathcal{R}_C(T) \leq \underbrace{\mathbb{E}\left[\sum_{t=1}^{T}\mathfrak{s}_{\boldsymbol{\lambda}_t}(f([\boldsymbol{x}_t]_t)) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t))\right]}_{A1} + \underbrace{\mathbb{E}\left[\sum_{t=1}^{T}U_t(\boldsymbol{\lambda}_t, [\boldsymbol{x}_t]_t) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f([\boldsymbol{x}_t]_t))\right]}_{A2} +$$
$$\underbrace{\mathbb{E}\left[\sum_{t=1}^{T}\mathfrak{s}_{\boldsymbol{\lambda}_t}(f([\boldsymbol{x}_t^\star]_t)) - U_t(\boldsymbol{\lambda}_t, [\boldsymbol{x}_t^\star]_t)\right]}_{A3} + \underbrace{\mathbb{E}\left[\sum_{t=1}^{T}\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t^\star)) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f([\boldsymbol{x}_t^\star]_t))\right]}_{A4} \tag{2.18}$$

The proof is on the same lines as Lemma 2.1 using the fact that $\boldsymbol{x}_t^\star$ and $\boldsymbol{x}_t$ have the same distribution, when conditioned on the $\mathcal{H}_t$.

We now bound each of the individual terms. $A1$ and $A4$ can be bounded by $C_1 K\mathbb{E}[L_{\boldsymbol{\lambda}}]$ using Eq. (2.17)), for some global constant C_1. Let $\beta_t = \ln\left(\frac{t^2|\mathcal{X}_t|}{\sqrt{2\pi}}\right)$. $A2 + A3$ can be bounded in the same way as Lemmas 2.2 and 2.3 by considering the discretized set $\mathcal{X}_t$ at each time step t instead of $\mathcal{X}$.

For UCB, we have the following decomposition,

Lemma 2.5. *For the same conditions as in Lemma 2.1, and $[\boldsymbol{x}]_t$ as defined above, we have the following decomposition for UCB,*

UCB:

$$\mathbb{E}\mathcal{R}_C(T) \leq \underbrace{\mathbb{E}\left[\sum_{t=1}^{T}U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t))\right]}_{B1} +$$
$$\underbrace{\mathbb{E}\left[\sum_{t=1}^{T}\mathfrak{s}_{\boldsymbol{\lambda}_t}(f([\boldsymbol{x}_t^\star]_t)) - U_t(\boldsymbol{\lambda}_t, [\boldsymbol{x}_t^\star]_t)\right]}_{B2} + \underbrace{\mathbb{E}\left[\sum_{t=1}^{T}\mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t^\star)) - \mathfrak{s}_{\boldsymbol{\lambda}_t}(f([\boldsymbol{x}_t^\star]_t))\right]}_{B3} \tag{2.19}$$

The proof follows in a similar way as Lemma 2.1 using the fact that $U_t(\boldsymbol{\lambda}_t, \boldsymbol{x}_t) \geq U_t(\boldsymbol{\lambda}_t, [\boldsymbol{x}_t^\star]_t)$.

We now bound each of the decomposed terms by considering the discretized $\mathcal{X}_t$ in each step, and the corresponding β_t. $B3$ can be bounded in the same way as $A4$ using Eq. (2.17). $B1$ can be bounded using Lemma 2.3. $B2$ can be bounded in the same way as Lemma 2.2.

This leads us to the following theorem.

Theorem 2.2. *For* $\beta_t = \ln\left(\frac{t^2|\mathcal{X}_t|}{\sqrt{2\pi}}\right)$, *where* $|\mathcal{X}_t| = \left(\frac{t^2 dAB\sqrt{\pi}}{2}\right)^d$, *for some global constants* $C_1, C_2 > 0$, *the following holds for both UCB and TS,*

$$\mathbb{E}\mathcal{R}_C(T) \leq C_1 K \mathbb{E}[L_{\boldsymbol{\lambda}}] + C_2 \bar{L}_{\boldsymbol{\lambda}} \left(KT(d \ln T + d \ln d) \sum_{k=1}^{K} \frac{\gamma_{Tk}}{\ln(1+\sigma_k^{-2})} \right)^{1/2} \tag{2.20}$$

2.5.3 Upper bound on Bayes regret

At a high level, optimizing the cumulative regret optimizes the pointwise regret for the observed $\boldsymbol{\lambda}_t$. However, it generalizes to the unseen $\boldsymbol{\lambda}$ in $\mathcal{R}_B(T)$ that are *close* to the sampled $\boldsymbol{\lambda}_t$. This requires us to define a notion of closeness or a metric on Λ. We assume that Λ is a bounded subset of a $\mathbb{R}^D$. We make the assumption that $\mathfrak{s}_{\boldsymbol{\lambda}}(\boldsymbol{y})$ is J-Lipschitz in $\boldsymbol{\lambda}$ for all $\boldsymbol{y} \in \mathbb{R}^K$, that is,

$$|\mathfrak{s}_{\boldsymbol{\lambda}_1}(\boldsymbol{y}) - \mathfrak{s}_{\boldsymbol{\lambda}_2}(\boldsymbol{y})| \leq J \, \|\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2\|_1. \tag{2.21}$$

Conditioned on the history $\mathcal{H}_T$, consider the Wasserstein or Earth Movers distance [178, 271] $W_1(p, \widehat{p})$ between the sampling distribution $p(\boldsymbol{\lambda})$ defined on Λ, and the empirical distribution $\widehat{p}$ corresponding to the samples $\{\boldsymbol{\lambda}_t\}_{t=1}^T$,

$$W_1(p, \widehat{p}) = \inf_q \left\{ \mathbb{E}_q \|X - Y\|_1, \; q(X) = p, \; q(Y) = \widehat{p} \right\}, \tag{2.22}$$

where q is a joint distribution on the RVs X, Y, with marginal distributions equal to p and $\widehat{p}$. We then have,

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T} \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t)) - \mathbb{E}\left[\max_{x \in \boldsymbol{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))\right] &\leq \frac{1}{T}\sum_{t=1}^{T} \max_{\boldsymbol{x} \in \boldsymbol{X}} \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x})) - \mathbb{E}\left[\max_{\boldsymbol{x} \in \boldsymbol{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))\right] \\
&\leq \mathbb{E}_{q(X,Y)}\left[\max_{\boldsymbol{x} \in \boldsymbol{X}} \mathfrak{s}_Y(f(\boldsymbol{x})) - \max_{\boldsymbol{x} \in \boldsymbol{X}} \mathfrak{s}_X(f(\boldsymbol{x}))\right] \\
&\leq \mathbb{E}_{q(X,Y)}\left\{ J \|X - Y\|_1 \right\}
\end{aligned}$$

Taking the infimum w.r.t. q, and the expectation w.r.t. the history $\mathcal{H}_t$, we get,

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}_t))\right] - \mathbb{E}\left[\max_{x \in \boldsymbol{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))\right] \leq J \, \mathbb{E}W_1(p, \widehat{p}) \tag{2.23}$$

Using the fact that $\mathbb{E}\left[\max_{\boldsymbol{x} \in \mathcal{X}} \mathfrak{s}_{\boldsymbol{\lambda}}(f(\boldsymbol{x}))\right] = \mathbb{E}\left[\max_{\boldsymbol{x} \in \mathcal{X}} \mathfrak{s}_{\boldsymbol{\lambda}_t}(f(\boldsymbol{x}))\right]$, we get,

$$\mathbb{E}\mathcal{R}_B(T) \leq \underbrace{\frac{1}{T}\mathbb{E}\mathcal{R}_C(T)}_{\mathbf{I}} + J \underbrace{\mathbb{E}W_1(p, \widehat{p})}_{\mathbf{II}}. \tag{2.24}$$

As $T \to \infty$, **I** converges to zero at a rate of $\mathcal{O}^*(T^{-1/2})$[3] as given by Theorem 2.2. Term **II** converges to zero at a rate of $\mathcal{O}^*(T^{-1/D})$ when $D \geq 2$, under mild regulatory conditions as shown by Canas and Rosasco [33].

2.5.4 Auxilliary Results

Lemma 2.6. *Suppose $s : \mathbb{R}^D \to \mathbb{R}$ is L-Lipschitz in the ℓ_1-norm, and monotonically increasing in all coordinates. Then it holds that,*

$$\big(s(\boldsymbol{x}) - s(\boldsymbol{y})\big)_+ \leq L \sum_{d=1}^{D} (\boldsymbol{x}_d - \boldsymbol{y}_d)_+ \tag{2.25}$$

where $(x)_+$ is defined as $\max(0, x)$.

Proof. We first note that when $s(\boldsymbol{x}) \leq s(\boldsymbol{y})$, it holds trivially. Now we assume $s(\boldsymbol{x}) > s(\boldsymbol{y})$. Let $U_d,\ 0 \leq d \leq D$ be defined as

$$U_d = \begin{cases} s(\boldsymbol{x}), & \text{if } d = 0 \\ s(\boldsymbol{y}), & \text{if } d = D \\ s(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_d, \boldsymbol{x}_{d+1}, \ldots, \boldsymbol{x}_D), & \text{otherwise} \end{cases}$$

Then,

$$0 \leq s(\boldsymbol{x}) - s(\boldsymbol{y}) = \sum_{d=0}^{D-1} U_d - U_{d+1}.$$

Using the facts that $U_d,\ U_{d+1}$ differ only in the $d+1$ component, and that s is increasing in all the components, we get

$$\begin{aligned} 0 \leq s(\boldsymbol{x}) - s(\boldsymbol{y}) &\leq \sum_{d=0}^{D-1} |U_d - U_{d+1}| \mathbb{I}\big(\boldsymbol{x}_{d+1} - \boldsymbol{y}_{d+1} \geq 0\big) \\ &\leq \sum_{d=0}^{D-1} L|\boldsymbol{x}_{d+1} - \boldsymbol{y}_{d+1}| \mathbb{I}\big(\boldsymbol{x}_{d+1} - \boldsymbol{y}_{d+1} \geq 0\big) = L \sum_{d=1}^{D} (\boldsymbol{x}_d - \boldsymbol{y}_d)_+ \end{aligned}$$

concluding the proof. □

2.6 Additional Experiments

Fig. 2.9 shows the sampling patterns for all baselines, and all combinations of the sampling region, method and scalarization for our approach. Fig. 2.10 show the plots for the Bayes for all sampling regions and scalarizations for the two-objective problem. Fig. 2.11 shows the Bayes regret for all sampling regions and scalarizations for all the other multi-objective problems. Fig. 2.12 shows the sampled objective values for the LSH Glove experiment.

[3] $\mathcal{O}^*$ ignores logarithmic factors.

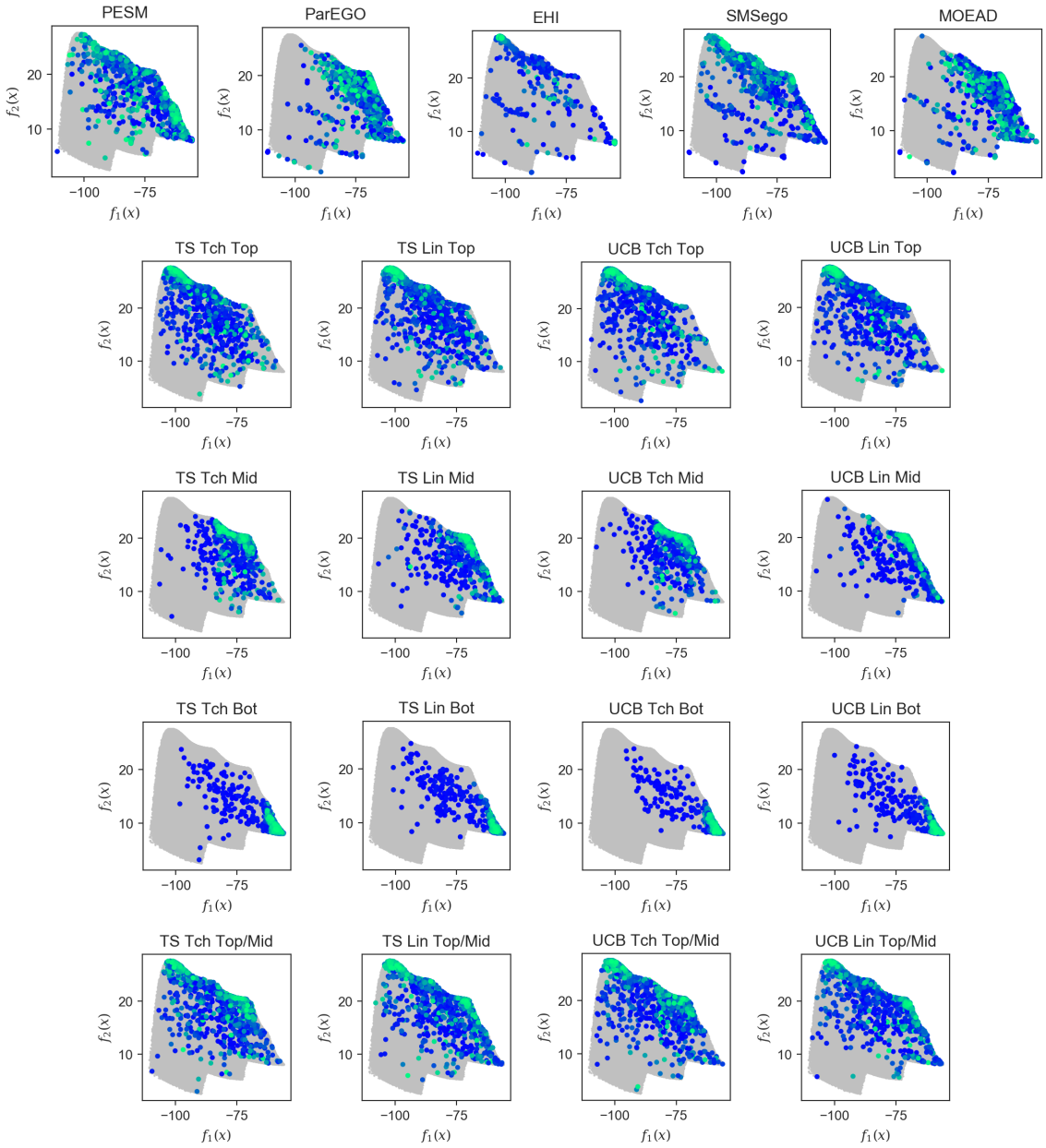


Figure 2.9: The plots show the sampled values for various algorithms and sampling regions. The feasible region is shown in grey. The color of the sampled points corresponds to the iteration in which they were sampled. Brighter colors were sampled in the later iterations. The figure titles denote the method used and the region sampled.

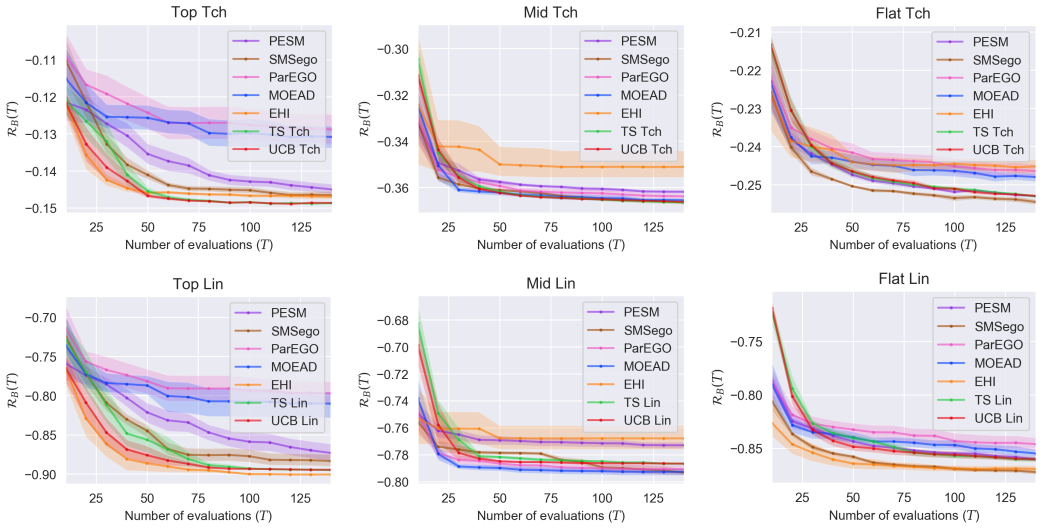


Figure 2.10: Bayes regret plots for the synthetic two-objective function. The mean and the 90% confidence interval were computed over 10 independent runs. The figure titles denote the sampling region and the scalarization used.

2.7 Implementation Details

The two-objective experiment was repeated 10 times with 150 iterations per run. All other experiments were repeated 5 time with 120 iterations per run. MOEA/D-EGO supports batch evaluation of points in every iteration. However, for fair comparison with the other methods, we set the batch size to 1.

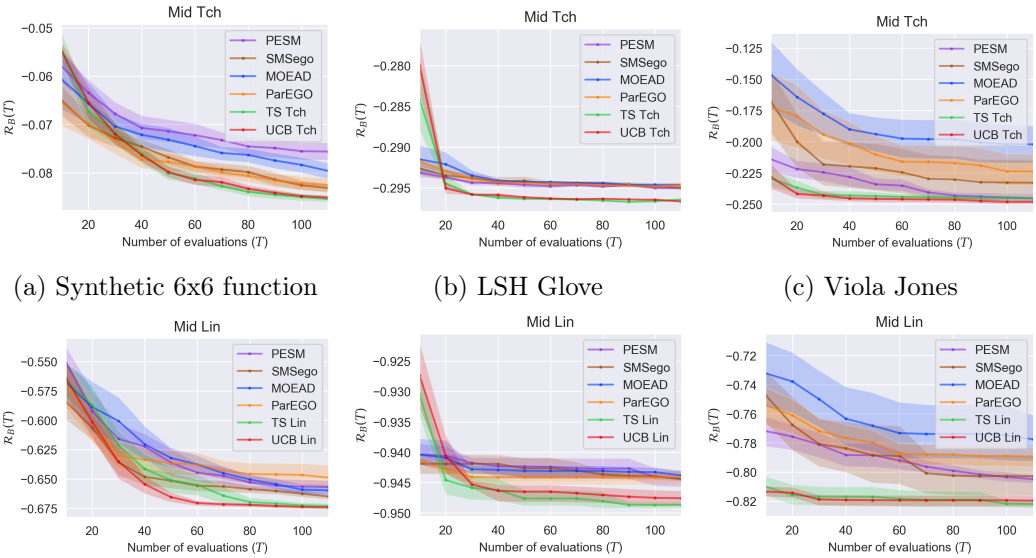
Domain space: For all our experiments we map the input domain appropriately such that $\mathcal{X} = [0, 1]^d$.

Initial evaluations: Similar to Kandasamy et al. [114], we randomly choose n_{init} initial points. We then evaluate the MO function at the initial points before using our optimization strategy.

Hyper-parameter estimation: To estimate the GP hyper-parameters, the GP is fitted to the observed data every 10 evaluations. We use the squared exponential kernel for all our experiments. We have a separate bandwidth parameter for each dimension of the input domain. The bandwidth, scale and noise variance are estimated by maximizing the marginal likelihood [218]. We set the mean of the GP as the median of the observations.

UCB parameter β_t: As discussed in Kandasamy et al. [114], β_t as suggested in [252] is too conservative in practice, and with unknown constants. Following the recommendation in [114], we use $\beta_t = 0.125 \log(2t + 1)$ for all our experiments.

Optimizing the acquisition function We use the DiRect algorithm [111] for optimizing the acquisition function in each iteration.



(a) Synthetic 6x6 function (b) LSH Glove (c) Viola Jones

(d) Synthetic 6x6 function (e) LSH Glove (f) Viola Jones

Figure 2.11: Bayes regret plots. The mean and the 90% confidence interval were computed over 5 runs. The figure titles denote the region sampled and the scalarization used.

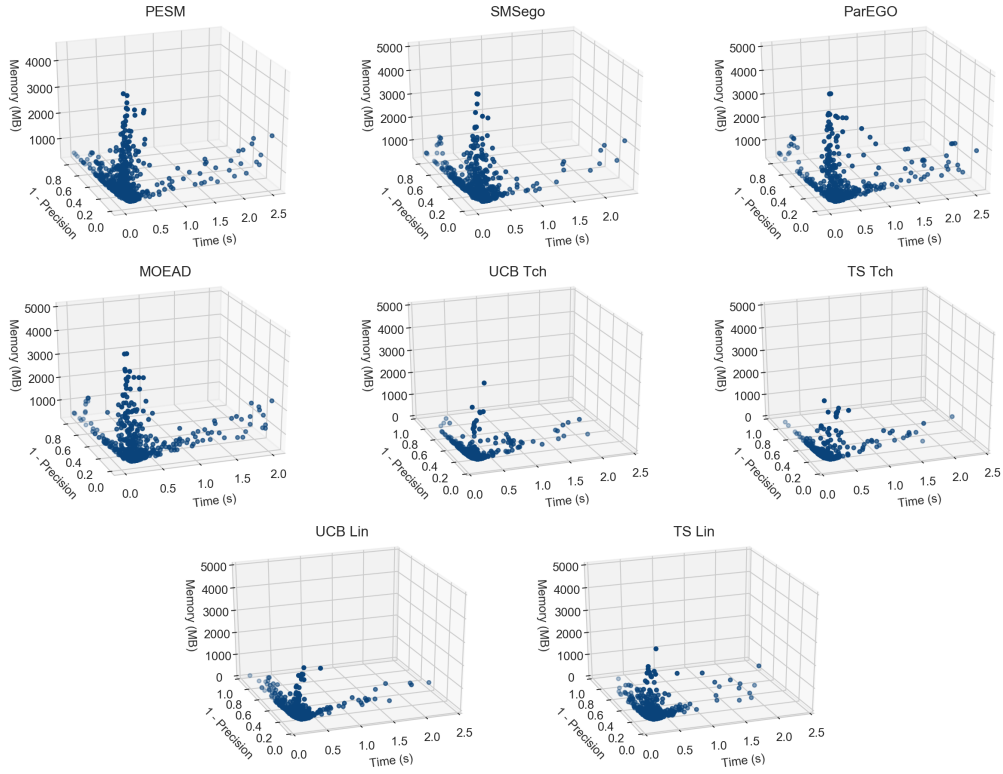


Figure 2.12: Sampled values for the LSH-Glove experiment over 5 independent runs. The figure titles denote the method used.

3 | On Greedy Algorithms for Black-box Optimization using Neural Networks

Bayesian Optimization (BO), and in particular GP optimization, has been quite successful in practice for black-box optimization and has become the de facto technique in industry [73]. The main reason for this success is the *simplicity* and *expressivity* of GPs; that is, GPs are easy to use, and impose minimal structural assumptions on the unknown blackbox function.

Despite its success, GP optimization faces issues with *scale* and *flexibility*. In particular, the computational complexity of GPs scales cubically with the number of function evaluations [154]. This makes GPs less appealing in high-dimensional search spaces when a large number of function evaluations are often needed to identify a good solution. In addition, GPs are not easily extendable to structured domains involving objects such as images, audio, text, and graphs [118, 266]. The design of appropriate kernels for such structured objects is problem specific and is still an ongoing research problem with state-of-the-art structured kernels not as performant as modern neural feature representations [125].

An emerging line of work has tried to address these issues by developing neural network (NN) based blackbox optimization techniques [125, 220, 250, 299]. These approaches use NNs as their surrogate models, and rely on standard acquisition functions such as EI [112], TS (*a.k.a* posterior sampling) [229, 260], UCB [3] to determine the next query point. The computational complexity of these approaches typically grows linearly with the number of function evaluations, making them attractive for high-dimensional problems. In addition, these techniques can be easily extended to structured domains involving objects such as images, audio, and graphs, for which we have neural architectures that can encode priors from the domain (for instance, convolutional neural networks).

One of the key challenges in designing NN based approaches lies in extending the classical ideas of EI, TS, UCB to neural networks. For instance, UCB and EI require construction of valid confidence intervals for the unknown function. While this is easy for simple models such as linear models [45], constructing such intervals is highly non-trivial for complex models such as neural networks. TS, on the other hand, require computing the posterior distribution of Bayesian Neural Networks (BNNs), which, again, is non-trivial [250]. Existing approaches overcome these issues by relying on a variety of heuristics. Snoek et al. [247], Xu

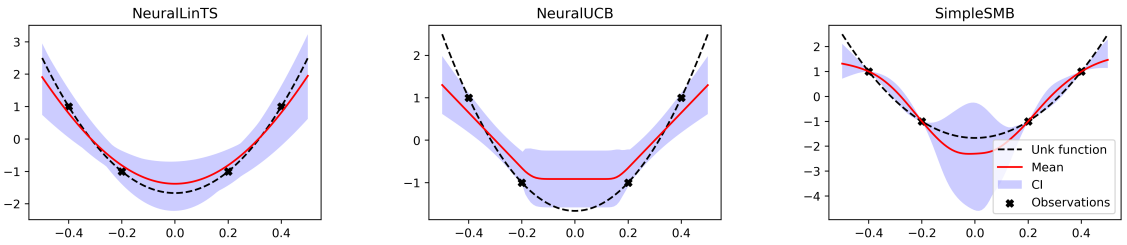


Figure 3.1: Plots show the confidence bands estimated by various neural network based BBO techniques on a $1D$ (noiseless) blackbox function. NeuralLinTS, NeuralUCB are the techniques developed by Snoek et al. [247], Zhou et al. [299] respectively. SimpleSMB is our greedy technique. It can be seen that our confidence bands are narrower for points close to observations.

et al. [289] perform LinearUCB/LinearTS on top of the representation of the last layer of the neural network. Zhang et al. [295], Zhou et al. [299] work in the infinite-width limit of NNs, where the networks can be viewed as linear models, and perform LinearUCB/LinearTS on the *linearized networks*. However, both these heuristics have several drawbacks: (a) they often lead to conservative confidence bands, which in turn reduces the speed of convergence of the algorithm (see Fig. 3.1), (b) some of these techniques are computation and memory intensive as they involve manipulation of large matrices with sizes quadratic in the number of NN parameters, in each iteration [295, 299], (c) some are geared towards finite search spaces and their extension to continuous domains is non-trivial [295]. Another heuristic that is often used involves mimicking the posterior distribution of BNNs using ensembles of NNs [125, 220]. However, this heuristic requires a large number of models in the ensemble to get a good approximation of the posterior distribution.

In this work, we show that one need not explicitly estimate the confidence sets or posterior distributions for NN based blackbox optimization. We show that a simple greedy algorithm which fits a NN to the current set of observations, and uses the learned network as the acquisition function achieves better performance than existing NN based approaches. A crucial aspect of our algorithm is that we train our neural network surrogate model from scratch in each iteration, and rely on stochastic gradient descent with randomly initialized weights. Our key insight is that such an approach mimics GP sampling and Thompson sampling in wide neural networks [86, 147].

3.1 Related Work

Global optimization of expensive blackbox functions is a well studied problem. Numerous techniques have been proposed for this problem over the years. We review some of the relevant literature below.

Structural Assumptions. One category of works impose structural assumptions on the unknown blackbox function f^*. For instance, Agrawal and Goyal [4], Filippi et al. [61] assume f^* is a linear function. Kveton et al. [141] assume f^* can be modeled using a generalized linear model (GLM). Several works assume convexity of f^* [2, 13, 18, 241].

No Structural Assumptions. The second category of works impose minimal assumptions on f. Bubeck et al. [29], Kleinberg et al. [130] impose Lipschitz continuity on f^*. In GPs, it is typically assumed that f^* lies in an RKHS [44, 251]. As previously mentioned GPs do not scale well to high-dimensional problems. Consequently, several works have attempted to speed-up GP inference [31, 154].

Neural network based approaches. A recent line of work assumes f^* is a neural network. While this might look like a structural assumption, it actually is not because NNs have the power to arbitrarily approximate any continuous function. Early works relied on Bayesian neural networks (BNN) to impose a prior over the unknown blackbox function [250]. These techniques relied on exact posterior sampling to compute acquisition functions such as EI. A variety of algorithms have been developed for posterior sampling on BNNs. These include Hamiltonian Monte Carlo [183], stochastic gradient Langevin MCMC [134], variational inference methods [75]. A drawback with these techniques is that they are very complex to implement in practice with a large number of hyper-parameters. In addition, many of these techniques provide conservative estimates of the uncertainty for points that are far from the observed data [250]. Recent works considered wide neural networks and relied on NTK theory to develop UCB and TS algorithms for neural networks [295, 299]. Another class of approaches have relied on heuristics such as performing LinearUCB, LinearTS using features from a neural network [247, 289] or relying on ensembles to mimic posterior sampling [125, 142].

Infinite-width limit of Neural Networks. Wide NNs and their infinite-width limits have gained attention of late. Lee et al. [146], Matthews et al. [168] showed that wide NNs at initialization are equivalent to GPs. Arora et al. [7], Jacot et al. [106] showed that training wide NNs (with random initialization that leads to zero or small initial outputs) using gradient descent is equivalent to performing kernel ridge regression with NTK kernel. Lee et al. [147] showed that gradient descent on randomly initialized wide NNs is equivalent to sampling from GPs. NTK theory was extended from feed forward networks to other architectures such as convolutional networks [7], graph neural networks [54]. Arora et al. [6] derived generalization bounds for 2 layer wide networks, that rely on NTK kernel. Eldan et al. [56] derived non-asymptotic rates for the speed at which finite-width NN training approaches the NTK regime.

3.2 Background

In this section, for the sake of completeness and introducing more compact notation, we recap some of the background about black-box optimization, Gaussian processes, and introduce some background about infinitely wide neural networks.

Blackbox Optimization. In blackbox optimization (BBO), our aim is to minimize a (potentially non-convex) function f^* over an action space $\mathcal{X}$, given only zeroth-order oracle access to the function. That is, the only information about f^* comes via querying the oracle at a point $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and observing $y = f^*(x) + \xi$. Here, ξ is the independent noise, sampled from the Gaussian distribution $\mathcal{N}(0, \sigma_*^2)$. The special case of $\sigma_*^2 = 0$ is called *noiseless* BBO.

Gaussian Processes. A GP over $\mathcal{X}$, denoted by $\mathrm{GP}(\mu, \mathcal{K})$, is a collection of random variables $\{f(x)\}_{x\in\mathcal{X}}$ such that the joint distribution of every finite subset $\{f(x_i)\}_{i=1}^n$ of them is multivariate Gaussian with mean $\mathbb{E}[f(x_i)] = \mu(x_i)$ and covariance $\mathrm{Cov}(f(x_i), f(x_j)) = \mathcal{K}(x_i, x_j)$ [217]. Here, $\mu, \mathcal{K}$ are the mean and covariance functions of the GP. We assume $\mu = 0$ for GPs not conditioned on the data.

In BBO, GPs are often used to place a prior distribution over the unknown blackbox function f^*. Suppose, we observe n datapoints $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where $y_i = f^*(x_i) + \xi_i$ is the output of the zeroth-order oracle when queried at x_i. Conditioned on $\mathcal{D}$, the posterior distribution over f^* is again a GP with the following mean and covariance functions

$$\mu_n(x) = \mathcal{K}_{x\mathcal{X}_n}\left(\mathcal{K}_{\mathcal{X}_n\mathcal{X}_n} + \sigma^2 I\right)^{-1}\mathcal{Y}_n, \quad \mathcal{K}_n(x, x') = \mathcal{K}_{xx'} - \mathcal{K}_{x\mathcal{X}_n}\left(\mathcal{K}_{\mathcal{X}_n\mathcal{X}_n} + \sigma^2 I\right)^{-1}\mathcal{K}_{\mathcal{X}_n x'},$$

where $\mathcal{X}_n = [x_i]_{i=1}^n, \mathcal{Y}_n = [y_i]_{i=1}^n$. Here, $\mathcal{K}_{x\mathcal{X}_n} \in \mathbb{R}^{1\times n}$ with i^{th} entry given by $\mathcal{K}(x, x_i)$, and $\mathcal{K}_{\mathcal{X}_n\mathcal{X}_n} \in \mathbb{R}^{n\times n}$ with $(i,j)^{th}$ entry given by $\mathcal{K}(x_i, x_j)$.

Acquisition functions play a key role in BBO, as optimizing them helps us decide the next query point. The acquisition function for Thompson sampling is given by $\alpha_{\mathrm{TS}}(x; \mathcal{D}) = f(x)$, where f is randomly sampled from the posterior $\mathrm{GP}(\mu_n, \nu_n^2\mathcal{K}_n)$. Here, ν_n is a scale parameter that controls the exploration.

Neural Networks. In this chapter, we study neural network based BBO algorithms. We consider feed-forward networks $f(x, \theta)$ with L hidden layers and d_l neurons in the l^{th} hidden layer. Such a neural network can be defined using the following recurrence relation

$$\alpha^{(l+1)}(x, \theta) = \phi\left(\frac{\sigma_W}{\sqrt{d_l}} W^{(l)}\alpha^{(l)}(x, \theta) + \sigma_b b^{(l)}\right), \quad f(x, \theta) = \frac{\sigma_W}{\sqrt{d_L}} W^{(L)}\alpha^{(L)}(x, \theta) + \sigma_b b^{(L)},$$

with $\alpha^{(0)}(x, \theta) = x$. Here, $W^{(l)} \in \mathbb{R}^{d_{l+1}\times d_l}$, $b^{(l)} \in \mathbb{R}^{d_{l+1}}$ are the weights of layer l, with $d_0 = d$, $d_{L+1} = 1$. ϕ is the elementwise nonlinearity (we set it to `tanh` in our experiments), and the hyper-parameters $\gamma = (\sigma_W, \sigma_b)$ are the weight and bias variances. This particular parameterization of neural network is called Neural Tangent Kernel (NTK) parameterization [106]. In this parameterization, the weights $\theta = \{W^{(\le L)}, b^{(\le L)}\}$ are initialized by generating i.i.d samples from $\mathcal{N}(0, 1)$. We consider this particular initialization scheme for our approach.

Infinite-width limit of NNs. Suppose the weights of a NN are initialized to θ_0 using the random initialization scheme described above. For sufficiently wide networks (*a.k.a.* NTK regime), the outputs $\{f(x, \theta_0)\}_{x\in\mathcal{X}'}$, for any finite set $\mathcal{X}'$, converge to a multivariate Gaussian distribution [146]. To be precise, such randomly initialized networks correspond to a GP with mean 0 and the following covariance function $\mathcal{K}^{\mathrm{NN}}(x, x') = \lim_{\min(d_1, d_2 \ldots d_L)\to\infty} \mathbb{E}[f(x, \theta_0) f(x', \theta_0)]$.

Now, let's suppose we train the network to minimize the following squared loss over the training dataset $\{(x_i, y_i)\}_{i=1}^n$: $\sum_{i=1}^n (f(x_i, \theta) - y_i)^2$. Suppose we randomly initialize the network at θ_0 and use gradient descent (GD) to minimize the objective. For sufficiently wide networks and small enough step-size, the GD iterates $\{\theta_t\}_{t>0}$ stay close to θ_0 [147]. Consequently, the NN can be well approximated using the following linear model: $f(x, \theta_t) \approx f(x, \theta_0) + \langle \nabla_\theta f(x, \theta_0), \theta_t - \theta_0 \rangle$. Letting $\phi(x) = \nabla_\theta f(x, \theta_0)$, the kernel

$\hat{\Theta}(x, x') = \langle \phi(x), \phi(x') \rangle$ is called the empirical NTK kernel. Note that $\hat{\Theta}$ is a random quantity. In the NTK regime, $\hat{\Theta}(x, x')$ converges in probability to a deterministic quantity $\Theta(x, x')$, which is called the NTK kernel [7]. As a consequence, GD on wide networks can be viewed as performing kernel regression in the Reproducing Kernel Hilbert Space (RKHS) associated with Θ. For large t, Lee et al. [147] showed that the neural network $f(\cdot, \theta_t)$ (which is a random function that depends on θ_0), can be viewed as being sampled from a GP with the following mean and covariance functions

$$\mu_n(x) = \Theta_{x\mathcal{X}_n} \Theta^{-1}_{\mathcal{X}_n\mathcal{X}_n} \mathcal{Y}_t, \quad \mathcal{K}_n^{\text{NNGD}}(x, x') = \mathcal{K}^{\text{NN}}(x, x') + \Theta_{x\mathcal{X}_n} \Theta^{-1}_{\mathcal{X}_n\mathcal{X}_n} \mathcal{K}^{\text{NN}}_{\mathcal{X}_n\mathcal{X}_n} \Theta^{-1}_{\mathcal{X}_n\mathcal{X}_n} \Theta_{\mathcal{X}_n x'} - \left(\Theta_{x\mathcal{X}_n} \Theta^{-1}_{\mathcal{X}_n\mathcal{X}_n} \mathcal{K}^{\text{NN}}_{\mathcal{X}_n x'} + h.c. \right),$$

where $+h.c.$ denotes the Hermitian conjugate of its preceding term, and $\mathcal{X}_n = [x_i]_{i=1}^n, \mathcal{Y}_n = [y_i]_{i=1}^n$ are the features and response variables in the training dataset. Note that $\mathcal{K}^{\text{NN}}, \Theta$, and $\mathcal{K}_n^{\text{NNGD}}$ all depend on the variance hyper-parameter γ.

3.3 Greedy Algorithm

We now present our greedy algorithm for Blackbox Optimization (BBO) (shown in Algorithm 3.1). Our algorithm has three key hyper-parameters: initialization weight variance γ, noise variance σ^2, and scale parameter ν. γ dictates the kind of surrogate models we fit to the data. Smaller values of γ lead to smoother surrogate models and larger values lead to less smoother models. The scale parameter ν controls the exploration-exploitation trade-off (set to 1 in our experiments). Larger values lead to more exploration.

There are two key steps in our algorithm, namely surrogate model building step (line 7 of Algorithm 3.1) and the acquisition step (line 8). Our acquisition step simply involves minimizing the learned surrogate model. The first surrogate model builder we consider is SimpleSMB which is described in Algorithm 3.2. For the noiseless case, where $\sigma^2 = 0$, this algorithm simply fits a neural network to the observed data by minimizing the squared ℓ_2 loss. The network is learned using (stochastic) gradient descent with the parameters randomly initialized using NTK initialization scheme described in Section 3.2. In the noisy case, the algorithm regularizes the learned network to prevent over-fitting. Without regularization, the learned network interpolates the data, which leads to undesirable behavior. To avoid this, we use two forms of regularization in our algorithm. The first one involves perturbing the response variables $\{y_i\}_{i=1}^n$ with Gaussian noise. The second one is the *distance-from-initialization* term (line 3) which biases the learned network to stay close to the initialization. Both these forms of regularization play a key role in our algorithm. In Section 3.3.1, we show that training the network in this way is equivalent to sampling from GPs in the infinite-width limit.

Algorithm 3.1 Neural Greedy

Input: NN initialization weight variance γ, noise variance σ^2, scale parameter ν, budget T, surrogate model building sub-routine: SurrModelBuilder

Initialization: $\mathcal{D}_0 = \{\}$
for $t = 1, \cdots, T_e$ **do** ▷ Random Exploration Phase
    Sample x_t randomly from the domain
    Query blackbox function at x_t and obtain y_t
    $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(x_t, y_t)\}$
for $t = T_e + 1, \cdots, T$ **do** ▷ Greedy Phase
    $\tilde{f}_t \leftarrow \text{SurrModelBuilder}(\mathcal{D}_{t-1}, \gamma, \sigma^2, \nu)$
    $x_t \leftarrow \text{argmin}_{x \in \mathcal{X}} \tilde{f}_t(x)$
    Query blackbox function at x_t to obtain y_t
    $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(x_t, y_t)\}$
return $y_{\text{best}} = \min(\{y_t\}_{t=1}^T)$

Algorithm 3.2 Simple Surrogate Model Builder (SimpleSMB)

Input: Data $\{(x_i, y_i)\}_{i=1}^{t-1}$, initialization weight variance γ, noise variance σ^2, scale parameter ν

Add i.i.d noise to targets: $y_i' = y_i + \nu\epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$
Initialize $\theta_0 \sim \text{NTK-Init}(\gamma)$ ▷ Random weight initialization
Solve $\min_\theta \sum_{i=1}^{t-1}(y_i' - \nu f(x_i, \theta))^2 + \sigma^2\nu^2\|\theta - \theta_0\|_2^2$ using GD/SGD with θ_0 as initialization, and obtain θ_t
return $\nu \times f(\cdot, \theta_t)$

Discussion. While we show that Algorithm 3.1 results in a sample from a GP, and works well in practice (as shown in Section 3.5), it doesn't perform posterior sampling with respect to any kernel. We propose a slight modification to SimpleSMB called PosteriorSMB based on recent developments in wide neural network theory [86] which results in a posterior sample with respect to the neural tangent kernel in the infinite-width limit. At a high level, this method perturbs the surrogate model of SimpleSMB with a random function that helps us sample from the posterior distribution. Further details about this method can be found in Section 3.4.

Both the forms of regularization used in Algorithm 3.2 have been studied in the literature in various contexts. For instance, Nagarajan and Kolter [181] showed that controlling $\|\theta - \theta_0\|_2$ helps in better generalization of the learned network. Kveton et al. [141] studied reward perturbations in the context of GLM bandits. However, reward perturbation alone doesn't suffice for NNs. Not using $\|\theta - \theta_0\|_2$ can lead to networks with poor generalization and wider confidence bands which slows down the convergence of bandit algorithm (see Section 3.3.1 for more details).

Kannan et al. [120] studied a greedy algorithm for linear contextual bandits. Unlike our algorithm which perturbs the response variables, their algorithm perturbs the context vectors (this corresponds to perturbing the actions $x \in \mathcal{X}$ in BBO setting). Riquelme et al. [220] studied a neural greedy algorithm for contextual bandits. Their algorithm constructs

the surrogate model by minimizing the following objective $\sum_{i=1}^{n}(y_i - f(x_i, \theta))^2$, and determines the next query point by minimizing the learned surrogate model. While this might look similar to our algorithm, their algorithm differs from ours in two crucial aspects: (a) their algorithm doesn't differentiate noisy and noiseless settings and uses the same surrogate model builder for both, (b) they only update their surrogate model once every 20 steps, and more importantly, they don't train the model from scratch in each iteration. They instead warm-start the training with the previous surrogate model. We note that training the surrogate model from scratch in each iteration, with random initialization, is crucial as it helps us explore the search space better (see Section 3.3.1). In a recent work, Papalexopoulos et al. [198] used a neural greedy algorithm to solve constrained, discrete BBO problems. However, they only considered noiseless setting and did not provide any theoretical insights for the algorithm.

In Algorithm 3.1 we have an exploration phase that lasts for T_e rounds. We note that this is not the same as the exploration phase of classical explore-then-commit (ETC) algorithms [245]. T_e in our algorithm is independent of T (typically ≤ 10). Whereas, the number of exploration rounds needed in ETC algorithms to achieve non-trivial regret guarantees (*i.e.,* $o(T)$ regret) scales polynomially with T.

3.3.1 Theoretical analysis of SimpleSMB in the infinite-width limit

In this section, we study our algorithm in the infinite-width limit of NNs. We show that in each iteration of Algorithm 3.1, the learned network θ_t is sampled from a GP.

Proposition 3.1. *Suppose the width of the NNs used in the Algorithm 3.1 approaches infinity; that is,* $\min(d_1, d_2 \dots d_L) \to \infty$. *Suppose GD with small enough step size is used to optimize the surrogate model objective (*i.e., *line 3 of Algorithm 3.2). Consider the* $(t+1)^{th}$ *iteration of the algorithm, for any* $t \geq T_e$. *Conditioned on* $\mathcal{D}_t$ *and the past randomness, the surrogate model* $\tilde{f}_{t+1}$ *can be viewed as being randomly sampled from a* GP *with the following mean and covariance functions*

$$\mu_t(x) = \Theta_{x\mathcal{X}_t}\left(\Theta_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)^{-1}\mathcal{Y}_t,$$

$$\mathcal{K}_t^{NNGD}(x, x') = \nu^2\mathcal{K}^{NN}(x, x') - \nu^2\left(\Theta_{x\mathcal{X}_t}\left(\Theta_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)^{-1}\mathcal{K}^{NN}_{\mathcal{X}_t x'} + h.c.\right)$$
$$+ \nu^2\Theta_{x\mathcal{X}_t}\left(\Theta_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)^{-1}\left(\mathcal{K}^{NN}_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)\left(\Theta_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)^{-1}\Theta_{\mathcal{X}_t x'},$$

where $\mathcal{K}^{NN}, \Theta$ *are defined in Section 3.2. Here* $\mathcal{X}_t = \{x_i\}_{i=1}^t$, *and* $\mathcal{Y}_t = \{y_i\}_{i=1}^t$.

Discussion. The variance function $\mathcal{K}_t^{\text{NNGD}}(x, x)$ is usually large at points far away from the observed points $\mathcal{X}_t$. This helps our algorithm explore unobserved areas of the action space.

Instead of retraining the surrogate model from scratch at each iteration, suppose we warm-start Algorithm 3.2 with the surrogate model from the previous step (as done in Riquelme et al. [220]). Then the variance of θ_{t+1} conditioned on the history is 0, and the algorithm wouldn't perform any exploration. This would in turn lead to poor performance. This is also evident in the experiments of Riquelme et al. [220]. Next, let's suppose we do not use the regularization term $\|\theta - \theta_0\|_2$ in Algorithm 3.2. Then a simple calculation shows that the mean function $\mu_t(x)$ is equal to $\Theta_{x\mathcal{X}_t}\Theta^{-1}_{\mathcal{X}_t\mathcal{X}_t}\mathcal{Y}_t$. When evaluated at the observed points $\mathcal{X}_t$, this gives us $\mu_t(\mathcal{X}_t) = \mathcal{Y}_t$. This shows that without the regularization term,

Algorithm 3.3 Posterior Corrected Surrogate Model Builder (PosteriorSMB)

Input: Data $\{(x_i, y_i)\}_{i=1}^{t-1}$, initialization weight variance γ, noise variance σ^2, scale parameter ν

1: Add i.i.d noise to targets $y_i' = y_i + \nu\epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$
2: Initialize $\theta_0, \tilde{\theta}_0 \sim$ NTK-Init(γ)
3: Set the last layer parameters in $\tilde{\theta}_0$ to 0.
4: Let $\delta(x) = \left\langle \nabla_\theta f(x, \theta_0), \tilde{\theta}_0 \right\rangle$
5: Solve $\min_\theta \sum_{i=1}^{t-1} (y_i' - \nu f(x_i, \theta) - \nu\delta(x_i))^2 + \sigma^2\nu^2\|\theta - \theta_0\|_2^2$ using GD/SGD with θ_0 as initialization, and obtain θ_t
6: **return** $\nu \times (f(\cdot, \theta_t) + \delta(\cdot))$

the mean function interpolates the noisy data and leads to over-fitted models. Next, let's suppose we do not perturb the targets $\{y_i\}_{i=1}^n$ in Algorithm 3.2. The covariance function $\mathcal{K}_t^{\text{NNGD}}(x, x')$ in this case is similar to the one in Proposition 3.1, except for one difference: $\left(\mathcal{K}^{\text{NN}}_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)$ in the last term is replaced by $\mathcal{K}^{\text{NN}}_{\mathcal{X}_t\mathcal{X}_t}$. That is, not perturbing the targets results in narrow confidence bands which can lead to poor exploration. This shows that both forms of regularization are important for the algorithm to achieve good performance.

While our algorithm samples from a GP in each iteration, it doesn't perform Thompson/posterior sampling. TS would require us to sample from a GP with the following mean and covariance functions

$$\mu_t(x) = \Theta_{x\mathcal{X}_t}\left(\Theta_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)^{-1}\mathcal{Y}_t, \quad \mathcal{K}_t(x, x') = \nu^2\Theta_{xx'} - \nu^2\Theta_{x\mathcal{X}_t}\left(\Theta_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)^{-1}\Theta_{\mathcal{X}_t x'}.$$

In Section 3.4, we present a modification to our algorithm which lets us sample from the above GP. Nevertheless, despite the lack of correspondence between our algorithm and GP-TS, our algorithm achieves better performance than existing neural network based BBO techniques (see Section 3.5).

3.4 Posterior Corrected Greedy Algorithm

In this section, we present a slight modification to Algorithm 3.2 that let's us perform posterior sampling in the infinite-width limit. This modification was originally proposed by He et al. [86] for constructing deep Bayesian ensembles. In this work, we use it for BBO. This modification involves adding a random perturbation $\delta(x) = \left\langle \nabla_\theta f(x, \theta_0), \tilde{\theta}_0 \right\rangle$ to the neural network $f(x, \theta)$. Here, $\theta_0, \tilde{\theta}_0$ are random weights generated using NTK initialization, with the last layer weights of $\tilde{\theta}_0$ set to 0. Observe that $\delta(x)$ doesn't have any trainable parameters. All the trainable parameters in $f(x, \theta)+\delta(x)$ come from the first term. The rest of the algorithm for building the surrogate model remains the same as Algorithm 3.2, and involves finding a θ that minimizes the regularized least squares objective (see Algorithm 3.3 for details).

3.4.1 Theoretical analysis of PosteriorSMB in the infinite-width limit

Similar to Section 3.3.1, we first present the following result which connects our algorithm to GPs and Thompson sampling. The proof of this proposition follows from a similar result

proved in He et al. [86].

Proposition 3.2. *Suppose Algorithm 3.1 is run with Algorithm 3.3 as the surrogate model builder. Suppose the width of the NNs used in the algorithm approaches infinity; that is,* $\min(d_1, d2 \dots d_L) \to \infty$. *Suppose GD with small enough step size is used to optimize the surrogate model objective (*i.e., *line 5 of Algorithm 3.3). Consider the* $(t+1)^{th}$ *iteration of the algorithm, for any* $t \geq T_e$. *Conditioned on* $\mathcal{D}_t$ *and the past randomness, the surrogate model* $\tilde{f}_{t+1}$ *can be viewed as being randomly sampled from a* GP *with the following mean and covariance functions*

$$\mu_t(x) = \Theta_{x\mathcal{X}_t} \left(\Theta_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)^{-1} \mathcal{Y}_t,$$
$$\mathcal{K}_t^{NNGD\text{-}PC}(x, x') = \nu^2 \Theta_{xx'} - \nu^2 \Theta_{x\mathcal{X}_t} \left(\Theta_{\mathcal{X}_t\mathcal{X}_t} + \sigma^2 I\right)^{-1} \Theta_{\mathcal{X}_t x'}.$$

This shows that our algorithm is equivalent to performing GP-TS with NTK kernel, in the infinite-width limit. We rely on this equivalence to derive regret bounds of our algorithm (Theorem 3.1). For the purpose of the theorem, we let the scale parameter ν vary with iteration, and let ν_t denote the scale parameter at iteration t. We assume that $T_e = 0$; that is, there is no exploration phase in the algorithm. If $T_e \neq 0$, our regret would have a $O(T_e)$ additive term. Next, we assume the NTK kernel Θ is bounded and satisfies $\sup_{x \in \mathcal{X}} \|\Theta(x, x)\| \leq 1$. This assumption is not very restrictive. If instead, $\sup_{x \in \mathcal{X}} \|\Theta(x, x)\| \leq c$, for some $c > 1$, our regret bounds would increase by a factor of c. Finally, we let $I_t = \max_{\mathcal{A} \subset \mathcal{X}, |\mathcal{A}| = t} I(y_{\mathcal{A}}; f_{\mathcal{A}})$ be the information gain between $f_A = [f(x)]_{x \in \mathcal{A}}$, and $y_{\mathcal{A}} = f_{\mathcal{A}} + \xi_{\mathcal{A}}$, where $f \sim \text{GP}(0, \Theta), \xi_{\mathcal{A}} \sim \mathcal{N}(0, \sigma^2 \nu_t^2 I)$.

Theorem 3.1. *Consider the setting of Proposition 3.2. Let* $\mathcal{X} \subseteq [0, r]^d$ *be a compact and convex set, and let* γ *be the initialization variance hyper-parameter used in Algorithm 3.1. Finally, let* $\mathcal{H}$ *be the RKHS associated with the NTK kernel* Θ. *Suppose the true blackbox function* f^* *is Lipschitz and satisfies* $\|f^*\|_{\mathcal{H}} \leq B$. *Suppose Algorithm 3.1 is run with the following hyper-parameters:* $\sigma^2 = 1 + \frac{2}{T}$, $\nu_t = B + \sigma_* \sqrt{2\left(I_{t-1} + 1 + \log(2/\delta)\right)}$. *Then with probability at least* $1 - \delta$, *the cumulative regret of our algorithm is upper bounded by* $O\left(\sqrt{(I_T + \log(2/\delta))\, d \log(BdT)} \left(\sqrt{T I_T} + B\sqrt{T \log(2/\delta)}\right)\right)$.

The proof of this Theorem can be found in the Appendix and relies on similar proof techniques as in Chowdhury and Gopalan [44]. Notice the regret bound depends on information gain I_T. This quantity can be bounded in terms of the eigen-spectrum of the NTK kernel Θ [264]. Let $\{\lambda_1 \geq \lambda_2 \geq \dots\}$ be the eigen-spectrum of Θ w.r.t uniform measure over $\mathcal{X}$. Then I_T depends on the tail function $B(n) = \sum_{t > n} \lambda_t$. Recent works have studied the tail function of NTK kernel for 2-layer networks [35, 66]. These results can be used to derive concrete regret bounds for Algorithm 3.1.

One can derive non-asymptotic versions of Theorem 3.1 using results of Arora et al. [7], Eldan et al. [56] which characterize the rate at which wide NN training converges to the infinite-width limit. Recent works have extended NTK theory to other architectures such as convolution and graph neural networks. These results can be used to derive regret bounds of our algorithm in the infinite-width limit of CNNs, GNNs.

Remark 3.1. *For NTK kernel of a 1-hidden layer network, one can rely on the results of Geifman et al. [66] to show that* $I_T = O(T^{1-d^{-1}})$. *Plugging this into the bound of Theorem 3.1, we obtain a regret bound of* $O(T^{3/2-d^{-1}})$ *which is vacuous. We note that the*

regret bounds of Zhang et al. [295], Zhou et al. [299] for NeuralUCB, NeuralTS also face this issue. Deriving a TS style algorithm that achieves non-vacuous regret bound in this setting is still an open problem.

3.5 Experiments

In this section, we present experimental results showing the effectiveness of the proposed BBO algorithms.

Experiment Setup. We run our experiments on a set of synthetic functions that are commonly used in blackbox optimization benchmarking and competitions [84, 158]. The functions we chose vary in dimensionality, modality, smoothness, and structure (more details about them can be found in Section 3.7.2). The budget T for each function was set based on the number of dimensions: functions of higher dimensions were allocated more rounds.

Baselines. We consider the following baselines in our experiments.

1. GP-EI [112]: this performs GP optimization with EI as the acquisition function. We chose EI over UCB and TS because EI has fewer hyper-parameters and is known to achieve similar performance as the other two [246]. We used squared-exponential kernel in our experiments, and relied on GPflow for the implementation [167]. GPflow automatically sets the kernel hyper-parameters using maximum likelihood estimation.
2. NeuralLinTS [289]: this baseline performs linear TS on top of the feature representations of NN. It has two hyper-parameters: variance of the prior (diagonal) covariance matrix γ and the observation noise variance σ^2. We do a grid search for γ over $\{10, 100, 1000\}$ and for σ^2 over $\{0.001, 0.01, 0.1\}$.
3. NeuralUCB [299]: this baseline constructs confidence bands that are motivated from NTK theory. It has two hyper-parameters: γ used to scale the size of the confidence interval, and λ a regularization parameter. We do grid search for γ over $\{0.01, 0.1\}$, and λ over $\{10^{-5}, \cdots, 10^{-2}\}$.
4. NeuralEnsembles [125]: this baseline builds an ensemble of m neural networks with identical architectures but different random initializations. This ensemble acts as a surrogate model. The different predictions of networks in the ensemble can be interpreted as samples from the posterior distribution. We use EI as the acquisition function over the predictions from the ensembles. Similar to Kim et al. [125] at each iteration, we warm-start the networks at their previous values and update the model using GD. In our experiments, we set $m = 10$ and train the model to convergence at each iteration.

For NeuralLinTS, NeuralEnsembles, we use a 1-hidden layer networks of widths 500 and 1000 respectively, as surrogate models. We use a smaller width for NeuralLinTS, since it requires inversion of a $d_f \times d_f$ matrix, where d_f is the dimension of the feature extracted from the neural network. For NeuralUCB, we can not use wide networks as it involves

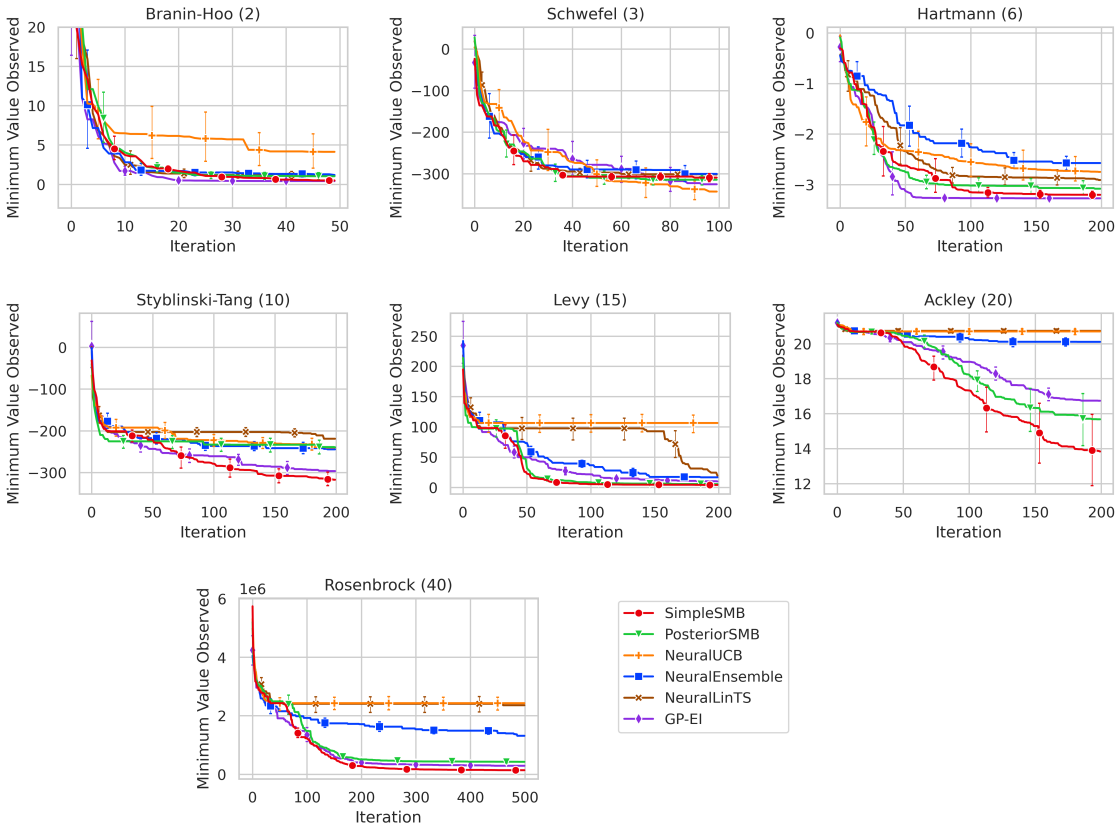


Figure 3.2: The plots show the minimum value observed over iterations for various baselines on the blackbox functions. The dimensionality of the blackbox functions is provided in parenthesis.

inversion of a $p \times p$ matrix, where p is the number of parameters in the NN. This can lead to computational and memory overflow issues. So we use smaller but deeper models. In particular, we use a 2-hidden layer network of width 20 as surrogate model. More details about our implementation can be found in Section 3.7. Finally, we note that we neither implement NeuralTS [295] as it is geared towards finite search spaces, nor NeuralLinUCB [289] as it has similar performance as NeuralLinTS.

Results. The plots for the minimum value observed over iterations is shown in Fig. 3.2 for the noiseless case. It can be seen that the SimpleSMB has the best performance over all the neural network based techniques, and has similar (if not better) performance as GP-EI. Moreover, while PosteriorSMB is better than all the neural baselines, it has slightly worse performance than SimpleSMB. The experimental results indicate the effectiveness of neural networks in BBO, especially in higher dimensions.

We also perform experiments with noisy versions of the blackbox functions. For all the points sampled by the algorithm, we consider the true value of the point (in contrast to the noisy value observed by the algorithm), and plot the minimum observed true value over iterations in Fig. 3.3. We notice similar trends as the noiseless plots shown in Fig. 3.2, with the notable exception of the Ackley function. We observe that neither of the methods are

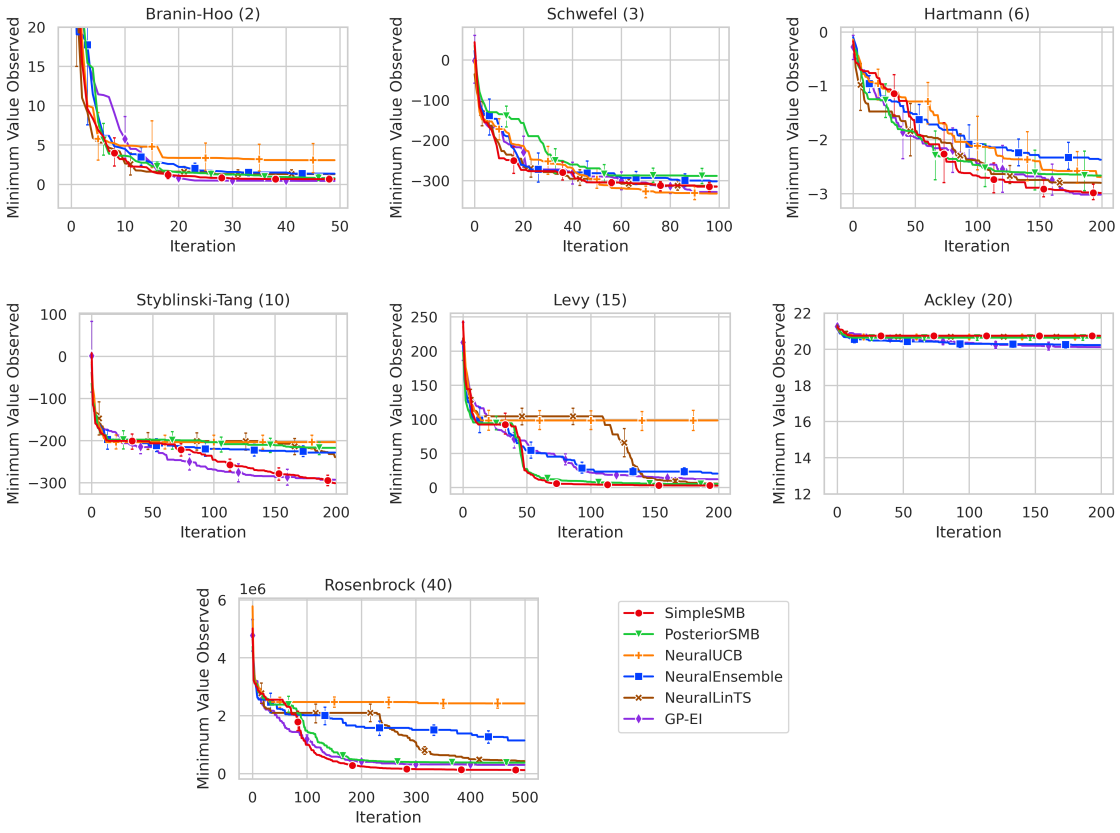


Figure 3.3: The plots show the minimum *true* value observed over iterations for all the compared methods on noisy version of the blackbox functions. The *true* value denotes the noise-free value of the black box function. The dimensionality of the blackbox functions is provided in parenthesis. The confidence intervals are based on 10 independent runs. While most of the plots follow similar trends as the noiseless plots in Fig. 3.2, we observe that none of the methods are able to reliably minimize the Ackley function.

able to reliably find the optimum of the Ackley function within the specified budget. We hypothesize that this is due to the fact that the Ackley function has a sharp global minima, making it even harder to find in the presence of noise.

3.6 Proofs

3.6.1 Proof of Proposition 3.1

Since we are in the infinite width limit, the iterates of GD with small enough step size stay close to the initialization θ_0 [147]. So, the following first order approximation is accurate: $f(x, \theta) = f(x, \theta_0) + \langle \phi(x), \theta - \theta_0 \rangle$, where $\phi(x) = \nabla_\theta f(x, \theta_0)$. Under this approximation, it suffices to study the following objective

$$\min_{\theta} \sum_{i=1}^{t} \left(y_i' - \nu f(x_i, \theta_0) - \nu \langle \phi(x_i), \theta - \theta_0 \rangle \right)^2 + \sigma^2 \nu^2 \|\theta - \theta_0\|_2^2.$$

This objective can equivalently be written as

$$\min_{\theta} \sum_{i=1}^{t} \left(\frac{y_i}{\nu} + \epsilon_i - f(x_i, \theta_0) - \langle \phi(x_i), \theta - \theta_0 \rangle \right)^2 + \sigma^2 \|\theta - \theta_0\|_2^2.$$

Let $\phi(\mathcal{X}_t) = [\phi(x_1)^T; \phi(x_2)^T \dots]$ be the matrix obtained by stacking the vectors $\{\phi(x_i)\}_{i=1}^t$ vertically. Let $f(\mathcal{X}_t, \theta) = [f(x, \theta)]_{x \in \mathcal{X}_t}$, $\mathcal{Y}_t = [y_i]_{i=1}^t$, and $\epsilon = [\epsilon_i]_{i=1}^t$. The minimizer θ_{t+1} of the above objective satisfies the following first order optimality conditions

$$\phi(\mathcal{X}_t)^T \left(\phi(\mathcal{X}_t)(\theta_{t+1} - \theta_0) + f(\mathcal{X}_t, \theta_0) - \epsilon - \frac{\mathcal{Y}_t}{\nu} \right) + \sigma^2(\theta_{t+1} - \theta_0) = 0.$$

Rearranging the terms, we get

$$\theta_{t+1} = \theta_0 + \left(\phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) + \sigma^2 I \right)^{-1} \phi(\mathcal{X}_t)^T \left(\frac{\mathcal{Y}_t}{\nu} + \epsilon - f(\mathcal{X}_t, \theta_0) \right).$$

Combining this with the linear approximation above, gives us the following expression for our surrogate model $\tilde{f}_{t+1}(x) = \nu f(x, \theta_{t+1})$

$$\nu f(x, \theta_0) + \left\langle \phi(x), \left(\phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) + \sigma^2 I \right)^{-1} \phi(\mathcal{X}_t)^T \left(\mathcal{Y}_t + \nu\epsilon - \nu f(\mathcal{X}_t, \theta_0) \right) \right\rangle.$$

Expectation. Consider any finite set $\mathcal{X}' \subseteq \mathcal{X}$. We now compute the mean function $\mu_t(\mathcal{X}') = \mathbb{E}\left[\tilde{f}_{t+1}(\mathcal{X}') | \mathcal{D}_t, \mathcal{H}_t \right]$, where $\mathcal{H}_t$ denotes the randomness from the past iterations.

$$\begin{aligned}
\mu_t(\mathcal{X}') = \nu \mathbb{E}\left[f(\mathcal{X}', \theta_0) | \mathcal{D}_t, \mathcal{H}_t \right] \\
&+ \nu \phi(\mathcal{X}') \left(\phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) + \sigma^2 I \right)^{-1} \phi(\mathcal{X}_t)^T \mathbb{E}\left[\epsilon - f(\mathcal{X}_t, \theta_0) | \mathcal{D}_t, \mathcal{H}_t \right] \\
&+ \phi(\mathcal{X}') \left(\phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) + \sigma^2 I \right)^{-1} \phi(\mathcal{X}_t)^T \mathcal{Y}_t.
\end{aligned}$$

Since $f(\cdot, \theta_0) \sim \mathrm{GP}(0, \mathcal{K}^{\mathrm{NN}})$ and ϵ is independent zero-mean Gaussian noise, the first two terms in the RHS above are 0. This shows that

$$\begin{aligned}
\mu_t(\mathcal{X}') &= \phi(\mathcal{X}') \left(\phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) + \sigma^2 I \right)^{-1} \phi(\mathcal{X}_t)^T \mathcal{Y}_t \\
&\overset{(a)}{=} \phi(\mathcal{X}') \phi(\mathcal{X}_t)^T \left(\sigma^2 I + \phi(\mathcal{X}_t) \phi(\mathcal{X}_t)^T \right)^{-1} \mathcal{Y}_t \\
&\overset{(b)}{=} \Theta_{\mathcal{X}'\mathcal{X}_t} (\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1} \mathcal{Y}_t,
\end{aligned}$$

where (a) follows from Woodbury matrix identity[1], and (b) follows from the fact that $\langle \phi(x), \phi(x') \rangle$ converges in probability to the NTK kernel $\Theta(x, x')$. This proves the first part of the Proposition.

[1] $(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)VA^{-1}$. Here, $A \in \mathbb{R}^{n \times n}, C \in \mathbb{R}^{k \times k}, U, V^T \in \mathbb{R}^{n \times k}$

Covariance. To simplify the notation, we define $M_t = \left(\phi(\mathcal{X}_t)^T\phi(\mathcal{X}_t) + \sigma^2 I\right)^{-1}$. Consider any finite set $\mathcal{X}' \subseteq \mathcal{X}$. The covariance function is given by

$$\begin{aligned}
\mathcal{K}_t^{\text{NNGD}}(\mathcal{X}', \mathcal{X}') &= \mathbb{E}\left[\left(\tilde{f}_{t+1}(\mathcal{X}') - \mu_t(\mathcal{X}')\right)\left(\tilde{f}_{t+1}(\mathcal{X}') - \mu_t(\mathcal{X}')\right)^T |\mathcal{D}_t, \mathcal{H}_t\right] \\
&\overset{(a)}{=} \nu^2\mathbb{E}\Big[\left(f(\mathcal{X}', \theta_0) - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T f(\mathcal{X}_t, \theta_0)\right) \\
&\qquad\qquad \left(f(\mathcal{X}', \theta_0) - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T f(\mathcal{X}_t, \theta_0)\right)^T |\mathcal{D}_t, \mathcal{H}_t\Big] \\
&\quad + \sigma^2\nu^2\phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T\phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T,
\end{aligned}$$

where (a) follows from the fact that θ_0, ϵ are independent of each other and $\mathbb{E}[\epsilon\epsilon^T] = \sigma^2 I$. Consequently, the cross terms involving θ_0, ϵ are equal to 0. First consider the second term in the RHS above. Using Woodbury matrix identity on M_t, together with the fact that $\langle\phi(x), \phi(x')\rangle$ converges in probability to $\Theta(x, x')$, it can be written as

$$\phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T\phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T = \Theta_{\mathcal{X}'\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-2}\Theta_{\mathcal{X}_t\mathcal{X}'}.$$

Next, consider the first term

$$\begin{aligned}
&\mathbb{E}\left[\left(f(\mathcal{X}', \theta_0) - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T f(\mathcal{X}_t, \theta_0)\right)\left(f(\mathcal{X}', \theta_0) - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T f(\mathcal{X}_t, \theta_0)\right)^T |\mathcal{D}_t, \mathcal{H}_t\right] \\
&= \mathbb{E}\left[f(\mathcal{X}', \theta_0)f(\mathcal{X}', \theta_0)^T|\mathcal{D}_t, \mathcal{H}_t\right] \\
&\qquad + \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T\mathbb{E}\left[f(\mathcal{X}_t, \theta_0)f(\mathcal{X}_t, \theta_0)^T|\mathcal{D}_t, \mathcal{H}_t\right]\phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T \\
&\qquad - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T\mathbb{E}\left[f(\mathcal{X}_t, \theta_0)f(\mathcal{X}', \theta_0)^T|\mathcal{D}_t, \mathcal{H}_t\right] \\
&\qquad - \mathbb{E}\left[f(\mathcal{X}', \theta_0)f(\mathcal{X}_t, \theta_0)^T|\mathcal{D}_t, \mathcal{H}_t\right]\phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T \\
&\overset{(a)}{=} \mathcal{K}^{\text{NN}}_{\mathcal{X}'\mathcal{X}'} + \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T\mathcal{K}^{\text{NN}}_{\mathcal{X}_t\mathcal{X}_t}\phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T \\
&\qquad - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T\mathcal{K}^{\text{NN}}_{\mathcal{X}_t\mathcal{X}'} - \mathcal{K}^{\text{NN}}_{\mathcal{X}'\mathcal{X}_t}\phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T \\
&\overset{(b)}{=} \mathcal{K}^{\text{NN}}_{\mathcal{X}'\mathcal{X}'} + \Theta_{\mathcal{X}'\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1}\mathcal{K}^{\text{NN}}_{\mathcal{X}_t\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1}\Theta_{\mathcal{X}_t\mathcal{X}'} \\
&\qquad - \Theta_{\mathcal{X}'\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1}\mathcal{K}^{\text{NN}}_{\mathcal{X}_t\mathcal{X}'} - \mathcal{K}^{\text{NN}}_{\mathcal{X}'\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1}\Theta_{\mathcal{X}_t\mathcal{X}'},
\end{aligned}$$

where (a) follows from the definition of $\mathcal{K}^{\text{NN}}$ (recall, $\mathcal{K}^{\text{NN}}(x, x') = \mathbb{E}[f(x, \theta_0)f(x', \theta_0)]$), and (b) follows from Woodbury matrix identity. Substituting the above two expressions in the expression for $\mathcal{K}_t^{\text{NNGD}}(\mathcal{X}', \mathcal{X}')$ gives us the required result.

3.6.2 Proof of Proposition 3.2

The proof of the Proposition relies on similar arguments as in the proof of Proposition 3.1. Since we are in the infinite width limit, the iterates of GD with small enough step size stay close to the initialization θ_0. So, the following first order approximation is accurate: $f(x, \theta) = f(x, \theta_0) + \langle\phi(x), \theta - \theta_0\rangle$, where $\phi(x) = \nabla_\theta f(x, \theta_0)$. Under this approximation, it suffices to study the following objective

$$\min_\theta \sum_{i=1}^t \left(y_i' - \nu f(x_i, \theta_0) - \nu\delta(x_i) - \nu\left\langle\phi(x_i), \theta - \theta_0\right\rangle\right)^2 + \sigma^2\nu^2\|\theta - \theta_0\|_2^2.$$

To simplify the notation, we define $f^{\text{pc}}(x, \theta_0) = f(x, \theta_0) + \delta(x)$. Using similar analysis as in Proposition 3.1, we get the following expression for our surrogate model $\tilde{f}_{t+1}(x) = \nu f(x, \theta_{t+1}) + \nu\delta(x)$

$$\nu f^{\text{pc}}(x, \theta_0) + \left\langle \phi(x), \left(\phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) + \sigma^2 I\right)^{-1} \phi(\mathcal{X}_t)^T \left(\mathcal{Y}_t + \nu\epsilon - \nu f^{\text{pc}}(\mathcal{X}_t, \theta_0)\right) \right\rangle.$$

The key result we now use is that the function $f^{\text{pc}}(\cdot, \theta_0)$ can be viewed as being sampled from $\text{GP}(0, \Theta)$ [86]. In contrast, $f(\cdot, \theta_0)$ can be viewed as being sampled from $\text{GP}(0, \mathcal{K}^{\text{NN}})$. The posterior correction term $\delta(x)$ essentially changes the covariance function from $\mathcal{K}^{\text{NN}}$ to Θ.

Expectation. Consider any finite set $\mathcal{X}' \subseteq \mathcal{X}$. We now compute the mean function $\mu_t(\mathcal{X}') = \mathbb{E}\left[\tilde{f}_{t+1}(\mathcal{X}') | \mathcal{D}_t, \mathcal{H}_t\right]$, where $\mathcal{H}_t$ denotes the randomness from the past iterations.

$$\begin{aligned}
\mu_t(\mathcal{X}') = \nu\mathbb{E}\left[f^{\text{pc}}(\mathcal{X}', \theta_0) | \mathcal{D}_t, \mathcal{H}_t\right] \\
+ \nu\phi(\mathcal{X}') \left(\phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) + \sigma^2 I\right)^{-1} \phi(\mathcal{X}_t)^T \mathbb{E}\left[\epsilon - f^{\text{pc}}(\mathcal{X}_t, \theta_0) | \mathcal{D}_t, \mathcal{H}_t\right] \\
+ \phi(\mathcal{X}') \left(\phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) + \sigma^2 I\right)^{-1} \phi(\mathcal{X}_t)^T \mathcal{Y}_t.
\end{aligned}$$

Since $f^{\text{pc}}(\cdot, \theta_0) \sim \text{GP}(0, \Theta)$ and ϵ is independent zero-mean Gaussian noise, the first two terms in the RHS above are 0. This shows that

$$\begin{aligned}
\mu_t(\mathcal{X}') &= \phi(\mathcal{X}') \left(\phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) + \sigma^2 I\right)^{-1} \phi(\mathcal{X}_t)^T \mathcal{Y}_t \\
&\overset{(a)}{=} \phi(\mathcal{X}') \phi(\mathcal{X}_t)^T \left(\sigma^2 I + \phi(\mathcal{X}_t)\phi(\mathcal{X}_t)^T\right)^{-1} \mathcal{Y}_t \\
&\overset{(b)}{=} \Theta_{\mathcal{X}'\mathcal{X}_t} (\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1} \mathcal{Y}_t,
\end{aligned}$$

where (a) follows from Woodbury matrix identity, and (b) follows from the fact that $\langle \phi(x), \phi(x') \rangle$ converges in probability to the NTK kernel $\Theta(x, x')$. This proves the first part of the Proposition.

Covariance. Using similar arguments as in Proposition 3.1, we get the following expression for the covariance function

$$\begin{aligned}
\mathcal{K}_t^{\text{NNGD-PC}}(\mathcal{X}', \mathcal{X}') = \nu^2\mathbb{E}\Big[\left(f^{\text{pc}}(\mathcal{X}', \theta_0) - \phi(\mathcal{X}') M_t \phi(\mathcal{X}_t)^T f^{\text{pc}}(\mathcal{X}_t, \theta_0)\right) \\
\left(f^{\text{pc}}(\mathcal{X}', \theta_0) - \phi(\mathcal{X}') M_t \phi(\mathcal{X}_t)^T f^{\text{pc}}(\mathcal{X}_t, \theta_0)\right)^T | \mathcal{D}_t, \mathcal{H}_t \Big] \\
+ \sigma^2\nu^2 \phi(\mathcal{X}') M_t \phi(\mathcal{X}_t)^T \phi(\mathcal{X}_t) M_t \phi(\mathcal{X}')^T,
\end{aligned}$$

where $M_t = \left(\phi(\mathcal{X}_t)^T\phi(\mathcal{X}_t) + \sigma^2 I\right)^{-1}$. Consider the first term in the RHS above

$$
\begin{aligned}
&\mathbb{E}\big[\left(f^{\text{pc}}(\mathcal{X}', \theta_0) - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T f^{\text{pc}}(\mathcal{X}_t, \theta_0)\right) \\
&\qquad \left(f^{\text{pc}}(\mathcal{X}', \theta_0) - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T f^{\text{pc}}(\mathcal{X}_t, \theta_0)\right)^T |\mathcal{D}_t, \mathcal{H}_t\big] \\
&= \mathbb{E}\left[f^{\text{pc}}(\mathcal{X}', \theta_0) f^{\text{pc}}(\mathcal{X}', \theta_0)^T |\mathcal{D}_t, \mathcal{H}_t\right] \\
&\qquad + \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T \mathbb{E}\left[f^{\text{pc}}(\mathcal{X}_t, \theta_0) f^{\text{pc}}(\mathcal{X}_t, \theta_0)^T |\mathcal{D}_t, \mathcal{H}_t\right] \phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T \\
&\qquad - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T \mathbb{E}\left[f^{\text{pc}}(\mathcal{X}_t, \theta_0) f^{\text{pc}}(\mathcal{X}', \theta_0) |\mathcal{D}_t, \mathcal{H}_t\right] \\
&\qquad - \mathbb{E}\left[f^{\text{pc}}(\mathcal{X}', \theta_0) f^{\text{pc}}(\mathcal{X}_t, \theta_0) |\mathcal{D}_t, \mathcal{H}_t\right] \phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T \\
&\overset{(a)}{=} \Theta_{\mathcal{X}'\mathcal{X}'} + \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T\Theta_{\mathcal{X}_t\mathcal{X}_t}\phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T \\
&\qquad - \phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T\Theta_{\mathcal{X}_t\mathcal{X}'} - \Theta_{\mathcal{X}'\mathcal{X}_t}\phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T \\
&\overset{(b)}{=} \Theta_{\mathcal{X}'\mathcal{X}'} + \Theta_{\mathcal{X}'\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1}\Theta_{\mathcal{X}_t\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1}\Theta_{\mathcal{X}_t\mathcal{X}'} \\
&\qquad - \Theta_{\mathcal{X}'\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1}\Theta_{\mathcal{X}_t\mathcal{X}'} - \Theta_{\mathcal{X}'\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-1}\Theta_{\mathcal{X}_t\mathcal{X}'},
\end{aligned}
$$

where (a) follows from the fact that $f^{\text{pc}}(\cdot, \theta_0)$ is sampled from $\text{GP}(0, \Theta)$, and (b) follows from Woodbury matrix identity. Next, consider the second term. Using similar arguments as in Proposition 3.1, it can be rewritten as

$$
\phi(\mathcal{X}')M_t\phi(\mathcal{X}_t)^T\phi(\mathcal{X}_t)M_t\phi(\mathcal{X}')^T = \Theta_{\mathcal{X}'\mathcal{X}_t}(\sigma^2 I + \Theta_{\mathcal{X}_t\mathcal{X}_t})^{-2}\Theta_{\mathcal{X}_t\mathcal{X}'}.
$$

Substituting the above two expressions in the expression for $\mathcal{K}_t^{\text{NNGD-PC}}(\mathcal{X}', \mathcal{X}')$ gives us the required result.

3.6.3 Proof of Theorem 3.1

The proof follows from the regret bound of GP-TS derived by Chowdhury and Gopalan [44] (similar proof techniques have been used to derive regret bounds of TS in various contexts including linear, generalized linear models [141]). We provide a high level argument here for the sake of completeness. For technical reasons, we consider a discretization of $\mathcal{X}$, instead of directly working with $\mathcal{X}$ (*i.e.,* we replace the domain $\mathcal{X}$ with its discretized version in the algorithm). In particular, in the t^{th} iteration of the algorithm, we consider the discretization $\mathcal{X}^{(t)} \subset \mathcal{X}$ which satisfies the following property: $\forall x \in \mathcal{X},\ |f^*(x) - f^*([x]_t)| \leq \frac{1}{t^2}$, where $[x]_t$ is the closest point to x in $\mathcal{X}^{(t)}$. Such a discretization can be achieved because of the fact that f^* is Lipschitz continuous.

The instantaneous regret at any iteration t is given by

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

where $[x^*]_t$ is the point closest to x^* in the discretization $\mathcal{X}^{(t)}$, and $\Delta_t(x_t) = f^*([x^*]_t) - f^*(x_t)$. To bound the regret, it suffices to bound $\sum_{t=1}^T \Delta_t(x_t)$. To this end, we partition the action space $\mathcal{X}^{(t)}$ into two sets namely, saturated and unsaturated sets. The saturated set consists of actions which satisfy the following condition: $\Delta_t(x) \geq c_t\sigma_{t-1}(x)$, where $\sigma_t(x) = \sqrt{\mathcal{K}_t^{\text{NNGD-PC}}(x, x)}$ (intuitively this consists of actions that are clearly sub-optimal). The unsaturated sets consist of points which satisfy the following condition: $\Delta_t(x) < c_t\sigma_{t-1}(x)$.

The key technical part of the proof involves showing that the probability of playing a saturated action is small. Once we have this result, the regret of the algorithm can be bounded as $O(\sum_{t=1}^{T} c_t \sigma_{t-1}(x_t))$. As proved by Chowdhury and Gopalan [44], this quantity is upper bounded by $O(\sqrt{I_T T})$, which is our desired regret bound.

3.7 Additional Experimental Details

3.7.1 Implementation details

Exploration budget. The exploration budget T_e was set to $5d$ where d is the input dimension, we also clip the exploration budget to lie between 2.5% and 7.5% of the total budget, so as to not exhaust most of the budget on exploration. All experiments in this chapter are repeated 10 times with different seeds, and the average and standard deviation of the results are reported.

Neural network implementation details. We primarily use 1-hidden layer neural architectures with a *large* number (> 1000) of hidden nodes, with the exception of neural UCB, for which we use a 2-layer and 20-wide neural networks as described in [299]. The weights of the neural network are initialized with independent samples from a normal distribution $\mathcal{N}(0, \gamma^2/\text{layer input size})$. We initialize the biases of the hidden layers with independent samples from a normal distribution $\mathcal{N}(0, \gamma^2)$. We initialize the bias of the final layer to 0, as is standard in neural network training. As described in the main text, γ is a hyper-parameter that we tune for our proposed approaches SimpleSMBand PosteriorSMB. We set $\gamma = 1$ for the baselines (as done in the original papers).

We implemented the above in python using Jax [26]. The networks (*i.e.,* surrogate models) were trained using the Adam optimizer [126] with a fixed learning rate of 0.001. We did not notice any practical difference from SGD other than faster convergence. All our experiments were run on a server with a 40 core Intel Xeon Silver 4210 CPU @ 2.20GHz and 187 GB memory.

Acquisition optimization. Optimizing the acquisition function is an essential step in all of the considered approaches. The acquisition function is built from the observed data. It takes as input a point from the input domain and produces a scalar value denoting the informativeness of the point. The acquisition function is optimized to select the most informative point to query next. Examples of acquisition functions include the upper confidence bound [251] and expected improvement [112].

Acquisition functions over continuous domains are typically differentiable and amenable to gradient based optimizers. We used standard gradient descent to optimize the acquisition function. We initialize the starting point with a uniform random sample from the input domain and perform 500 gradient descent steps with a fixed learning rate of 0.01. We repeat this process 10 times and return the best point found.

Hyper-parameters and acquisition functions. For each of these methods, we fix the number of hidden layers and the width. The rest of the tunable hyper-parameters are

described below. Each hyper-parameter is tuned w.r.t. the cumulative regret over the last $T/2$ steps of the algorithm, where T is the total budget.

SimpleSMB and PosteriorSMB: We set $\nu = 1, \sigma^2 = 0$, and set $\sigma_W = \sigma_b$ within γ. The only tuned hyper-parameter is the initialization variance parameter γ. We tune γ in the range $[0.5, 5.0]$. For our methods, we use 1-hidden layer networks of fixed width 5000 as surrogate models. The effect of the initialization variance on the performance of SimpleSMBis summarized in Figs. 3.4 and 3.5 for the noiseless and noisy cases respectively. The aquisition function in this case is the surrogate function returned from Algorithms 3.2 and 3.3.

NeuralUCB: The hyper-parameters for this method include γ, the scale of the confidence interval (Algorithm 1 in [299]) and λ the regularization parameter while training the surrogate model on observations (Algorithm 2 in [299]). We tune the hyper-parameters over the sets $\gamma \in \{0.01, 0.1, 1.0\}$ and $\lambda \in \{10^{-e}\}_{e=2}^{5}$.

NeuralEnsemble: This approach requires no hyper-parameters. We train 10 independent neural networks to convergence in each iteration. The neural networks are warm started with the weights from the previous iteration.

NeuralLinTS: Following Riquelme et al. [220], in this method we first train the neural network on the observed data, extract the final hidden layer activations, and use them for Bayesian linear regression. We assume a diagonal prior covariance $\gamma\mathbb{I}$ on the linear regression weights, and a Gaussian zero mean noise with variance σ^2. We tune the hyper-parameters over the sets $\gamma \in \{10, 100, 1000\}$ and $\sigma^2 \in \{0.001, 0.01, 0.1\}$. We vary γ over relatively large values in order to not regularize the regression weights too much.

GP-EI: For this, we estimate the GP kernel parameters from the data by maximizing the marginal likelihood of the observations [217]. We use the expected improvement acquisition function [112] with the estimated kernel parameters to select the next observation point.

3.7.2 Benchmark black-box functions

We use a set of commonly used benchmark synthetic black-box functions to evaluate our algorithm. Low-dimensional visualizations of the function are shown in Fig. 3.6. We refer the reader to the excellent repository of benchmarking functions by Surjanovic and Bingham [255] available at `https://www.sfu.ca/~ssurjano/optimization.html`, for the exact expression of these functions.

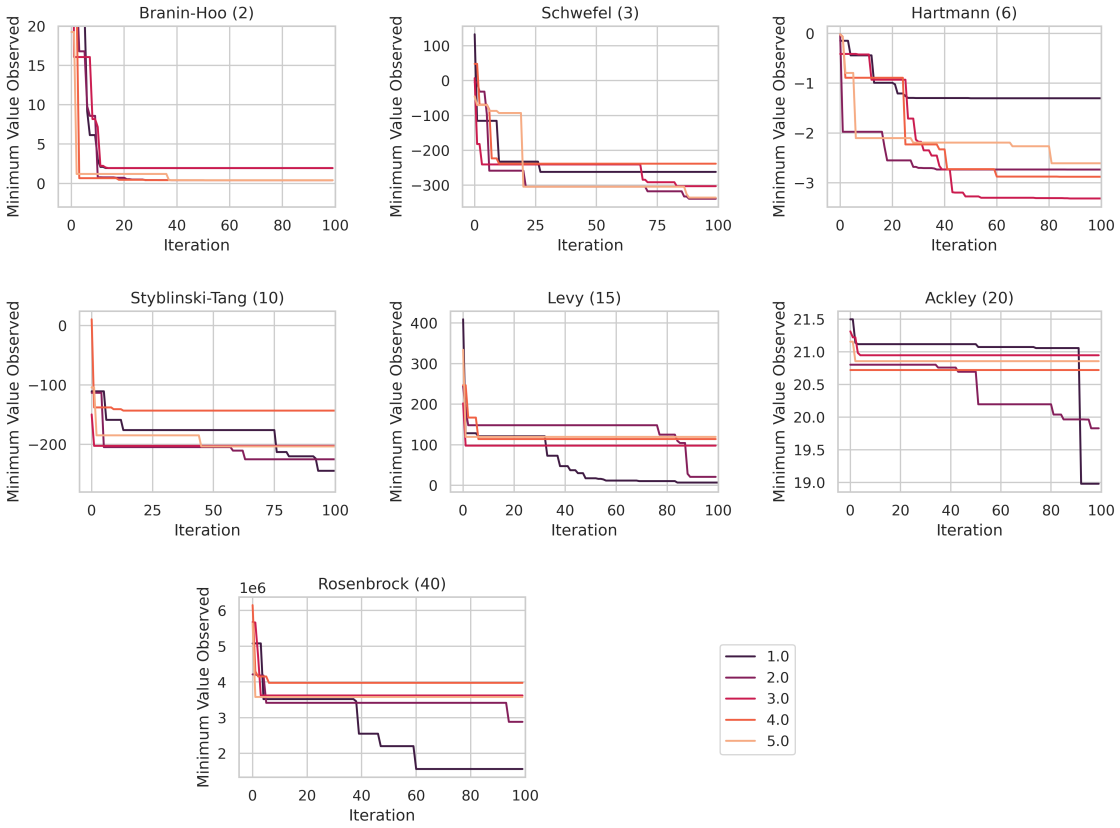


Figure 3.4: Hyper-parameter search plots for SimpleSMB: The plots show the minimum value observed over iterations for various initialization variances γ in SimpleSMB, for noiseless oracles. The dimensionality of the blackbox functions is provided in parenthesis. These runs were not averaged over multiple runs due to lack of computational resources.

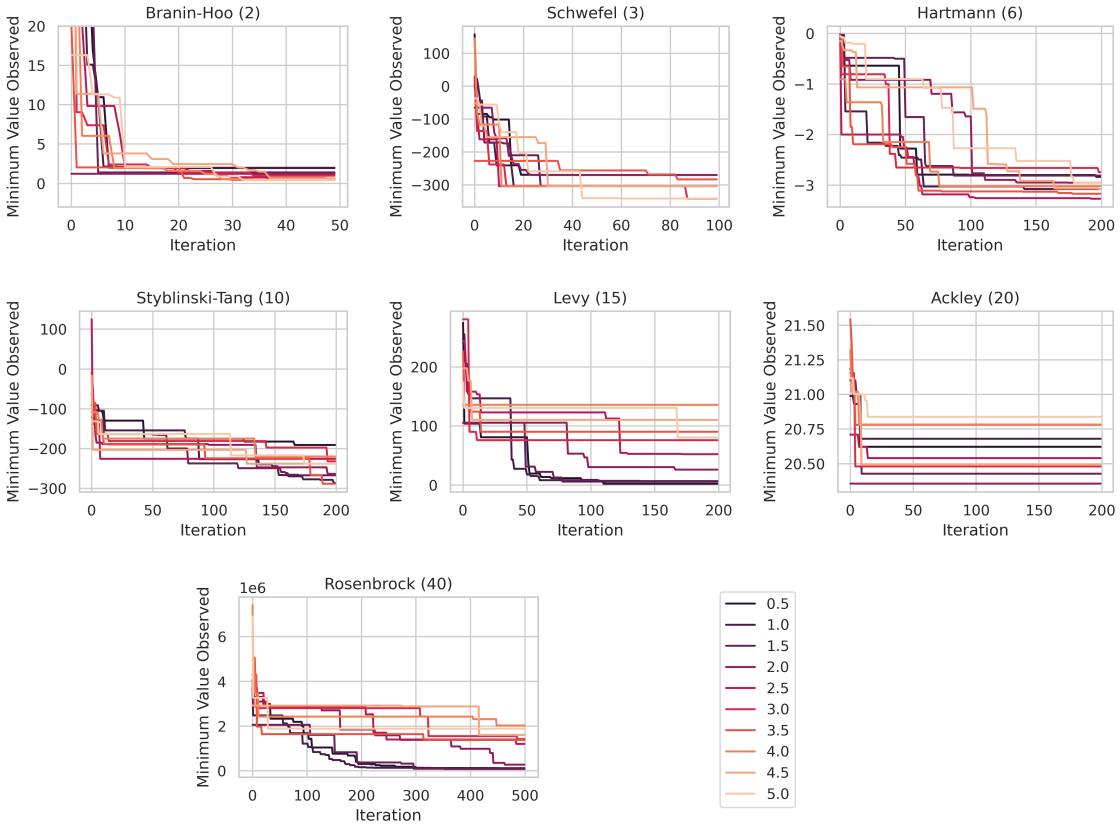
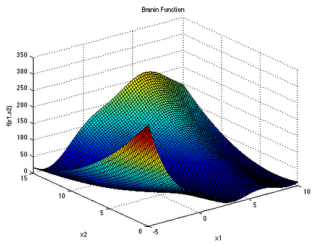
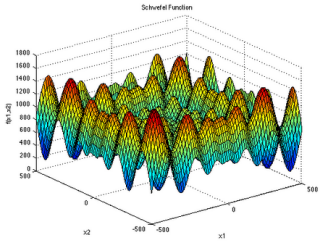


Figure 3.5: Noisy hyper-parameter search plots for SimpleSMB. The plots show the minimum *true* value observed over iterations for various initialization variances γ in SimpleSMB, for the noisy versions of the blackbox functions. The *true* value denotes the noise-free value of the black box function. The dimensionality of the blackbox functions is provided in parenthesis. These runs were not averaged over multiple runs due to lack of computational resources.



(a) Branin-Hoo (2)



(b) Schwefel (3)

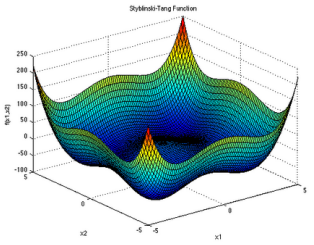
$$f(\mathbf{x}) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right), \text{ where}$$

$$\alpha = (1.0, 1.2, 3.0, 3.2)^T$$

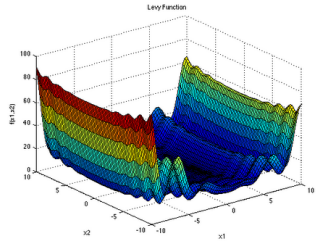
$$\mathbf{A} = \begin{pmatrix} 10 & 3 & 17 & 3.50 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}$$

$$\mathbf{P} = 10^{-4} \begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix}$$

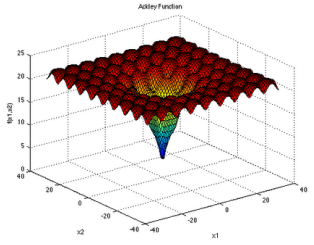
(c) Hartmann (6)



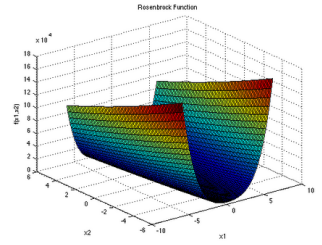
(d) Styblinski-Tang (10)



(e) Levy (15)



(f) Ackley (20)



(g) Rosenbrock (40)

Figure 3.6: The figures show 2-dimensional versions of the respective benchmark functions wherever available. Hartmann 6 does not have a 2-dimensional version, and hence the equation is displayed instead. All figures are credited to Surjanovic and Bingham [255].

4 | Cost-Aware BO via Information Directed Sampling

4.1 Introduction

Black-box optimization techniques such as Bayesian optimization are often based on *exploration* and *exploitation* approaches to identify the best proportion in a small number of experiments. Typically, BO approaches focus on achieving a small regret in a small number of experiments, while ignoring the cost of each individual experiment. In practice however, each experiment may utilize a different amount of resources. Furthermore, often cheap (or low fidelity) experiments are available that provide a significant amount of information about the location of the optimum. Multi-Fidelity [62, 115] and cost-aware [144] BO approaches address this by aiming to minimize the total experimental cost rather than the number of experiments. The key idea behind such approaches is to utilize low cost experiments for efficient *exploration* of the search space. We propose a method for cost-aware BO using information directed sampling (IDS) [229].

There are various notions of regret in the literature [30]. We aim to minimize the *simple regret* defined as the smallest regret among the individual regrets of all the sampled points. This notion of simple regret however does not fit well in the multi-fidelity setting. In the multi-fidelity we are often provided with low fidelity *approximations* of the true function. However, when measuring the simple regret, one must ignore the low fidelity evaluations, as they provide an inaccurate value of the function at that point, and hence cannot be used to make any practical decisions. Such a formulation for the simple regret in the multi-fidelity setting is also followed by Song et al. [249]. The main utility of the low-fidelity evaluations is exploration rather than exploitation.

The usual notion of simple regret holds in the cost-aware setting since there is no concept of fidelity and approximate evaluations. Hence all evaluations can be considered in the simple regret. This does not eliminate the need for cheap evaluations as they may still be used to gain information about the unknown function. In this chapter, we focus on the cost-aware setting with the goal of minimizing the simple regret given some cost budget.

As a concrete example, consider the battery optimization problem described earlier. One might have access to a simulator providing approximate evaluations. This is a multi-fidelity setting, as the approximate results from the simulator cannot be relied on for making decisions. One has to perform the highest fidelity evaluations eventually, in order to verify

the best found points. On the other hand, consider the setting where each experiment has a different cost – the cost-aware setting. This is a realistic setting since different configurations can utilize different amount of resources (battery chemicals in this example). All evaluations being exact in this scenario, they can be considered as reliable estimates of function values and hence included in the regret. While our proposed method can also be extended to the multi-fidelity setting, we focus on the cost-aware setting for simplicity of analysis.

Balancing *exploration* and *exploitation* is the main idea behind BO and bandit algorithms. In our setting, the cost must be factored in as well. At a high level, at each step, the algorithm should incur a small regret (exploitation), gain information (exploration) while using less resources (cost). Information Directed Sampling (IDS) [229] provides a principled strategy for balancing regret and information gain. In this work, we follow a similar approach and propose CostIDS, a cost-aware acquisition function that balances cost, regret, and information gain.

We propose a principled cost-aware acquisition function, which balances exploration, exploitation, and experimental cost. We show sub-linear regret bounds on the cumulative regret, thus resulting in zero simple regret as the budget is increased (no-regret property). Finally, we perform some preliminary experiments on a synthetic function and show promising results.

Compared to prior work, CostIDS enjoys a number of advantages. Our approach is a theoretically motivated no-regret algorithm, while being much simpler conceptually. Furthermore, our approach also does not require expensive entropy computations as PES [90], MES [278], and Multi-Fidelity MES [258]. Consequently, CostIDS is applicable to models beyond Gaussian processes where such entropy computations are prohibitive. Additionally, CostIDS does not require a pre-defined budget as assumed by Song et al. [249].

4.2 Related Work

Bayesian optimization of black-box functions is a well explored topic. A number of acquisition functions have been proposed in the literature including GP-UCB [251], Thompson sampling [230], expected improvement (EI) [112] and entropy based methods [90, 278]. Multi-objective BO has also been well explored. We refer to [200], for a discussion on multi-objective BO.

Experimental costs have been previously considered in BO in a variety of contexts. Kandasamy et al. [115], Song et al. [249] consider the multi-fidelity setting, where approximations of the true function are available as cheap low fidelity evaluations. On the other hand, Lee et al. [144], Takeno et al. [258] consider the cost-aware setting.

Information directed sampling (IDS) is a cumulative regret minimization approach that aims to balance regret incurred and information gained. IDS has been shown to perform well in cases where an action can provide information about other actions, also known as the complex information scenario [231]. IDS has been used for reinforcement learning [190], bandits with heteroscedastic noise [128], and linear partial monitoring [129].

4.3 Background

A background on Gaussian processes and Bayesian optimization can be found in Section 1.1.1.

Information Directed Sampling. Information Directed Sampling (IDS) is an information theoretic approach for cumulative regret minimization [229]. IDS uses the concept of information ratio which emerged in a related paper [231] on an information theoretic analysis of Thompson sampling. For a finite domain $\mathcal{X}$, a policy π is computed from which the next action $\boldsymbol{x}$ with will be sampled. IDS stipulates choosing the policy that minimizes the information ratio as defined below.

$$\pi_t = \operatorname*{argmin}_{\pi} \underbrace{\frac{\left(\mathbb{E}_{\boldsymbol{x}\sim\pi}\left[f(\boldsymbol{x}^*) - f(\boldsymbol{x}) \mid \mathcal{D}_{t-1}\right]\right)^2}{\mathbb{E}_{\boldsymbol{x}\sim\pi}\mathrm{IG}\left(\boldsymbol{x}^*, \boldsymbol{x} \mid \mathcal{D}_{t-1}\right)}}_{\text{Information Ratio}}, \tag{4.1}$$

where $\boldsymbol{x}^*$ denotes the optimal action, IG denotes the information gain, and $\mathcal{D}_t = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^t$ denotes the set of observations till step t. The next candidate is chosen as $\boldsymbol{x}_t \sim \pi_t$. The optimal policy π_t can potentially be a randomized policy, that is, π_t can be supported on more than one point in $\mathcal{X}$. At a high level, IDS minimizes the regret per information-gain. The key to bounding the regret of IDS is to bound the information ratio for the chosen policy π_t. Denote such an upper bound at step t by Γ_t. The regret at step t is upper bounded an increasing function of Γ_t. Further details and derivations can be found in Russo and Van Roy [229].

4.4 Cost-aware Information Directed Sampling

We define a modified version of the information ratio for Gaussian processes.

$$\boldsymbol{x}_t = \operatorname*{argmin}_{\boldsymbol{x}\in\mathcal{X}} \frac{\mathbb{E}\left[f^* - f(\boldsymbol{x}) \mid \mathcal{D}_{t-1}\right]^2}{\mathrm{IG}\left(f, \boldsymbol{x} \mid \mathcal{D}_{t-1}\right)} \tag{4.2}$$

This formulation differs from the original in quite a few aspects. The next candidate $\boldsymbol{x}_t$ is no longer randomized. While Russo and Van Roy [229] argue for randomized policies as necessary for certain problems, GPs are nice in that randomized policies are not necessary for optimization. GP-UCB [251] is an example of such a non-randomized policy. The other difference is that information gain on the maximizer $\mathrm{IG}\left(\boldsymbol{x}^*, \boldsymbol{x} \mid \mathcal{D}_{t-1}\right)$ is replaced by an upper bound $\mathrm{IG}\left(f, \boldsymbol{x} \mid \mathcal{D}_{t-1}\right)$, and consequently a smaller information ratio. The resulting regret bound is in terms of the maximum information gain of the function f rather than the maximizer $\boldsymbol{x}^*$.

The information ratio is dependent on the expected maximum of the posterior $\mathbb{E}[f^* \mid \mathcal{D}_{t-1}]$. We bound it using a discretization technique similar to Kandasamy et al. [117, Lemma 12], as summarized in the following lemma.

Proposition 4.1. *At any iteration t,*

$$\mathbb{E}[f^* \mid \mathcal{D}_{t-1}] \leq \frac{1}{t^2} + \underbrace{\max_{\boldsymbol{x}\in\mathcal{X}} \mu_t(\boldsymbol{x}) + \sqrt{\beta}\sigma_t(\boldsymbol{x})}_{\mathcal{U}_t\text{: max UCB at iteration } t}, \tag{4.3}$$

where $\beta_t = C_1 d \log t + C_2$, where C_1, C_2 are constants depending on the kernel κ of the GP.

Note that the upper bound is approximately the maximum UCB at iteration t, as $1/t^2$ tends to zero as $t \to \infty$. We denote the maximum UCB by $\mathcal{U}_t$ and the maximizer by $\boldsymbol{x}_t^{\text{UCB}}$. The remaining term $\mathbb{E}[f(\boldsymbol{x}) \mid \mathcal{D}_{t-1}]$ is a simply the posterior mean $\mu_t(\boldsymbol{x})$. Finally, for GPs the information gain can be expressed in closed form as $\text{IG}(f, \boldsymbol{x} \mid \mathcal{D}_{t-1}) = \frac{1}{2}\log(1+\sigma^{-2}\sigma_t(\boldsymbol{x})^2)$. However, in order to bound the information ratio we use $\sigma_t(\boldsymbol{x})^2$ instead, which is an increasing function of $\text{IG}(f, \boldsymbol{x} \mid \mathcal{D}_{t-1})$. As discussed earlier in Section 4.3, bounding the information ratio is essential to bound the regret.

CostIDS. With all the above substitutions and modifications, the information ratio, and the cost-aware information ratio are defined as,

$$\mathcal{R}_t(\boldsymbol{x}) = \frac{(\mathcal{U}_t - \mu_t(\boldsymbol{x}))^2}{\sigma_t(\boldsymbol{x})^2}, \qquad \mathcal{R}_t^{\text{cost}}(\boldsymbol{x}) = \lambda(\boldsymbol{x})\frac{(\mathcal{U}_t - \mu_t(\boldsymbol{x}))^2}{\sigma_t(\boldsymbol{x})^2}. \tag{4.4}$$

The cost $\lambda(\boldsymbol{x})$ is integrated as a multiplicative factor in $\mathcal{R}_t(\boldsymbol{x})$. The cost integrated acquisition function is then optimized to yield the next candidate. In practice, optimizing this directly can present some degenerate cases where extremely cheap points are chosen repeatedly which provide little information. To avoid such a pathology, we propose the following strategy.

$$\boldsymbol{x}_t = \operatorname*{argmin}_{\boldsymbol{x}\in\mathcal{X}} \mathcal{R}_t^{\text{cost}}(\boldsymbol{x}) \;\text{ s.t. }\; \mathcal{R}_t(\boldsymbol{x}) \le \rho\mathcal{R}_t^* \;\text{ where }\; \mathcal{R}_t^* = \min_{\boldsymbol{x}\in\mathcal{X}} \mathcal{R}_t(\boldsymbol{x}). \tag{4.5}$$

The constant $\rho > 1$ is a user-defined tolerance factor. The constraint that $\mathcal{R}_t(\boldsymbol{x})$ is not too far from the optimal $\mathcal{R}_t^*$ ensures that there is some progress in each iteration.

4.5 Regret Bounds

A key step in bounding the regret for IDS based algorithms is bounding the information ratio. We first assume the existence of upper bounds Γ_t and Γ_t^{cost} on $\mathcal{R}_t(\boldsymbol{x}_t)$ and $\mathcal{R}_t^{\text{cost}}(\boldsymbol{x}_t)$ respectively. Thereafter, we define the simple and cumulative regrets both wrt the step t and budget used Λ. Finally, we will also provide exact expressions for the upper bounds on the information ratio.

Assumption 4.1. *Assume that for all $t > 0$, the information ratios can be upper bounded as*

$$\mathcal{R}_t(\boldsymbol{x}_t) \le \Gamma_t, \quad \mathcal{R}_t^{\text{cost}}(\boldsymbol{x}_t) \le \Gamma_t^{\text{cost}} \quad \textit{almost surely,} \tag{4.6}$$

where Γ_t and Γ_t^{cost} are independent of the observations $\mathcal{D}_{t-1}$ and non-decreasing in t.

Definition 4.1 (Regrets wrt. to t)**.** *Define the instantaneous, cumulative, and simple regrets respectively, at step t as,*

$$r_t = f^* - f(\boldsymbol{x}_t), \quad R_t = \sum_{i=1}^{t} r_i, \quad s_t = \min_{i=1}^{t} r_t. \tag{4.7}$$

Next, we will propose a notion of the simple regret wrt. the total budget Λ.

Definition 4.2 (Regrets wrt. to Λ)**.** *Denote by Λ the total budget used. We define t_Λ as a random variable denoting largest time step such that the budget does exceed Λ.*

$$t_\Lambda = \max\left\{ t \;\middle|\; \sum_{i=1}^{t} \lambda(\boldsymbol{x}_i) \leq \Lambda \right\} \tag{4.8}$$

Define the simple and (cost-weighted) cumulative regret random variables for budget Λ as,

$$s_\Lambda = \min_{i=1}^{t_\Lambda} r_i, \quad R_\Lambda = \sum_{i=1}^{t_\Lambda} \lambda(\boldsymbol{x}_i) r_i \tag{4.9}$$

Theorem 4.1. *The cumulative and simple regrets wrt. the budget is upper bounded as,*

$$\mathbb{E}[R_\Lambda] \leq \sqrt{\Lambda C_1 \mathbb{E}[\Gamma^{\text{cost}}_{t_\Lambda} \gamma_{t_\Lambda}]} + C_2, \quad \mathbb{E}[s_\Lambda] \leq C_3 \sqrt{\mathbb{E}[\Gamma^{\text{cost}}_{t_\Lambda} \gamma_{t_\Lambda}]/\Lambda} + C_4, \tag{4.10}$$

for all $\Lambda > \lambda_{\max}$, where $C_1 = 2(1+\sigma^{-2})^{-1}$, $C_2 = \lambda_{\max}\pi^2/6$, $C_3 = \sqrt{C_1}(1-\lambda_{\max}/\Lambda)^{-1}$, $C_4 = \pi^2/6(\Lambda/\lambda_{\max}-1)^{-1}$, and γ_t denotes the maximum information gain.

Theorem 4.1 proof. We first show that $\mathbb{E}[s_\Lambda] \leq \frac{1}{\Lambda - \lambda_{\max}} \mathbb{E}[R_\Lambda]$.

$$\mathbb{E}[s_\Lambda] \leq \mathbb{E}\left[\frac{\sum_{i=1}^{t_\Lambda} \lambda(\boldsymbol{x}_i) r_i}{\sum_{i=1}^{t_\Lambda} \lambda(\boldsymbol{x}_i)} \right] \leq \mathbb{E}\left[\frac{R_\Lambda}{\Lambda - \lambda_{\max}} \right]$$

It remains to prove the upper bound on the cumulative regret.

$$\begin{aligned}
\mathbb{E}\left[R_\Lambda\right] &= \mathbb{E}\left[\sum_{i=1}^{t_\Lambda} \lambda(\boldsymbol{x}_i) r_i\right] = \mathbb{E}\left[\sum_{i=1}^{t_\Lambda} \lambda(\boldsymbol{x}_i)(f^* - f(\boldsymbol{x}_i))\right] \\
&\leq \mathbb{E}\left[\sum_{i=1}^{t_\Lambda} \lambda(\boldsymbol{x}_i)(\mathcal{U}_i - f(\boldsymbol{x}_i))\right] + \mathbb{E}\left[\sum_{i=1}^{t_\Lambda} \frac{\lambda_{\max}}{t^2}\right] \qquad \text{(by Proposition 4.1)} \\
&\leq \mathbb{E}\sqrt{\left(\sum_{i=1}^{t_\Lambda} \lambda(\boldsymbol{x}_i)\right)\left(\sum_{i=1}^{t_\Lambda} \lambda(\boldsymbol{x}_i)(\mathcal{U}_i - f(\boldsymbol{x}_i))^2\right)} + \lambda_{\max}\frac{\pi^2}{6} \text{ (by Cauchy Schwartz ineq.)} \\
&\leq \sqrt{\Lambda \mathbb{E}\left[\sum_{i=1}^{t_\Lambda} \Gamma^{\text{cost}}_i \sigma_i(\boldsymbol{x}_i)^2\right]} + \lambda_{\max}\frac{\pi^2}{6} \quad \text{(by concavity of the square root function)} \\
&\leq \sqrt{\Lambda C_1 \mathbb{E}\left[\Gamma^{\text{cost}}_{t_\Lambda} \gamma_{t_\Lambda}\right]} + \lambda_{\max}\frac{\pi^2}{6} \\
&\qquad \text{(follows from Srinivas et al. [251, Lemma 5.4 proof])}
\end{aligned}$$

□

The expectation in the last expression is with respect to t_Λ. Next we provide explicit expressions for the information ratio upper bounds, Γ^{cost}_t and Γ_t. Consider the information ratio for $\boldsymbol{x}^{\text{UCB}}_t$, the maximizer of the UCB. This leads to $\mathcal{R}^*_t \leq \mathcal{R}_t(\boldsymbol{x}^{\text{UCB}}_t) = \beta_t$. Therefore, $\Gamma_t = \beta_t$ is a valid upper bound. Consequently we have $\mathcal{R}^{\text{cost}}_t(\boldsymbol{x}_{t+1}) \leq \lambda_{\max}\rho\beta_t$, leading to $\Gamma^{\text{cost}}_t = \lambda_{\max}\rho\beta_t$. Substituting them above, we get a simple regret bound of $\mathbb{E}[s_\Lambda] \leq \mathcal{O}(\sqrt{\mathbb{E}[\beta_{t_\Lambda}\gamma_{t_\Lambda}]\lambda_{\max}/\Lambda})$.

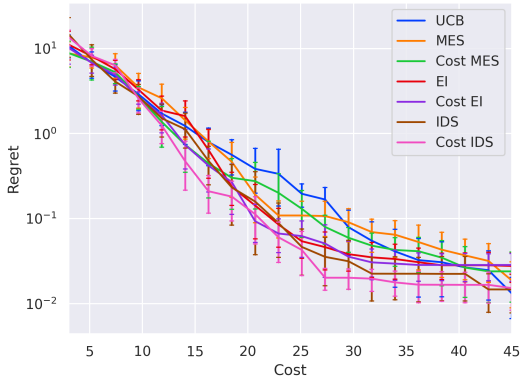


Figure 4.1: Cost vs. Simple regret plot for the modified Branin function.

4.6 Experiments

We perform experiments on the Branin (2-dim) function denoted by $b(x_1, x_2)$. We modify it to simulate a hyper-parameter optimization problem for a ML model by adding a 3rd dimension as $B(x_1, x_2, l)$. The modified function denotes the validation error of hypothetical ML model, l denotes the log of number of iterations the model is trained and x_1, x_2 denote the hyper-parameters of the model. We define the modified function as $B(x_1, x_2, l) = b(x_1, x_2) - l$ to model the decrease in error with more training iterations. Furthermore, the decrease in error diminishes with more training iterations, which is a standard phenomenon in practice. The training cost for the tuple (x_1, x_2, l) is $\exp(l)$. We compare our algorithm with MES, Multi-Fidelity MES (Cost MES), EI, Cost EI, GP-UCB, and standard IDS. Fig. 4.1 shows the simple regret vs. the cost incurred. We observe that CostIDS achieves a better cost vs. regret tradeoff.

Part II

Multi-Objective Optimization

5 | Minimizing FLOPs to Learn Efficient Sparse Representations

5.1 Introduction

Learning semantic representations using deep neural networks (DNN) is now a fundamental facet of applications ranging from visual search [78, 109], semantic text matching [184], oneshot classification [132], clustering [193], and recommendation [242]. The high-dimensional dense embeddings generated from DNNs however pose a computational challenge for performing nearest neighbor search in large-scale problems with millions of instances. In particular, when the embedding dimension is high, evaluating the distance of any query to all the instances in a large database is expensive, so that efficient search without sacrificing accuracy is difficult. Representations generated using DNNs typically have a higher dimension compared to hand-crafted features such as SIFT [159], and moreover are dense. The key caveat with dense features is that unlike bag-of-words features they cannot be efficiently searched through an inverted index, without approximations.

Since accurate search in high dimensions is prohibitively expensive in practice [276], one has to typically sacrifice accuracy for efficiency by resorting to approximate methods. Addressing the problem of efficient approximate *Nearest-Neighbor Search (NNS)* [107] or *Maximum Inner-Product Search (MIPS)* [244] is thus an active area of research, which we review in brief in the related work section. Most approaches [40, 107] aim to learn compact lower-dimensional representations that preserve distance information. Achieving the optimal trade-off (in a Pareto optimal sense) is often the goal in these approaches.

While there has been ample work on learning compact representations, learning sparse higher dimensional representations have been addressed only recently [37, 108]. As a seminal instance, Jeong and Song [108] propose an end-to-end approach to learn sparse and high-dimensional hashes, showing significant speed-up in retrieval time on benchmark datasets compared to dense embeddings. This approach has also been motivated from a biological viewpoint [151] by relating to a fruit fly's olfactory circuit, thus suggesting the possibility of hashing using higher dimensions instead of reducing the dimensionality. Furthermore, as suggested by Glorot et al. [72], sparsity can have additional advantages of linear separability and information disentanglement.

In a similar vein, in this work, we propose to learn high dimensional embeddings that are sparse and hence efficient to retrieve using sparse matrix multiplication operations. In

contrast to compact lower-dimensional ANN-esque representations that typically lead to decreased representational power, a key facet of our higher dimensional sparse embeddings is that they can have the same representational capacity as the initial dense embeddings. The core idea behind our approach is inspired by two key observations: (i) retrieval of d (high) dimensional sparse embeddings with fraction p of non-zero values on an average, can be sped up by a factor of $1/p$. (ii) The speed up can be further improved to a factor of $1/p^2$ by ensuring that the non-zero values are evenly distributed across all the dimensions. This indicates that sparsity alone is not sufficient to ensure maximal speedup; the distribution of the non-zero values plays a significant role as well. This motivates us to consider the effect of sparsity on the number of *floating point operations (FLOPs)* required for retrieval with an inverted index. We propose a penalty function on the embedding vectors that is a continuous relaxation of the exact number of FLOPs, and encourages an even distribution of the non-zeros across the dimensions. This results in a multi-objective problem where the objectives are retrieval accuracy and retrieval speed. However, as none of these quantities are differentiable, we work with their respective surrogates: the representation learning loss and the continuous relaxation of the FLOPs, respectively.

We apply our approach to the large scale metric learning problem of learning embeddings for facial images. Our training loss consists of a *metric learning* [279] loss aimed at learning embeddings that mimic a desired metric, and a *FLOPs loss* to minimize the number of operations. We perform an empirical evaluation of our approach on the Megaface dataset [122], and show that our proposed method successfully learns high-dimensional sparse embeddings that are orders-of-magnitude faster. We compare our approach to multiple baselines demonstrating an improved or similar speed-vs-accuracy trade-off.

The rest of the chapter is organized as follows. In Section 5.3 we analyze the expected number of FLOPs, for which we derive an exact expression. In Section 5.4 we derive a continuous relaxation that can be used as a regularizer, and optimized using gradient descent. We also provide some analytical justifications for our relaxation. In Section 5.5 we then compare our method on a large metric learning task showing an improved speed-accuracy trade-off compared to the baselines.

5.2 Related Work

Learning compact representations, ANN. Exact retrieval of the top-k nearest neighbours is expensive in practice for high-dimensional dense embeddings learned from deep neural networks, with practitioners often resorting to *approximate nearest neighbours* (ANN) for efficient retrieval. Popular approaches for ANN include *Locality sensitive hashing* (LSH) [5, 70, 210] relying on random projections, *Navigable small world graphs* (NSW) [163] and hierarchical NSW (HNSW) [164] based on constructing efficient search graphs by finding clusters in the data, *Product Quantization* (PQ) [65, 107] approaches which decompose the original space into a cartesian product of low-dimensional subspaces and quantize each of them separately, and *Spectral hashing* [280] which involves an NP hard problem of computing an optimal binary hash, which is relaxed to continuous valued hashes, admitting a simple solution in terms of the spectrum of the similarity matrix. Overall, for compact representations and to speed up query times, most of these approaches use a variety of carefully chosen data structures, such as hashes [186, 275], locality sensitive hashes [5], inverted

file structure [16, 107], trees [212], clustering [11], quantization sketches [107, 191], as well as dimensionality reductions based on principal component analysis and t-SNE [161].

End to end ANN. Learning the ANN structure end-to-end is another thread of work that has gained popularity recently. Norouzi et al. [192] propose to learn binary representations for the Hamming metric by minimizing a margin based triplet loss. Erin Liong et al. [59] use the signed output of a deep neural network as hashes, while imposing independence and orthogonality conditions on the hash bits. Other end-to-end learning approaches for learning hashes include [36, 148]. An advantage of end-to-end methods is that they learn hash codes that are optimally compatible to the feature representations.

Sparse representations. Sparse representations have been previously studied from various viewpoints. Glorot et al. [72] explore sparse neural networks in modeling biological neural networks and show improved performance, along with additional advantages such as better linear separability and information disentangling. Lee et al. [145], Ranzato et al. [215, 216] propose learning sparse features using deep belief networks. Olshausen and Field [194] explore sparse coding with an overcomplete basis, from a neurobiological viewpoint. Sparsity in auto-encoders have been explored by Kavukcuoglu et al. [121], Ng et al. [187]. Arpit et al. [8] provide sufficient conditions to learn sparse representations, and also further provide an excellent review of sparse autoencoders. Dropout [253] and a number of its variants [12, 177, 203] have also been shown to impose sparsity in neural networks.

High-dimensional sparse representations. *Sparse deep hashing (SDH)* [108] is an end-to-end approach that involves starting with a pre-trained network and then performing alternate minimization consisting of two minimization steps, one for training the binary hashes and the other for training the continuous dense embeddings. The first involves computing an optimal hash best compatible with the dense embedding using a *min-cost-max-flow* approach. The second step is a gradient descent step to learn a dense embedding by minimizing a metric learning loss. A related approach, k-sparse autoencoders [162] learn representations in an unsupervised manner with at most k non-zero activations. The idea of high dimensional sparse embeddings is also reinforced by the *sparse-lifting* approach [151] where sparse high dimensional embeddings are learned from dense features. The idea is motivated by the biologically inspired *fly* algorithm [48]. Experimental results indicated that *sparse-lifting* is an improvement both in terms of precision and speed, when compared to traditional techniques like LSH that rely on dimensionality reduction.

ℓ_1 regularization, lasso. The *Lasso* [261] is the most popular approach to impose sparsity and has been used in a variety of applications including sparsifying and compressing neural networks [153, 282]. The *group lasso* [172] is an extension of lasso that encourages all features in a specified group to be selected together. Another extension, the *exclusive lasso* [133, 300], on the other hand, is designed to select a single feature in a group. Our proposed regularizer, originally motivated by idea of minimizing FLOPs closely resembles exclusive lasso. Our focus however is on sparsifying the produced embeddings rather than sparsifying the parameters.

Sparse matrix vector product (SpMV). Existing work for SpMV computations include [79, 138], proposing algorithms based on inverted indices. Inverted indices are however known to suffer from severe cache misses. Linear algebra back-ends such as BLAS [21] rely on efficient cache accesses to achieve speedup. Haffner [79], Krotkiewski and Dabrowski [137], Mellor-Crummey and Garvin [173] propose cache efficient algorithms for sparse matrix vector products. There has also been substantial interest in speeding up SpMV computations using specialized hardware such as GPUs [269, 270], FPGAs [296, 301], and custom hardware [209].

Metric learning. While there exist many settings for learning embeddings [93, 123, 127], we restrict our attention to the context of metric learning [279]. Some examples of metric learning losses include large margin softmax loss for CNNs [155], triplet loss [238], and proxy based metric loss [180].

5.3 Expected number of FLOPs

In this section we study the effect of sparsity on the expected number of FLOPs required for retrieval and derive an exact expression for the expected number of FLOPs. Our main idea is based on the key insight that if each of the dimensions of the embedding are non-zero with a probability p (not necessarily independently), then it is possible to achieve a speedup up to an order of $1/p^2$ using an inverted index on the set of embeddings. Consider two embedding vectors $\boldsymbol{u}, \boldsymbol{v}$. Computing $\boldsymbol{u}^T\boldsymbol{v}$ requires computing only the pointwise product at the indices k where both $\boldsymbol{u}_k$ and $\boldsymbol{v}_k$ are non-zero. This is the main motivation behind using inverted indices and leads to the aforementioned speedup. Before we analyze it more formally, we introduce some notation.

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be a set of n independent training samples drawn from $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ according to a distribution $\mathcal{P}$, where $\mathcal{X}, \mathcal{Y}$ denote the input and label spaces respectively. Let $\mathscr{F} = \{f_\theta : \mathcal{X} \to \mathbb{R}^d \mid \theta \in \Theta\}$ be a class of functions parameterized by $\theta \in \Theta$, mapping input instances to d-dimensional embeddings. Typically, for image tasks, the function is chosen to be a suitable CNN [136]. Suppose $X, Y \sim \mathcal{P}$, then define the activation probability $p_j = \mathbb{P}(f_\theta(X)_j \neq 0)$, and its empirical version $\bar{p}_j = \frac{1}{n}\sum_{i=1}^n \mathbb{I}[f_\theta(\boldsymbol{x}_i)_j \neq 0]$.

We now show that sparse embeddings can lead to a quadratic speedup. Consider a d-dimensional sparse query vector $\boldsymbol{u}_q = f_\theta(\boldsymbol{x}_q) \in \mathbb{R}^d$ and a database of n sparse vectors $\{\boldsymbol{v}_i = f_\theta(\boldsymbol{x}^{(i)})\}_{i=1}^n \subset \mathbb{R}^d$ forming a matrix $\boldsymbol{D} \in \mathbb{R}^{n \times d}$. We assume that $\boldsymbol{x}_q, \boldsymbol{x}^{(i)}$ $(i = 1, \ldots, n)$ are sampled independently from $\mathcal{P}$. Computing the vector matrix product $\boldsymbol{D}\boldsymbol{u}_q$ requires looking at only the columns of $\boldsymbol{D}$ corresponding to the non-zero entries of $\boldsymbol{u}_q$ given by $N_q = \{j \mid 1 \leq j \leq d, (\boldsymbol{u}_q)_j \neq 0\}$. Furthermore, in each of those columns we only need to look at the non-zero entries. This can be implemented efficiently in practice by storing the non-zero indices for each column in independent lists, as depicted in Fig. 5.2.

The number of FLOPs incurred is given by,

$$F(\boldsymbol{D}, \boldsymbol{u}_q) = \sum_{j \in N_q} \sum_{i : \boldsymbol{v}_{ij} \neq 0} 1 = \sum_{i=1}^{n} \sum_{j=1}^{d} \mathbb{I}[(\boldsymbol{u}_q)_j \neq 0 \wedge \boldsymbol{v}_{ij} \neq 0]$$

Taking the expectation on both sides w.r.t. $\boldsymbol{x}_q, \boldsymbol{x}^{(i)}$ and using the independence of the data, we get

$$\mathbb{E}[F(\boldsymbol{D}, \boldsymbol{u}_q)] = \sum_{i=1}^{n} \sum_{j=1}^{d} \mathbb{P}\big((\boldsymbol{u}_q)_j \neq 0\big)\mathbb{P}\big(\boldsymbol{v}_{ij} \neq 0\big) = n \sum_{j=1}^{d} \mathbb{P}(f_\theta(X)_j \neq 0)^2 \tag{5.1}$$

where $X \sim \mathcal{P}$. Since the expected number of FLOPs scales linearly with the number of vectors in the database, a more suitable quantity is the *mean-FLOPs-per-row* defined as

$$\mathcal{F}(f_\theta, \mathcal{P}) = \mathbb{E}[F(\boldsymbol{D}, \boldsymbol{u}_q)]/n = \sum_{j=1}^{d} \mathbb{P}(f_\theta(X)_j \neq 0)^2 = \sum_{j=1}^{d} p_j^2. \tag{5.2}$$

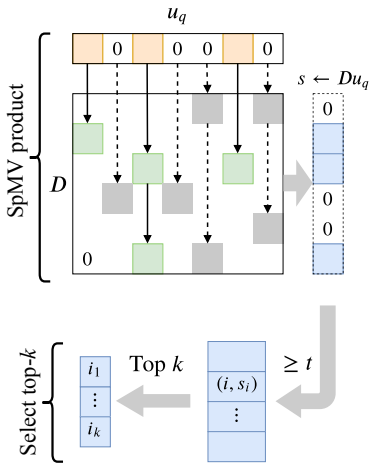
Note that for a fixed amount of sparsity $\sum_{j=1}^{d} p_j = d\,p$, this is minimized when each of the dimensions are non-zero with equal probability $p_j = p$, $\forall 1 \leq j \leq d$, upon which $\mathcal{F}(f_\theta, \mathcal{P}) = d\,p^2$ (so that as a regularizer, $\mathcal{F}(f_\theta, \mathcal{P})$ will in turn encourage such a uniform distribution across dimensions). Given such a uniform distribution, compared to dense multiplication which has a complexity of $O(d)$ per row, we thus get an improvement by a factor of $1/p^2$ ($p < 1$). Thus when only p fraction of all the entries is non-zero, and evenly distributed across all the columns, we achieve a speedup of $1/p^2$. Note that independence of the non-zero indices is not necessary due to the linearity of expectation – in fact, features from a neural network are rarely uncorrelated in practice.

FLOPs versus speedup. While FLOPs reduction is a reasonable measure of speedup on primitive processors of limited parallelization and cache memory. FLOPs is not an accurate measure of actual speedup when it comes to mainstream commercial processors such as Intel's CPUs and Nvidia's GPUs, as the latter have cache and SIMD (Single-Instruction Multiple Data) mechanism highly optimized for dense matrix multiplication, while sparse matrix multiplication are inherently less tailored to their cache and SIMD design [248]. On the other hand, there have been threads of research on hardwares with cache and parallelization tailored to sparse operations that show speedup proportional to the FLOPs reduction [81, 199]. Modeling the cache and other hardware aspects can potentially lead to better performance but less generality and is left to future work.

5.4 Our Approach

The ℓ_1 regularization is the most common approach to induce sparsity. However, as we will also verify experimentally, it does not ensure an uniform distribution of the non-zeros in all the dimensions that is required for the optimal speed-up. Therefore, we resort to incorporating the actual FLOPs incurred, directly into the loss function which will lead to an optimal trade-off between the search time and accuracy. The FLOPs $\mathcal{F}(f_\theta, \mathcal{P})$ being a discontinuous function of model parameters, is hard to optimize, and hence we will instead optimize using a continuous relaxation of it.

Denote by $\ell(f_\theta, \mathcal{D})$, any metric loss on $\mathcal{D}$ for the embedding function f_θ. Our goal is to minimize the loss while controlling the expected FLOPs $\mathcal{F}(f_\theta, \mathcal{P})$ defined in Eq. (5.2). Since the distribution $\mathcal{P}$ is unknown, we use the samples to get an estimate of $\mathcal{F}(f_\theta, \mathcal{P})$. Recall the empirical fraction of non-zero activations $\bar{p}_j = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[f_\theta(\boldsymbol{x}_i)_j \neq 0]$, which converges



Algorithm 5.1 Sparse Nearest Neighbour

(Build Index)
Input: Sparse matrix $\boldsymbol{D}$
  for $j = 1 \cdots d$ **do**
    Init $C[j] \leftarrow \{(i, \boldsymbol{D}_{ij}) \mid \boldsymbol{D}_{ij} \neq 0 \ \wedge \ 1 \leq i \leq n\}$
      ▷ Stores the non-zero values and their indices

(Query)
Input: Sparse query $\boldsymbol{u}_q$, threshold t, number of NNs k
  Init score vector $s[i] = 0, \ 1 \leq i \leq n$.
  for $j = 1 \cdots d$ s.t. $\boldsymbol{u}_q[j] \neq 0$ **do** ▷ SpMV product
    for $(i, v) \in C[j]$ **do**
      $s[i] \mathrel{+}= v\boldsymbol{u}_q[j]$
  $S \leftarrow \{(i, s[i]) \mid 1 \leq i \leq n, \ s[i] \geq t\}$ ▷ Thresholding
  $S_k \leftarrow \{i_1, \ldots, i_k\}$ top-k indices i of S based on $s[i]$
    ▷ Using `nth_select` from C++ STL
  return S_k

Figure 5.2: **SpMV product:** The colored cells denote non-zero entries, and the arrows indicate the list structure for each of the columns, with solid arrows denoting links that were traversed for the given query. The green and grey cells denote the non-zero entries that were accessed and not accessed, respectively. The non-zero values in $\boldsymbol{D}\boldsymbol{u}_q$ (blue) can be computed using only the common non-zero values (green). **Selecting top-k:** The sparse product vector is then filtered using a threshold t, after which the top-k indices are returned.

in probability to p_j. Therefore, with a slight abuse of notation define $\mathcal{F}(f_\theta, \mathcal{D}) = \sum_{j=1}^d \bar{p}_j^2$, which is a consistent estimator for $\mathcal{F}(f_\theta, \mathcal{P})$ based on the samples $\mathcal{D}$. Note that $\mathcal{F}$ denotes either the population or empirical quantities depending on whether the functional argument is $\mathcal{P}$ or $\mathcal{D}$. We now consider the following regularized loss.

$$\min_{\theta \in \Theta} \underbrace{\ell(f_\theta, \mathcal{D}) + \lambda \mathcal{F}(f_\theta, \mathcal{D})}_{\mathcal{L}(\theta)} \tag{5.3}$$

for some parameter λ that controls the FLOPs-accuracy tradeoff. The regularized loss poses a further hurdle, as $\bar{p}_j$ and consequently $\mathcal{F}(f_\theta, \mathcal{D})$ are not continuous due the presence of the indicator functions. We thus compute the following continuous relaxation. Define the mean absolute activation $a_j = \mathbb{E}[|f_\theta(X)_j|]$ and its empirical version $\bar{a}_j = \frac{1}{n} \sum_{i=1}^n |f_\theta(\boldsymbol{x}_i)_j|$, which is the ℓ_1 norm of the activations (scaled by $1/n$) in contrast to the ℓ_0 quasi norm in the FLOPs calculation. Define the relaxations, $\widetilde{\mathcal{F}}(f_\theta, \mathcal{P}) = \sum_{j=1}^d a_j^2$ and its consistent estimator $\widetilde{\mathcal{F}}(f_\theta, \mathcal{D}) = \sum_{j=1}^d \bar{a}_j^2$. We propose to minimize the following relaxation, which can be optimized using any off-the-shelf stochastic gradient descent optimizer. The resulting loss is essentially a linear scalarization of our two surrogate objectives: the representation learning loss and the continuous relaxation of the FLOPs.

$$\min_{\theta \in \Theta} \underbrace{\ell(f_\theta, \mathcal{D}) + \lambda \widetilde{\mathcal{F}}(f_\theta, \mathcal{D})}_{\widetilde{\mathcal{L}}(\theta)}. \tag{5.4}$$

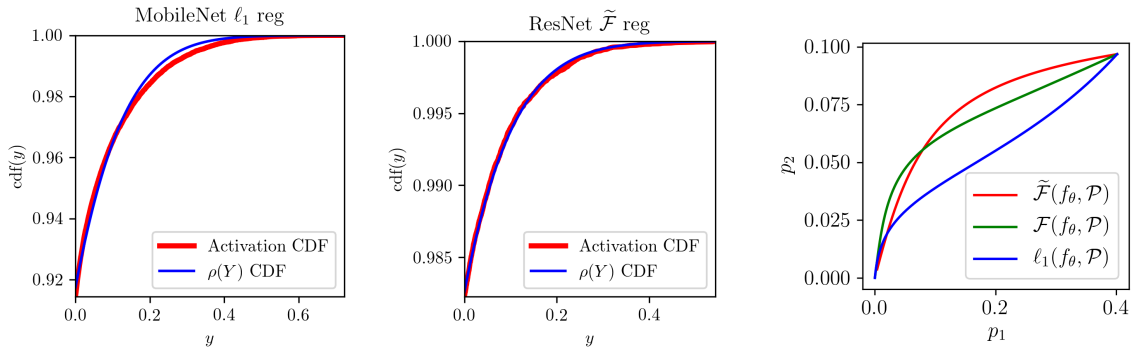
Sparse retrieval and re-ranking. During inference, the sparse vector of a query image is first obtained from the learned model and the nearest neighbour is searched in a database of sparse vectors forming a sparse matrix. An efficient algorithm to compute the dot product of the sparse query vector with the sparse matrix is presented in Algorithm 5.1. This consists of first building a list of the non-zero values and their positions in each column. As motivated in Section 5.3, given a sparse query vector, it is sufficient to only iterate through the non-zero values and the corresponding columns. Next, a filtering step is performed keeping only scores greater than a specified threshold. Top-k candidates from the remaining items are returned. The complete algorithm is presented in Algorithm 5.1. In practice, the sparse retrieval step is not sufficient to ensure good performance. The top-k shortlisted candidates are therefore further re-ranked using dense embeddings as done in SDH. This step involves multiplication of a small dense matrix with a dense vector. The number of shortlisted candidates k is chosen such that the dense re-ranking time does not dominate the total time.

Comparison to SDH [108]. It is instructive to contrast our approach with that of SDH [108]. In contrast to the binary hashes in SDH, our approach learns sparse real valued representations. SDH uses a min-cost-max-flow approach in one of the training steps, while we train ours only using SGD. During inference in SDH, a shortlist of candidates is first created by considering the examples in the database that have hashes with non-empty intersections with the query hash. The candidates are further re-ranked using the dense embeddings. The shortlist in our approach on the other hand is constituted of the examples with the top scores from the sparse embeddings.

Comparison to unrelaxed FLOPs regularizer. We provide an experimental comparison of our continuous relaxation based FLOPs regularizer to its unrelaxed variant, showing that the performance of the two are markedly similar. Setting up this experiment requires some analytical simplifications based on recent deep neural network analyses. We first recall recent results that indicate that the output of a batch norm layer nearly follows a Gaussian distribution [236], so that in our context, we could make the simplifying approximation that $f_\theta(X)_j$ (where $X \sim \mathcal{P}$) is distributed as $\rho(Y)$ where $Y \sim \mathcal{N}(\mu_j(\theta), \sigma_j^2(\theta))$, ρ is the ReLU activation used at the neural network output. We have modelled the pre-activation as a Gaussian distribution with mean and variance depending on the model parameters θ. We experimentally verify that this assumption holds by minimizing the KS distance [166] between the CDF of $\rho(Y)$ where $Y \sim \mathcal{N}(\mu, \sigma^2)$ and the empirical CDF of the activations. The KS distance is minimized wrt. μ, σ. Fig. 5.3a shows the empirical CDF and the fitted CDF of $\rho(Y)$ for two different architectures.

While $\mu_j(\theta), \sigma_j(\theta)$ $(1 \leq j \leq d)$ cannot be tuned independently due to their dependence on θ, in practice, the huge representational capacity of neural networks allows $\mu_j(\theta)$ and $\sigma_j(\theta)$ to be tuned almost independently. We consider a toy setting with 2-d embeddings. For a tractable analysis, we make the simplifying assumption that, for $j = 1, 2$, $f_\theta(X)_j$ is distributed as $\text{ReLU}(Y)$ where $Y \sim \mathcal{N}(\mu_j, \sigma_j^2)$, thus losing the dependence on θ.

We now analyze how minimizing the continuous relaxation $\widetilde{\mathcal{F}}(f_\theta, \mathcal{P})$ compares to minimizing $\mathcal{F}(f_\theta, \mathcal{P})$. Note that we consider the population quantities here instead of the empirical quantities, as they are more amenable to theoretical analyses due to the existence of closed



(a) The CDF of $\rho(Y)$ fitted to minimize the KS distance to the empirical CDF of the activations for two different architectures.

(b) The trajectory of the activation probabilities when minimizing the respective regularizers.

Figure 5.3: Figure (a) shows that the CDF of the activations (red) closely resembles the CDF of $\rho(Y)$ (blue) where Y is a Gaussian random variable. Figure (b) shows that $\mathcal{F}$ and $\widetilde{\mathcal{F}}$ behave similarly by sparsifying the less sparser activation at a faster rate when compared to the ℓ_1 regularizer.

form expressions. We also consider the ℓ_1 regularizer as a baseline. We initialize with $(\mu_1, \mu_2, \sigma_1, \sigma_2) = (-1/4, -1.3, 1, 1)$, and minimize the three quantities via gradient descent with infinitesimally small learning rates. For this contrastive analysis, we have not considered the effect of the metric loss. Note that while the discontinuous empirical quantity $\mathcal{F}(f_\theta, \mathcal{D})$ cannot be optimized via gradient descent, it is possible to do so for its population counterpart $\mathcal{F}(f_\theta, \mathcal{P})$ since it is available in closed form as a continuous function when making Gaussian assumptions. The details of computing the gradients can be found in Section 5.6.

We start with activation probabilities $(p_1, p_2) = (0.4, 0.1)$, and plot the trajectory taken when performing gradient descent, shown in Fig. 5.3b. Without the effect of the metric loss, the probabilities are expected to go to zero as observed in the plot. It can be seen that, in contrast to the ℓ_1-regularizer, $\mathcal{F}$ and $\widetilde{\mathcal{F}}$ both tend to sparsify the less sparse activation (p_1) at a faster rate, which *corroborates the fact that they encourage an even distribution of non-zeros.*

$\widetilde{\mathcal{F}}$ promotes orthogonality. We next show that, when the embeddings are normalized to have a unit norm, as typically done in metric learning, then minimizing $\widetilde{\mathcal{F}}(f_\theta, \mathcal{D})$ is equivalent to promoting orthogonality on the *absolute values* of the embedding vectors. Let $\|f_\theta(\boldsymbol{x})\|_2 = 1, \ \forall x \in \mathcal{X}$, we then have the following:

$$\widetilde{\mathcal{F}}(f_\theta, \mathcal{D}) = \sum_{j=1}^{d} \left(\frac{1}{n} \sum_{i=1}^{n} |f_\theta(\boldsymbol{x}_i)_j| \right)^2 = \frac{1}{n^2} \sum_{p,q \in [1:n]} \left\langle |f_\theta(\boldsymbol{x}_p)|, |f_\theta(\boldsymbol{x}_q)| \right\rangle \tag{5.5}$$

$\widetilde{\mathcal{F}}(f_\theta, \mathcal{D})$ is minimized when the vectors $\{|f_\theta(\boldsymbol{x}_i)|\}_{i=1}^n$ are orthogonal. Metric learning losses aim at minimizing the interclass dot product, whereas the FLOPs regularizer aims at minimizing pairwise dot products irrespective of the class, leading to a tradeoff between sparsity

and accuracy. This approach of pushing the embeddings apart, bears some resemblance to the idea of *spreading vectors* [233] where an entropy based regularizer is used to uniformly distribute the embeddings on the unit sphere, albeit without considering any sparsity. Maximizing the pairwise dot product helps in reducing FLOPs as is illustrated by the following toy example. Consider a set of d vectors $\{\boldsymbol{v}_i\}_{i=1}^{d} \subset \mathbb{R}^d$ (here $n = d$) satisfying $\|\boldsymbol{v}_i\|_2 = 1,\ \forall i \in [1:d]$. Then $\sum_{p,q\in[1:d]} \langle |\boldsymbol{v}_p|, |\boldsymbol{v}_q| \rangle$ is minimized when $\boldsymbol{v}_p = e_p$, where e_p is an one-hot vector with the p th entry equal to 1 and the rest 0. The FLOPs regularizer thus tends to spread out the non-zero activations in all the dimensions, thus producing balanced embeddings. This simple example also demonstrates that when the number of classes in the training set is smaller or equal to the number of dimensions d, a trivial embedding that minimizes the metric loss and also achieves a small number of FLOPs is $f_\theta(\boldsymbol{x}) = e_y$ where y is true label for $\boldsymbol{x}$. This is equivalent to predicting the class of the input instance. The caveat with such embeddings is that they might not be semantically meaningful beyond the specific supervised task, and will naturally hurt performance on unseen classes, and tasks where the representation itself is of interest. In order to avoid such a collapse in our experiments, we ensure that the embedding dimension is smaller than the number of training classes. Furthermore, as recommended by Sablayrolles et al. [232], we perform all our evaluations on unseen classes.

Exclusive lasso. Also known as $\ell_{1,2}$-norm, in previous works it has been used to induce competition (or exclusiveness) in features in the same group. More formally, consider d features indexed by $\{1, \ldots, d\}$, and groups $g \subset \{1, \ldots, d\}$ forming a set of groups $\mathcal{G} \subset 2^{\{1,\ldots,d\}}$.[1] Let $\boldsymbol{w}$ denote the weight vector for a linear classifier. The exclusive lasso regularizer is defined as,

$$\Omega_{\mathcal{G}}(\boldsymbol{w}) = \sum_{g\in\mathcal{G}} \|\boldsymbol{w}_g\|_1^2,$$

where $\boldsymbol{w}_g$ denotes the sub-vector $(\boldsymbol{w}_i)_{i\in g}$, corresponding to the indices in g. $\mathcal{G}$ can be used to induce various kinds of structural properties. For instance $\mathcal{G}$ can consist of groups of correlated features. The regularizer prevents feature redundancy by selecting only a few features from each group.

Our proposed FLOPs based regularizer has the same form as exclusive lasso. Therefore exclusive lasso applied to the batch of activations, with the groups being columns of the activation matrix (and rows corresponding to different inputs), is equivalent to the FLOPs regularizer. It can be said that, within each activation column, the FLOPs regularizer induces competition between different input examples for having a non-zero activation.

5.5 Experiments

We evaluate our proposed approach on a large scale metric learning dataset: the Megaface [122] used for face recognition. This is a much more fine grained retrieval tasks (with 85k classes for training) compared to the datasets used by Jeong and Song [108]. This dataset also satisfies our requirement of the number of classes being orders of magnitude higher than the dimensions of the sparse embedding. As discussed in Section 5.4, a few number of classes during training can lead the model to simply learn an encoding of the training classes

[1] Denotes the powerset of $\{1, \ldots, d\}$.

and thus not generalize to unseen classes. Face recognition datasets avoid this situation by virtue of the huge number of training classes and a balanced distribution of examples across all the classes.

Following standard protocol for evaluation on the Megaface dataset [122], we train on a refined version of the MSCeleb-1M [77] dataset released by Deng et al. [53] consisting of 1 million images spanning 85k classes. We evaluate with 1 million distractors from the Megaface dataset and 3.5k query images from the Facescrub dataset [188], which were not seen during training. In our experiments, we vary the hyper-parameters of all our compared methods, so as to explore the whole Pareto front. In particular, for our proposed method, we vary the weight of the FLOPs regularizer λ.

Network architecture. We experiment with two architectures: MobileFaceNet [43], and ResNet-101 [88]. We use ReLU activations in the embedding layer for MobileFaceNet, and SThresh activations (defined below) for ResNet. The activations are ℓ_2-normalized to produce an embedding on the unit sphere, and used to compute the Arcface loss [53]. We learn 1024 dimensional sparse embeddings for the ℓ_1 and $\widetilde{\mathcal{F}}$ regularizers; and 128, 512 dimensional dense embeddings as baselines. All models were implemented in Tensorflow [1] with the sparse retrieval algorithm implemented in C++. The re-ranking step used 512-d dense embeddings.

Activation function. In practice, having a non-linear activation at the embedding layer is crucial for sparsification. Layers with activations such as ReLU are easier to sparsify due to the bias parameter in the layer before the activation (linear or batch norm) which acts as a direct control knob to the sparsity. More specifically, $\text{ReLU}(\boldsymbol{x} - \boldsymbol{\lambda})$ can be made more (less) sparse by increasing (decreasing) the components of $\boldsymbol{\lambda}$, where $\boldsymbol{\lambda}$ is the bias parameter of the previous linear layer. We consider two types of activations: $\text{ReLU}(\boldsymbol{x}) = \max(\boldsymbol{x}, \boldsymbol{0})$, and the soft thresholding operator $\text{SThresh}(\boldsymbol{x}) = \text{sgn}(\boldsymbol{x}) \max(|\boldsymbol{x}| - 1/2, 0)$ [25]. ReLU activations always produce positive values, whereas soft thresholding can produce negative values as well.

Practical considerations. In practice, setting a large regularization weight λ from the beginning is harmful for training. Sparsifying too quickly using a large λ leads to many dead activations (saturated to zero) in the embedding layer and the model getting stuck in a local minima. Therefore, we use an annealing procedure and gradually increase λ throughout the training using a regularization weight schedule $\lambda(t) : \mathbb{N} \mapsto \mathbb{R}$ that maps the training step to a real valued regularization weight. In our experiments we choose a $\lambda(t)$ that increases quadratically as $\lambda(t) = \lambda(t/T)^2$, until step $t = T$, where T is the threshold step beyond which $\lambda(t) = \lambda$.

Baselines. We compare our proposed $\widetilde{\mathcal{F}}$-regularizer, with multiple baselines: exhaustive search with dense embeddings, sparse embeddings using ℓ_1 regularization, *Sparse Deep Hashing (SDH)* [108], and PCA, LSH, PQ applied to the 512 dimensional dense embeddings from both the architectures. We train the SDH model using the aforementioned architectures for 512 dimensional embeddings, with number of active hash bits $k = 3$. We use numpy (using efficient MKL optimizations in the backend) for matrix multiplication

required for exhaustive search in the dense and PCA baselines. We use the CPU version of the *Faiss* [110] library for LSH and PQ (we use the IVF-PQ index from Faiss).

Further details on the training hyperparameters and the hardware used can be found in Section 5.7.

5.5.1 Results

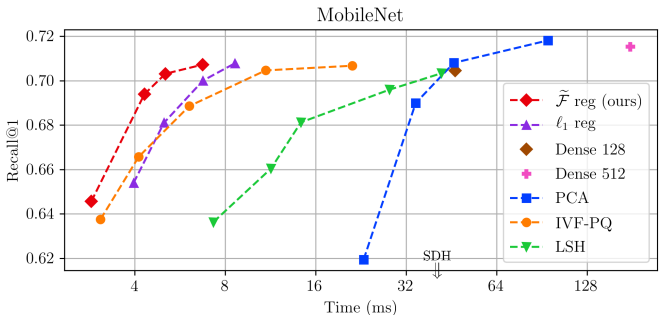
We report the recall and the time-per-query for various hyperparameters of our proposed approach and the baselines, yielding trade-off curves. The reported times include the time required for re-ranking. The trade-off curves for MobileNet and ResNet are shown in Figs. 5.4a and 5.4c respectively. We observe that while vanilla ℓ_1 regularization is an improvement by itself for some hyperparameter settings, the $\widetilde{\mathcal{F}}$ regularizer is a further improvement, and yields the most optimal trade-off curve. SDH has a very poor speed-accuracy trade-off, which is mainly due to the explosion in the number of shortlisted candidates with increasing number of active bits leading to an increase in the retrieval time. On the other hand, while having a small number of active bits is faster, it leads to a smaller recall. For the other baselines we notice the usual order of performance, with PQ having the best speed-up compared to LSH and PCA. While dimensionality reduction using PCA leads to some speed-up for relatively high dimensions, it quickly wanes off as the dimension is reduced even further.

We also report the sub-optimality ratio $R_{sub} = \mathcal{F}(f_\theta, \mathcal{D})/d\bar{p}^2$ computed over the dataset $\mathcal{D}$, where $\bar{p} = \frac{1}{d}\sum_{j=1}^{d} \bar{p}_j$ is the mean activation probability estimated on the test data. Notice that $R_{sub} \geq 1$, and the optimal $R_{sub} = 1$ is achieved when $\bar{p}_j = \bar{p},\ \forall 1 \leq j \leq d$, that is when the non-zeros are evenly distributed across the dimensions. The sparsity-vs-suboptimality plots for MobileNet and ResNet are shown in Figs. 5.4a and 5.4c respectively. We notice that the $\widetilde{\mathcal{F}}$-regularizer yields values of R_{sub} closer to 1 when compared to the ℓ_1-regularizer. For the MobileNet architecture we notice that the ℓ_1 regularizer is able to achieve values of R close to that of $\widetilde{\mathcal{F}}$ in the less sparser region. However, the gap increases substantially with increasing sparsity. For the ResNet architecture on the other hand the ℓ_1 regularizer yields extremely sub-optimal embeddings in all regimes. The $\widetilde{\mathcal{F}}$ regularizer is therefore able to produce more balanced distribution of non-zeros.

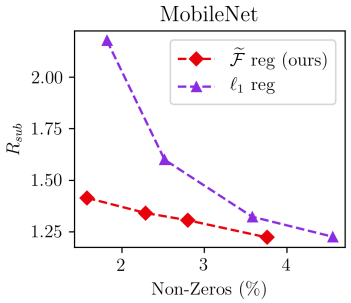
The sub-optimality is also reflected in the recall values. The gap in the recall values of the ℓ_1 and $\widetilde{\mathcal{F}}$ models is much higher when the sub-optimality gap is higher, as in the case of ResNet, while it is small when the sub-optimality gap is smaller as in the case of MobileNet. This shows the significance of having a balanced distribution of non-zeros. Additional results, including results without the re-ranking step and performance on CIFAR-100 can be found in Section 5.8.

5.6 Gradient computations for analytical experiments

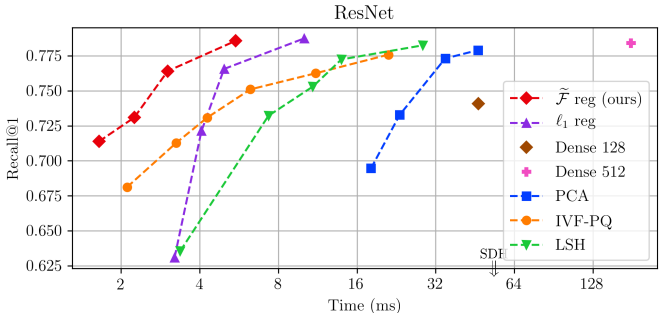
As described in the main text, for purposes of an analytical toy experiment, we consider a simplified setting with 2-d embeddings with the jth ($j = 1, 2$) activation being distributed as $(Y_j)_+ = \text{ReLU}(Y_j)$ where $Y_j \sim \mathcal{N}(\mu_j, \sigma_j)$. We assume $\mu_j \leq 0$, which is typical for sparse activations ($p_j \leq 0.5$). Then the three compared regularizers are $\mathcal{F}(p_\theta, \mathcal{P}) = \sum_{j=1}^{2} \mathbb{P}((Y_j)_+ > 0)^2$, $\widetilde{\mathcal{F}}(p_\theta, \mathcal{P}) = \sum_{j=1}^{2} \mathbb{E}[(Y_j)_+]^2$, and $\ell_1(p_\theta, \mathcal{P}) = \sum_{j=1}^{2} \mathbb{E}[(Y_j)_+]$. Computing the regu-



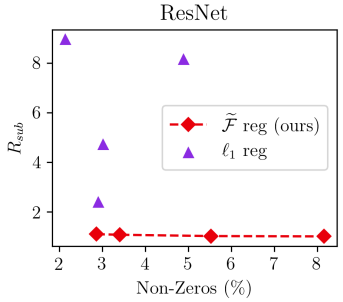
(a) Time per query vs recall for MobileNet.



(b) R_{sub} vs sparsity for MobileNet.



(c) Time per query vs recall for ResNet.



(d) R_{sub} vs sparsity for ResNet.

Figure 5.4: Figures (a) and (c) show the speed vs recall trade-off for the MobileNet and ResNet architectures respectively. The trade-off curves produced by varying the hyper-parameters of the respective approaches. The points with higher recall and lower time (top-left side of the plots) are better. The SDH baseline being out of range of both the plots is indicated using an arrow. Figures (b) and (d) show the sub-optimality ratio vs sparsity plots for MobileNet and ResNet respectively. R_{sub} closer to 1 indicates that the non-zeros are uniformly distributed across the dimensions.

larizer gradients thus boils down to computing the gradients of $\mathbb{P}((Y_j)_+ > 0)^2, \mathbb{E}[(Y_j)_+]^2$, and $\mathbb{E}[(Y_j)_+]$ as provided in the following lemmas. We hide the subscript j for brevity, as computations are similar for all j.

Lemma 5.1.

$$\mathbb{E}[Y_+] = \frac{\sigma}{\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right) + \mu\left(1 - \Phi\left(-\frac{\mu}{\sigma}\right)\right), \tag{5.6}$$

and,

$$\mathbb{P}(Y_+ > 0) = 1 - \Phi\left(-\frac{\mu}{\sigma}\right), \tag{5.7}$$

where Φ denotes the cdf of the Gaussian distribution.

Proof of Lemma 5.1. The proof is based on standard Gaussian identities.

$$\begin{aligned}\mathbb{E}[Y_+] &= \int_0^\infty \frac{x}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \int_{-\mu}^\infty \frac{x+\mu}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \\ &= \int_{-\mu}^\infty \frac{x}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx + \int_{-\mu}^\infty \frac{\mu}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \\ &= \frac{\sigma}{\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right) + \mu\left(1 - \Phi\left(-\frac{\mu}{\sigma}\right)\right)\end{aligned}$$

$$\begin{aligned}\mathbb{P}(Y_j > 0) &= \int_0^\infty \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \int_{-\mu/\sigma}^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx \\ &= 1 - \Phi\left(-\frac{\mu}{\sigma}\right)\end{aligned}$$

□

Lemma 5.2.

$$\frac{\partial \mathbb{P}(Y_+ > 0)}{\partial \mu} = -\frac{\partial \Phi\left(-\frac{\mu}{\sigma}\right)}{\partial \mu} = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right). \tag{5.8}$$

$$\frac{\partial \mathbb{P}(Y_+ > 0)}{\partial \sigma} = -\frac{\partial \Phi\left(-\frac{\mu}{\sigma}\right)}{\partial \sigma} = -\frac{\mu}{\sigma^2\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right). \tag{5.9}$$

Proof of Lemma 5.2. Follows directly from the statement by standard differentiation. □

Lemma 5.3.

$$\frac{\partial \mathbb{E}[Y_+]}{\partial \mu} = 1 - \Phi\left(-\frac{\mu}{\sigma}\right). \tag{5.10}$$

$$\frac{\partial \mathbb{E}[Y_+]}{\partial \sigma} = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right). \tag{5.11}$$

Proof of Lemma 5.3.

$$\frac{\partial \mathbb{E}[Y_+]}{\partial \mu} = -\frac{\mu}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right) + \frac{\partial\left[\mu\left(1 - \Phi\left(-\frac{\mu}{\sigma}\right)\right)\right]}{\partial \mu} = 1 - \Phi\left(-\frac{\mu}{\sigma}\right)$$

where the last step follows from Lemma 5.2.

$$\begin{aligned}\frac{\partial \mathbb{E}[Y_+]}{\partial \sigma} &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right) + \frac{\mu^2}{\sigma^2\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right) + \frac{\partial\left[\mu\left(1 - \Phi\left(-\frac{\mu}{\sigma}\right)\right)\right]}{\partial \sigma} \\ &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right)\end{aligned}$$

where the last step follows from Lemma 5.2. □

Lemma 5.4.

$$\frac{\partial \mathbb{E}[Y_+]^2}{\partial \mu} = 2\mathbb{E}[Y_+]\left(1 - \Phi\left(-\frac{\mu}{\sigma}\right)\right). \tag{5.12}$$

$$\frac{\partial \mathbb{E}[Y_+]^2}{\partial \sigma} = 2\mathbb{E}[Y_+]\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{\mu^2}{2\sigma^2}\right). \tag{5.13}$$

Proof of Lemma 5.4. Follows directly from Lemma 5.3. □

Lemma 5.5.

$$\frac{\partial \mathbb{P}(Y_+ > 0)^2}{\partial \mu} = 2\mathbb{P}(Y_+ > 0)\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{\mu^2}{2\sigma^2}\right). \tag{5.14}$$

$$\frac{\partial \mathbb{P}(Y_+ > 0)^2}{\partial \sigma} = -2\mathbb{P}(Y_+ > 0)\frac{\mu}{\sigma^2\sqrt{2\pi}}\exp\left(-\frac{\mu^2}{2\sigma^2}\right). \tag{5.15}$$

Proof of Lemma 5.5. Follows directly from Lemma 5.2. □

5.7 Experimental details

All images were resized to size 112×112 and aligned using a pre-trained aligner[2]. For the Arcloss function, we used the recommended parameters of margin $m = 0.5$ and temperature $s = 64$. We trained our models on 4 NVIDIA Tesla V-100 GPUs using SGD with a learning rate of 0.001, momentum of 0.9. Both the architectures were trained for a total of 230k steps, with the learning rate being decayed by a factor of 10 after 170k steps. We use a batch size of 256 and 64 per GPU for MobileFaceNet for ResNet respectively.

Pre-training in SDH is performed in the same way as described above. The hash learning step is trained on a single GPU with a learning rate of 0.001. The ResNet model is trained for 200k steps with a batch size of 64, and the MobileFaceNet model is trained for 150k steps with a batch size of 256. We set the number of active bits $k = 3$ and a pairwise cost of $p = 0.1$.

Hyper-parameters for MobileNet models.

1. The regularization parameter λ for the $\widetilde{\mathcal{F}}$ regularizer was varied as 200, 300, 400, 600.
2. The regularization parameter λ for the ℓ_1 regularizer was varied as 1.5, 2.0, 2.7, 3.5.
3. The PCA dimension is varied as 64, 96, 128, 256.
4. The number of LSH bits were varied as 512, 768, 1024, 2048, 3072.
5. For IVF-PQ from the faiss library, the following parameters were fixed: `nlist=4096, M=64, nbit=8`, and `nprobe` was varied as 100, 150, 250, 500, 1000.

[2] `https://github.com/deepinsight/insightface`

Hyper-parameters for ResNet baselines.

1. The regularization parameter λ for the $\widetilde{\mathcal{F}}$ regularizer was varied as 50, 100, 200, 630.
2. The regularization parameter λ for the ℓ_1 regularizer was varied as 2.0, 3.0, 5.0, 6.0.
3. The PCA dimension is varied as 48, 64, 96, 128.
4. The number of LSH bits were varied as 256, 512, 768, 1024, 2048.
5. For IVF-PQ, the following parameters were the same as in MobileNet: `nlist=4096, M=64, nbit=8`. `nprobe` was varied as 50, 100, 150, 250, 500, 1000.

Selecting top-k. We use the following heuristic to create the shortlist of candidates after the sparse ranking step. We first shortlist all candidates with a score greater than some confidence threshold. For our experiments we set the confidence threshold to be equal to 0.25. If the size of this shortlist is larger than k, it is further shrunk by consider the top k scorers. For all our experiments we set $k = 1000$. This heuristic avoids sorting the whole array, which can be a bottleneck in this case. The parameters are chosen such that the time required for the re-ranking step does not dominate the total retrieval time.

Hardware.

1. All models were trained on 4 NVIDIA Tesla V-100 GPUs with 16G of memory.
2. System Memory: 256G.
3. CPU: Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz.
4. Number of threads: 32.
5. Cache: L1d cache 32K, L1i cache 32K, L2 cache 256K, L3 cache 46080K.

All timing experiments were performed on a single thread in isolation.

5.8 Additional Results

5.8.1 Results without re-ranking

Fig. 5.5 shows the comparison of the approaches with and without re-ranking. We notice that there is a significant dip in the performance without re-ranking with the gap being smaller for ResNet with FLOPs regularization. We also notice that the FLOPs regularizers has a better trade-off curve for the no re-ranking setting as well.

5.8.2 FPR and TPR curves

In the main text we have reported the recall@1 which is a standard face *recognition* metric. This however is not sufficient to ensure good face *verification* performance. The goal in face verification is to predict whether two faces are similar or dissimilar. A natural metric in such a scenario is the FPR-TPR curve. Standard face verification datasets include LFW [97] and AgeDB [179]. We produce embeddings using our trained models, and use them to compute similarity scores (dot product) for pairs of images. The similarity scores are used

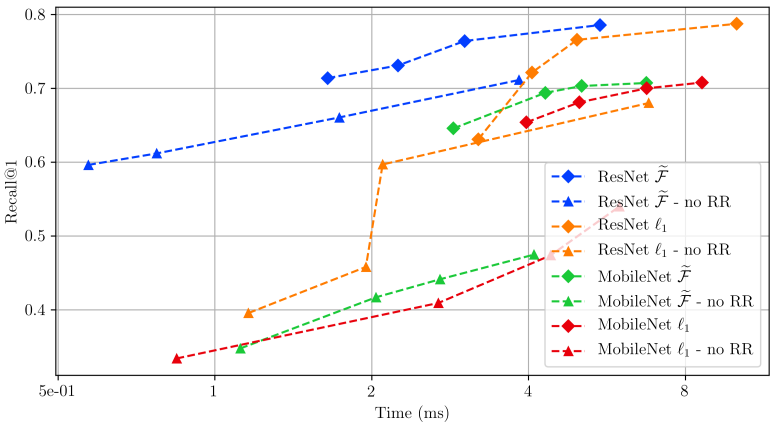


Figure 5.5: Time vs Recall@1 plots for retrieval with and without re-ranking. Results from the same model and regularizer have same colors. Diamonds ($\Diamond$) denote results with re-ranking, and triangles ($\triangle$) denote results without re-ranking.

to compute the FPR-TPR curves which are shown in Fig. 5.6. We notice that for curves with similar probability of activation p, the FLOPs regularizer performs better compared to ℓ_1. This demonstrates the efficient utilization of all the dimensions in the case of the FLOPs regularizer that helps in learning richer representations for the same sparsity.

We also observe that the gap between sparse and dense models is smaller for ResNet, thus suggesting that the ResNet model learns better representations due to increased model capacity. Lastly, we also note that the gap between the dense and sparse models is smaller for LFW compared to AgeDB, thus corroborating the general consensus that LFW is a relatively easier dataset.

5.8.3 Cifar-100 results

We also experimented with the Cifar-100 dataset [135] consisting of 60000 examples and 100 classes. Each class consists of 500 train and 100 test examples. We compare the ℓ_1 and FLOPs regularized approaches with the sparse deep hashing approach. All models were trained using the triplet loss [238] and embedding dim $d = 64$. For the dense and DH baselines, no activation was used on the embeddings. For the ℓ_1 and FLOPs regularized models we used the SThresh activation. Similar to Jeong and Song [108], the train-test and test-test precision values have been reported in Table 5.1. Furthermore, the reported results are without re-ranking. Cifar-100 being a small dataset, we only report the FLOPs-per-row, as time measurements can be misleading. In our experiments, we achieved slightly higher precisions for the dense model compared to [108]. We notice that our models use less than 50% of the computation compared to SDH, albeit with a slightly lower precision.

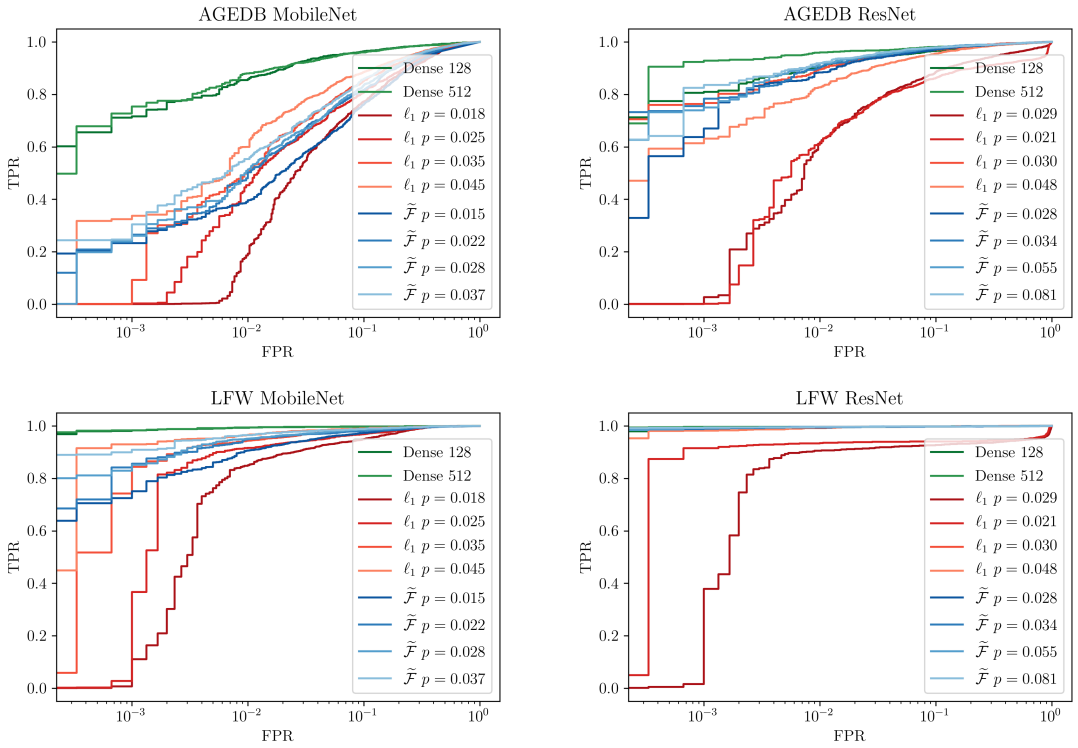


Figure 5.6: FPR-TPR curves. The ℓ_1 curves are all shown in shades of red, where as the FLOPs curves are all shown in shades of blue. The probability of activation is provided in the legend for comparison. For curves with similar probability of activation p, the FLOPs regularizer performs better compared to ℓ_1, thus demonstrating that the FLOPs regularizer learns richer representations for the same sparsity.

Model	F	Train		Test	
		prec@4	prec@16	prec@4	prec@16
Dense	64	61.53	61.26	**57.31**	**56.95**
SDH $k = 1$	**1.18**	**62.29**	**61.94**	57.22	55.87
SDH $k = 2$	3.95	60.93	60.15	55.98	54.42
SDH $k = 3$	8.82	60.80	59.96	55.81	54.10
$\widetilde{F}$ no re-ranking	**0.40**	**61.05**	**61.08**	**55.23**	**55.21**
ℓ_1 no re-ranking	0.47	60.50	60.17	54.32	54.96

Table 5.1: Cifar-100 results using triplet loss and embedding size $d = 64$. For ℓ_1 and $\widetilde{\mathcal{F}}$ models, no re-ranking was used. F is used to denote the FLOPs-per-row (lower is better). The SDH results have been reported from the original paper.

6 | Hierarchically Regularized Deep Forecasting

6.1 Introduction

Multivariate time series forecasting is a key problem in many domains such as retail demand forecasting [24], financial predictions [298], power grid optimization [102], road traffic modeling [152], and online ads optimization [46]. In many of these setting, the problem involves simultaneously forecasting a large number of possibly correlated time series for various downstream applications. In the retail domain, the time series may capture sales of items in a product inventory, and in power grids, the time series may correspond to energy consumption in a household. Often, these time series are arranged in a natural multi-level hierarchy - for example in retail forecasting, items are grouped into subcategories and categories, and arranged in a product taxonomy. In the case of power consumption forecasting, individual households are grouped into neighborhoods, counties, and cities. The hierarchical structure among the time series can usually be represented as a tree, with the leaf nodes corresponding to time series at the finest granularity, and the edges representing parent-child relationships. Fig. 6.1 illustrates a typical hierarchy in the retail forecasting domain for time series of product sales.

In such settings, it is often required to obtain good forecasts, not just for the leaf level time-series (fine grained forecasts), but also for the aggregated time-series corresponding to higher level nodes (coarse gained forecasts). Furthermore, for interpretability and business decision making purposes, it is often desirable to obtain predictions that are roughly *coherent* or *consistent* [103] with respect to the hierarchy tree. This means that the predictions for each parent time-series is equal to the sum of the predictions for its children time-series.

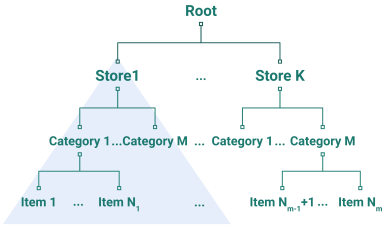


Figure 6.1: An example hierarchy for retail demand forecasting. The blue triangle represents the sub-tree rooted at the node *Store1* with leaves denoted by *Item i*.

More importantly, incorporating coherence constraints in a hierarchical forecasting model captures the natural inductive bias in most hierarchical datasets, where the ground truth parent and children time series indeed adhere to additive constraints. For example, total sales of a product category is equal to the sum of sales of all items in that category.

Some standard approaches for hierarchical forecasting include bottom-up aggregation, or reconciliation-based approaches. Bottom-Up aggregation involves training a model to obtain predictions for the leaf nodes, and then aggregate up along the hierarchy tree to obtain predictions for higher-level nodes. Reconciliation methods [19, 104, 197, 257, 267, 283, 285] make use of a trained model to obtain predictions for all nodes of the tree, and then, in a separate post-processing phase, *reconcile* (or modify) them using various optimization formulations to obtain coherent predictions. Both of these approaches suffer from shortcomings in term of either aggregating noise as one moves to higher level predictions (bottom-up aggregation), or not jointly optimizing the forecasting predictions along with the coherence constraints (for instance, reconciliation).

At the same time, there have been several recent advances on using Deep Neural Network models for multivariate forecasting, including Recurrent Neural Network (RNN), Convolutional Neural Network (CNN) architectures [20, 195, 213, 235], and models designed for multivariate time series based on dimensionality reduction techniques [50, 189, 219, 234, 239, 277], that have been shown to outperform classical time-series models such as autoregressive and exponential smoothing models [100, 101, 169], especially for large datasets. However, most of these approaches do not explicitly address the question of how to model the hierarchical relationships in the dataset. Deep forecasting models based on Graph Neural Networks (GNN) [15, 34, 152, 288, 292] do offer a general framework for learning on graph-structured data. However it is well known [22] that GNNs are hard to scale for learning on graphs with a very large number of nodes, which in real-world settings such as retail forecasting, could involve hundreds of thousands of time series. More importantly, a desirable practical feature for multi-variate forecasting models is to let the prediction of future values for a particular time series only require as input historical data from that time series (along with covariates), without requiring access to historical data from all other time series in the hierarchy. This allows for scalable training and inference of such models using mini-batch gradient descent, without requiring each batch to contain all the time series in the hierarchy. This is often not possible for GNN-based forecasting models, which require batch sizes of the order of the number of time series.

Problem Statement: Based on the above motivations, our goal is to design a hierarchical forecasting model with the following requirements: 1) The model can be trained using a single-stage pipeline on all the time series data, without any separate post-processing, 2) The model captures the additive coherence constraints along the edges of the hierarchy, 3) The model is efficiently trainable on large datasets, without requiring, for instance, batch sizes that scale with the number of time series. Furthermore, in this work we focus on point forecasts only. We also emphasize that, improving forecast accuracy across all levels is our main goal, without restricting to coherent predictions.

We propose a principled methodology to address all these above requirements for hierarchical forecasting. Our model comprises of two components, both of which can support coherence constraints. The first component is purely a function of the historical values of

a time series, without distinguishing between the individual time series themselves in any other way. Coherence constraints on such a model correspond to imposing an additivity property on the prediction function - which constrains the model to be a linear autoregressive model. However, crucially, our model uses time-varying autoregressive coefficients that can themselves be nonlinear functions of the timestamp and other global features (linear versions of time-varying AR have been historically used to deal with non-stationary signals [243]). We will refer to this component as the *time-varying autoregressive model*.

The second component focuses on modeling the global temporal patterns in the dataset through identifying a small set of temporal *global basis functions*. The basis time-series, when combined in different ways, can express the individual dynamics of each time series. In our model, the basis time-series are encoded in a trained seq-2-seq model [256] model in a functional form. Each time series is then associated with a learned embedding vector that specifies the weights for decomposition along these basis functions. Predicting a time series into the future using this model then just involves extrapolating the global basis functions and combining them using its weight vector, without explicitly using the past values of that time series. The coherence constraints therefore only impose constraints on the embedding vectors of each time series, which can be easily modeled by a hierarchical regularization function. We call this component a *basis decomposition model*. As we will see, this part of the model is only approximately coherent unless the embedding constraints hold exactly. In particular, we focus on improving the prediction accuracy rather than preserving exact coherency. In Section 6.7.2, we also provide theoretical justification for how such hierarchical regularization using basis decomposition results in improved prediction accuracy.

We experimentally evaluate our model on multiple publicly available hierarchical forecasting datasets. We compare our approach to state-of-the-art (non-hierarchical) deep forecasting models, GNN-based models and hierarchical models, and show that our approach can obtain consistently more accurate predictions at all levels of the hierarchy tree. Lastly, we consider the multi-objective aspects of this problem and demonstrate the tradeoff between the aggregated metrics at various levels.

6.2 Related Work on Deep Hierarchical Models

In addition to the works referenced in the previous section, we now discuss a few works that are more relevant to our approach. Specifically, we discuss some recent deep hierarchical forecasting methods that do not require a post-processing reconciliation step. Hierarchical forecasting methods can be roughly divided into two categories: *point forecasters* and *probabilistic forecasters*. Mishchenko et al. [175] propose a point-forecasting approach which imposes coherency on a base model via ℓ_2 regularization on the predictions. Gleason [71] extend the idea further to impose the hierarchy on an embedding space rather than the predictions directly. SHARQ [82] follows a similar ℓ_2 regularization based approach as Mishchenko et al. [175], and also extends the idea to probabilistic forecasting. Their model is trained separately for each of the hierarchical levels starting from the leaf level, thus requiring a separate prediction model for each level.

Probabilistic forecasting methods include HierE2E [214] which produces perfectly coherent forecasts by using a projection operation on base predictions from a DeepVAR model [234]. It requires the whole hierarchy of time series to be fed as input to the model leading to a

large number of parameters, and hence may not scale well to large hierarchies. Yanchenko et al. [290] take a fully Bayesian approach by modelling the hierarchy using conditional distributions.

6.3 Problem Setting

We are given a set of N coherent time series of length T, arranged in a pre-defined hierarchy consisting of N nodes. At time step t, the time series data can be represented as a vector $\boldsymbol{y}_t \in \mathbb{R}^N$ denoting the time series values of all N nodes. We compactly denote the set of time series for all T steps as a matrix $\boldsymbol{Y} = [\boldsymbol{y}_1, \cdots, \boldsymbol{y}_T]^\top \in \mathbb{R}^{T \times N}$. Also define $\boldsymbol{y}^{(i)}$ as the ith column of the matrix $\boldsymbol{Y}$ denoting all time steps of the i th time series, and $\boldsymbol{y}_t^{(i)}$ as the t th value of the i th time series. We compactly denote the H-step history of $\boldsymbol{Y}$ by $\boldsymbol{Y}_\mathcal{H} = [\boldsymbol{y}_{t-H}, \cdots, \boldsymbol{y}_{t-1}]^\top \in \mathbb{R}^{H \times N}$ and the H-step history of $\boldsymbol{y}^{(i)}$ by $\boldsymbol{y}_\mathcal{H}^{(i)} = [\boldsymbol{y}_{t-H}^{(i)}, \cdots, \boldsymbol{y}_{t-1}^{(i)}] \in \mathbb{R}^H$. Similarly define the F-step future of $\boldsymbol{Y}$ as $\boldsymbol{Y}_\mathcal{F} = [\boldsymbol{y}_t, \cdots, \boldsymbol{y}_{t+F-1}]^\top \in \mathbb{R}^{F \times N}$. We use the $\widehat{\cdot}$ notation to denote predicted values, for example $\widehat{\boldsymbol{Y}}_\mathcal{F}$, $\widehat{\boldsymbol{y}}_\mathcal{F}^{(i)}$ and $\widehat{\boldsymbol{y}}_t$.

Time series forecasts can often be improved by using features as input to the model along with historical time series. The features often evolve with time, for example, categorical features such as *type of holiday*, or continuous features such as *time of the day*. We denote the matrix of such features by $\boldsymbol{X} \in \mathbb{R}^{T \times D}$, where the t th row denotes the D-dimensional feature vector at the t time step. For simplicity, we assume that the features are *global*, meaning that they are shared across all time series. We similarly define $\boldsymbol{X}_\mathcal{H}$ and $\boldsymbol{X}_\mathcal{F}$ as above.

Hierarchically Coherent Time Series: We assume that the time series data are coherent, that is, they satisfy the *sum constraints* over the hierarchy. The time series at each node of the hierarchy is the equal to the sum of the time series of its children, or equivalently, equal to the sum of the leaf time series of the sub-tree rooted at that node. Fig. 6.1 shows an example of a sub-tree rooted at a node.

As a result of aggregation, the data can have widely varying scales with the values at higher level nodes being magnitudes higher than the leaf level nodes. It is well known that neural network training is more efficient if the data are similarly scaled. Hence, we work with rescaled time series data. The time series at each node is downscaled by the number of leaves in the sub-tree rooted at the node, so that now they satisfy *mean constraints* rather than sum constraints described above. Denote by $\mathcal{L}(p)$, the set of leaf nodes of the sub-tree rooted at p. Hierarchically coherent data satisfy the following *data mean property*,

$$\boldsymbol{y}^{(p)} = \frac{1}{|\mathcal{L}(p)|} \sum_{i \in \mathcal{L}(p)} \boldsymbol{y}^{(i)} \quad \textit{(Data Mean Property)}. \tag{6.1}$$

6.4 Hierarchically Regularized Deep Forecasting - HiReD

We now introduce the two components in our model, namely the *time-varying AR model* and the *basis decomposition model*. As mentioned in the introduction a combination of these two components satisfy the three requirements in our problem statement. In particular,

we shall see that the coherence property plays a central role in both the components. For simplicity, in this section, all our equations will be for forecasting one step into the future ($F = 1$), even though all the ideas trivially extend to multi-step forecasting. The defining equation of our model can be written as,

$$\widehat{\boldsymbol{y}}_{\mathcal{F}}^{(i)} = f(\boldsymbol{y}_{\mathcal{H}}^{(i)}, \boldsymbol{X}_{\mathcal{H}}, \boldsymbol{X}_{\mathcal{F}}, \boldsymbol{Z}_{\mathcal{H}}, \theta_i) = \underbrace{\left\langle \boldsymbol{y}_{\mathcal{H}}^{(i)}, a(\boldsymbol{X}_{\mathcal{H}}, \boldsymbol{X}_{\mathcal{F}}, \boldsymbol{Z}_{\mathcal{H}}) \right\rangle}_{\text{Time varying AR (TVAR)}} + \underbrace{\left\langle \theta_i, b(\boldsymbol{X}_{\mathcal{H}}, \boldsymbol{X}_{\mathcal{F}}, \boldsymbol{Z}_{\mathcal{H}}) \right\rangle}_{\text{Basis decomposition (BD)}}. \quad (6.2)$$

In the above equation, $\boldsymbol{Z}_{\mathcal{H}}$ is a latent state vector that summarizes the temporal information about the whole dataset, and $\theta_i \in \mathbb{R}^K$ is the embedding/weight vector for time-series i in the basis decomposition model. The functions $a(\boldsymbol{X}_{\mathcal{H}}, \boldsymbol{X}_{\mathcal{F}}, \boldsymbol{Z}_{\mathcal{H}}) \in \mathbb{R}^H$ and $b(\boldsymbol{X}_{\mathcal{H}}, \boldsymbol{X}_{\mathcal{F}}, \boldsymbol{Z}_{\mathcal{H}}) \in \mathbb{R}^K$ are not dependent on $\boldsymbol{y}_{\mathcal{H}}^{(i)}$ and θ_i. The parameters of the functions a, b and the embeddings θ_i are learned from the data. The above equation describes our model for single future time step. It can be extended to multi-horizon prediction in a straightforward manner as summarized in Fig. 6.2. We provide further details as we delve into the individual components.

Summary Vector $\boldsymbol{Z}_{\mathcal{H}}$: The vector $\boldsymbol{Z}_{\mathcal{H}}$ can be any relatively low-dimensional temporally evolving variable that captures information about the global state of the dataset at a particular time. While several dimensionality reduction approaches can be used here, in our approach, we use the *Non-Negative Matrix Factorization* (NMF) algorithm by Gillis and Vavasis [69] to select a small set of representative time series that encode the global state. If the indices of the selected representative time-series is denoted by $\{i_1, \cdots, i_R\}$ (R denotes the *rank* of the factorization), then we define $\boldsymbol{Z} = [\boldsymbol{Y}^{(i_1)}, \cdots, \boldsymbol{Y}^{(i_R)}] \in \mathbb{R}^{T \times R}$. Note that we only feed the past values $\boldsymbol{Z}_{\mathcal{H}}$ as input to the model, since future values are not available when making future forecasts. Also, note that the final basis time-series is a non-linear function of $\boldsymbol{Z}_{\mathcal{H}}$. We tune the value of R in our experiments, and observe that it is much smaller than N in all cases. Further details about $\boldsymbol{Z}$ including other possible alternatives to dimensionality reduction can be found in Section 6.8.2.

Time-Varying AR (TVAR): The first part of the expression in Eq. (6.2) denoted by *Time Varying AR (TVAR)* resembles a linear auto-regressive model with coefficients $a(\boldsymbol{X}_{\mathcal{H}}, \boldsymbol{X}_{\mathcal{F}}, \boldsymbol{Z}_{\mathcal{H}}) \in \mathbb{R}^H$, that are a function of the input features, and thus can change with time. The AR parameters of this model are shared across all time series and hence do not encode any time-series specific information, a drawback that is overcome by the Basis Decomposition part of our model. This component is *coherent by design* because it is a shared linear AR model. However, even though the AR weights are shared across all the time-series at a given time-point, they can crucially change with time, thus lending more flexibility to the model.

Implementation: In order to model the sequential nature of the data, we use an LSTM encoder to encode the past $\boldsymbol{X}_{\mathcal{H}}$ and $\boldsymbol{Z}_{\mathcal{H}}$. Then, we use a fully connected (FC) decoder for predicting the auto-regressive weights. Similar to Wen et al. [281]'s multi-horizon approach, we use a different head of the decoder for each future time step resulting in a F-headed decoder producing F-step predictions for TVAR weights. The decoder also takes as input the future covariates $\boldsymbol{X}_{\mathcal{F}}$ if available. The produced weights are then multiplied (inner product) to the history to produce the final TVAR predictions. We illustrate this architecture in Fig. 6.2 (bottom).

Basis Decomposition (BD) with Hierarchical Regularization: Now we come to the second part of our model in Eq. (6.2). As discussed before, this part of the model has per time-series adaptivity, as different time-series can have different embeddings. It resembles an expansion of the time series on a set of basis functions $b(\boldsymbol{X}_{\mathcal{H}}, \boldsymbol{X}_{\mathcal{F}}, \boldsymbol{Z}_{\mathcal{H}}) \in \mathbb{R}^K$, with the basis weights/embedding for time series i denoted by $\theta_i \in \mathbb{R}^K$. Both the basis functions and the time series specific weights are learned from the data, rather than fixing a specific form such as Fourier or Wavelet basis.

The idea of using a basis has also been recently invoked in the time-series literature [239, 277]. The basis recovered in the implementation of Wang et al. [277] is allowed to vary for each individual time-series and therefore is not a true basis. Sen et al. [239] do explicitly recover an approximate basis in the training set through low-rank matrix factorization regularized by a deep global predictive model alternatingly trained on the basis vectors, thus not amenable to end-to-end optimization. We shall see that our model can be trained in an end-to-end manner.

Embedding Regularization for Approximate Coherency: The TVAR part of our model is coherent by design due to its linearity. The BD model however requires the embeddings of the time-series to satisfy the mean property along the hierarchy. This directly translates to coherency of the predictions due to linearity with respect to θ.

$$\theta_p = \frac{1}{|\mathcal{L}(p)|} \sum_{i \in \mathcal{L}(p)} \theta_i \quad (\textit{Embedding Mean Property}), \tag{6.3}$$

We impose this constraint approximately via an ℓ_2 regularization on the embedding.

$$E_{\text{reg}}(\boldsymbol{\theta}) = \sum_{p=1}^{N} \sum_{i \in \mathcal{L}(p)} \|\theta_p - \theta_i\|_2^2. \tag{6.4}$$

The purpose of this regularizer is two fold. Firstly, we observe that, when the leaf embeddings are kept fixed, the regularizer is minimized when the embeddings satisfy the mean property in Eq. (6.3), thus encouraging coherency in the predictions. Secondly, it also encodes the inductive bias present in the data corresponding to the hierarchical additive constraints. We provide some theoretical justification for this hierarchical regularization in Section 6.6.

Implementation: As before, we use an LSTM encoder to encode the past $\boldsymbol{X}_{\mathcal{H}}$ and $\boldsymbol{Z}_{\mathcal{H}}$. Then, we use the encoding from the encoder along with the future features $\boldsymbol{X}_{\mathcal{F}}$ (sequential in nature) and pass them through an LSTM decoder to yield the F-step basis predictions which are then multiplied with the embedding (inner product) to produce the final BD predictions. Thus, a functional representation of the basis time-series is implicitly maintained within the trained weights of the basis generating seq-2-seq model. Note that the embeddings are also trained in our end-to-end model. We illustrate this architecture in Fig. 6.2 (top).

We emphasize that the main ideas in our model are agnostic to the specific type of neural network architecture used. For our experiments, we specifically use an LSTM architecture [94] for the encoder and decoder. Other types of architectures including transformers [268] and temporal convolution networks [23] can also be used.

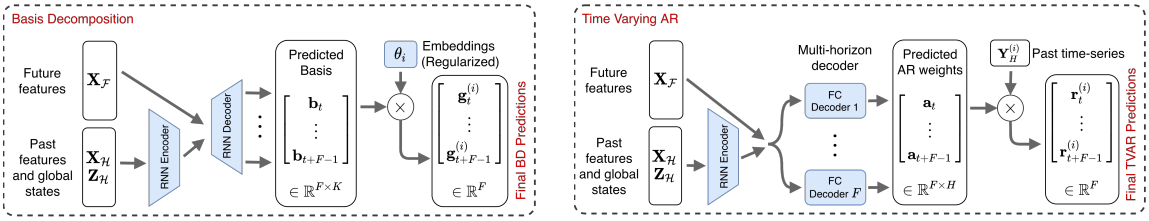


Figure 6.2: In this figure we show the architectures of our two model components separately. At the top we show the BD model, where the seq-2-seq model implicitly maintains the basis in a functional form. Note that the time-series specific weights $\{\theta_i\}$ are also trained. At the bottom, we show the TVAR model. The fully connected decoder has a different prediction head for each future time-point.

Loss Function: During training, we minimize the mean absolute error (MAE) of the predictions along with the embedding regularization term introduced above (our method generalizes to other losses too, such as mean square error, or mean absolute percentage error). For regularization weight λ_E, and $\widehat{\boldsymbol{y}}_{\mathcal{F}}^{(i)}$ defined as Eq. (6.2), and Θ denoting the trainable parameters of a, b, our training loss function is,

$$\ell(\Theta, \boldsymbol{\theta}) = \underbrace{\sum_i \sum_{\mathcal{F}} |\boldsymbol{y}_{\mathcal{F}}^{(i)} - \widehat{\boldsymbol{y}}_{\mathcal{F}}^{(i)}|}_{\text{Prediction loss}} + \underbrace{\lambda_E E_{\text{reg}}(\boldsymbol{\theta})}_{\text{Embedding regularization}} . \tag{6.5}$$

We minimize the above loss function using stochastic gradient descent. In each iteration, the prediction loss computed using a mini-batch sampled from the training data, where as the regularization loss is computed using all the embeddings. The embedding matrix being relatively small in size does not affect the scalability of our approach.

Model Justification. We provide a more concrete justification of the above model. Our goal is to learn a function f of the form $\widehat{\boldsymbol{y}}_{\mathcal{F}}^{(i)} = f(\boldsymbol{y}_{\mathcal{H}}^{(i)}, \boldsymbol{X}_{\mathcal{H}}, \boldsymbol{X}_{\mathcal{F}}, \boldsymbol{Z}_{\mathcal{H}}, \theta_i)$, with the main objectives of our approach is to model coherency while being scalable. For our approach, predictions for time series i only need access to the history of the ith time series $\boldsymbol{y}_{\mathcal{H}}^{(i)}$ and θ_i. For a prediction function to be coherent, the prediction for a parent node must be equal to the aggregated predictions of its children nodes. Assuming that θ_is satisfy Eq. (6.3), a sufficient condition for the prediction function f to be coherent with respect to $\boldsymbol{y}_{\mathcal{H}}^{(i)}$ and θ_i is for it to be linear with respect to $\boldsymbol{y}_{\mathcal{H}}^{(i)}$ and θ_i. Linearity leads to the final form of the model shown in Eq. (6.2). The constraint in Eq. (6.3) leads to the regularization term in the training loss shown in Eq. (6.5). This is one of the key motivations behind our approach.

6.5 Experiments

We implemented our proposed model in Tensorflow [1] and compared against multiple baselines on popular hierarchical time-series datasets.

Datasets. We experimented with three hierarchical forecasting datasets - Two retail forecasting datasets, M5 [160] and Favorita [60]; and the Tourism [262] dataset consisting of

tourist count data. The history length and forecast horizon (H, F) were set to (28, 7), (28, 7) and (24, 4), for Favorita, M5 and Tourism respectively.

1. The M5 dataset[1] consists of time series data of product sales from 10 Walmart stores in three US states. The data consists of two different hierarchies: the product hierarchy and store location hierarchy. For simplicity, in our experiments we use only the product hierarchy consisting of 3k nodes and 1.8k time steps. The validation scores are computed using the predictions from time steps 1843 to 1877, and test scores on steps 1878 to 1913.

2. The Favorita dataset[2] is a similar dataset, consisting of time series data from Corporación Favorita, a South-American grocery store chain. As above, we use the product hierarchy, consisting of 4.5k nodes and 1.7k time steps. The validation scores are computed using the predictions from time steps 1618 to 1652, and test scores on steps 1653 to 1687.

3. The Australian Tourism dataset[3] consists of monthly domestic tourist count data in Australia across 7 states which are sub-divided into regions, sub-regions, and visit-type. The data consists of around 500 nodes and 230 time steps. The validation scores are computed using the predictions from time steps 122 to 156, and test scores on steps 157 to 192.

We divide each of the datasets into training, validation and test sets, with details on the splits provided in Section 6.8.1.

Baselines. We compare our proposed approach HiRed with the following baseline models:

1. *RNN*: We use an LSTM decoder and encoder to implement a seq-2-seq model shared across all the time series, trained using mean absolute error loss.

2. *DeepGLO* [239]: We use the implementation released by the authors on Github[4]. We modify the loss function, data handling and evaluation to adapt to our setting.

3. *DCRNN* [152]: DCRNN being a GNN based approach requires a *correlation* graph as input. We use the official implementation released by the authors[5] and provide the hierarchy tree as the input graph. The implementation uses MAE loss by default.

4. *Deep Factors (DF)* [277]: The original implementation released by the authors makes rolling probabilistic forecasts. We implement our own version in Tensorflow using an LSTM encoder for the global model (producing point predictions) as in the original implementation, while leaving out the probabilistic local model. We manually tune the hyper-parameters for each of the datasets on the validation set.

5. L_2Emb [71], which is an improvement over Mishchenko et al. [175]. We implement this model using an LSTM decoder and encoder with MAE as the main training loss.

[1] `https://www.kaggle.com/c/m5-forecasting-accuracy/`
[2] `https://www.kaggle.com/c/favorita-grocery-sales-forecasting/`
[3] `https://robjhyndman.com/publications/mint/`
[4] `https://github.com/rajatsen91/deepglo`
[5] `https://github.com/liyaguang/DCRNN/`

In addition, we also use node embeddings which are fed as input to the encoder and decoder, and regularized according to the hierarchy as described by Gleason [71].

6. *SHARQ* [82]: We were not able to find an official release of SHARQ. We implemented it using an LSTM based seq-to-seq model, with layer-wise training as described in the paper. We used MAE as the *data fit loss* function and the default squared error *reconciliation loss* regularizer (See Han et al. [82] for terminology).

7. *HierE2E-PF* is a modification of HierE2E [214]. The official implementation produces rolling probabilistic forecasts. We adapt it to point forecasts by removing the variance model and using the proposed projection step on outputs from a seq-2-seq model.

8. *RNN+ERM* [19]: We also compare with the recent *ERM* [19] reconciliation method applied to the base forecasts from the RNN model. It has been shown in [19] to outperform many previous reconciliation techniques such as MinT [284]. This approach involves learning a sparse projection matrix from data, resulting in coherent predictions. We tune the sparsity parameter using the model accuracy on the validation set.

For a fair comparison, we use the the same Mean Absolute Error loss functions for training, and ensure that all the baseline models have similar number of parameters. We implement some of baselines since the official implementations of some of the baselines either make rolling probabilistic forecasts, or use a different set of covariates. Further details about the baselines and training parameters can be found in Section 6.8.

Metrics. We compare the accuracy of the various approaches with respect to the following two metrics. Denote the true values by $\boldsymbol{y}$ and the predicted values by $\widehat{\boldsymbol{y}}$, both n-dimensional vectors.

1. Symmetric mean absolute percent error SMAPE $= \frac{2}{n}\sum_i \frac{|\widehat{\boldsymbol{y}}_i - \boldsymbol{y}_i|}{|\boldsymbol{y}_i| + |\widehat{\boldsymbol{y}}_i|}$.

2. Weighted absolute percentage error WAPE $= \frac{\sum_i |\widehat{\boldsymbol{y}}_i - \boldsymbol{y}_i|}{\sum_i |\boldsymbol{y}_i|}$.

We report the metrics on the test data, for each level of the hierarchy (with level 0 denoting the root) in Table 6.1. As a measure of the aggregate performance across all the levels, we also report the mean of the metrics in all the levels of the hierarchy denoted by *Mean*.

6.5.1 Results

Table 6.1 shows the averaged test metrics for M5, Favorita, and Tourism datasets, along with confidence intervals.

We find that for all three datasets, our proposed model either yields the smallest error or close to the smallest error across most metrics and most levels. In particular, we find that our proposed method achieves the smallest errors in the mean column for all datasets in terms of WAPE and SMAPE, thus indicating good performance generally across all levels. We find that RNN+ERM in general, yields an improvement over the base RNN predictions for the higher levels closer to the root node (Levels 0 and 1), while, worsening at the lower levels. DCRNN, despite using the hierarchy as a graph also does not perform as well as our approach, especially in Tourism and M5 Datasets - possibly due to the fact that a GNN

Table 6.1: WAPE/SMAPE test metrics for all the three datasets, averaged over 10 runs. The standard deviations are shown in the parenthesis. We bold the smallest mean in each column and anything that comes within two standard deviations.

M5	Level 0		Level 1		Level 2		Level 3		Mean	
HiReD	**0.048** (0.0011)	**0.048** (0.0011)	**0.055** (0.0006)	**0.053** (0.0006)	**0.072** (0.0007)	**0.077** (0.0006)	0.279 (0.0003)	0.511 (0.0012)	**0.113** (0.0005)	**0.172** (0.0006)
RNN	0.059 (0.002)	0.059 (0.003)	0.083 (0.013)	0.083 (0.011)	0.085 (0.002)	0.098 (0.004)	0.282 (0.006)	0.517 (0.007)	0.127 (0.005)	0.189 (0.005)
DF	0.055 (0.001)	0.056 (0.001)	0.061 (0.001)	0.060 (0.001)	0.076 (0.001)	0.085 (0.002)	0.272 (0.000)	**0.501** (0.002)	0.116 (0.001)	0.176 (0.001)
DeepGLO	0.077 (0.0003)	0.081 (0.0004)	0.087 (0.0003)	0.092 (0.0004)	0.099 (0.0003)	0.113 (0.0003)	0.278 (0.0001)	0.538 (0.0001)	0.135 (0.0003)	0.206 (0.0003)
DCRNN	0.078 (0.006)	0.079 (0.007)	0.096 (0.005)	0.092 (0.004)	0.165 (0.003)	0.193 (0.007)	0.282 (0.000)	0.512 (0.000)	0.156 (0.002)	0.219 (0.003)
L_2Emb	0.055 (0.0016)	0.056 (0.001)	0.064 (0.0014)	0.063 (0.001)	0.080 (0.0011)	0.092 (0.001)	**0.269** (0.0003)	**0.501** (0.003)	0.117 (0.0009)	0.178 (0.001)
SHARQ	0.093 (0.002)	0.096 (0.002)	0.071 (0.004)	0.062 (0.003)	0.099 (0.002)	0.094 (0.001)	0.277 (0.000)	0.528 (0.000)	0.135 (0.001)	0.195 (0.001)
RNN+ERM	0.052 (0.001)	0.052 (0.001)	0.066 (0.001)	0.071 (0.002)	0.084 (0.001)	0.104 (0.002)	0.286 (0.002)	0.520 (0.004)	0.122 (0.001)	0.187 (0.001)
HierE2E-PF	0.152 (0.002)	0.160 (0.002)	0.152 (0.002)	0.158 (0.002)	0.152 (0.002)	0.181 (0.002)	0.396 (0.001)	0.615 (0.002)	0.213 (0.002)	0.278 (0.002)
HierE2E-PF Large	**0.047** (0.002)	0.050 (0.003)	0.057 (0.001)	0.063 (0.001)	**0.067** (0.001)	0.080 (0.001)	0.347 (0.001)	0.573 (0.001)	0.130 (0.001)	0.192 (0.001)

Favorita	Level 0		Level 1		Level 2		Level 3		Mean	
HiReD	0.061 (0.002)	**0.061** (0.002)	**0.094** (0.001)	**0.182** (0.002)	**0.127** (0.001)	**0.267** (0.003)	0.210 (0.000)	**0.322** (0.004)	**0.123** (0.001)	**0.208** (0.002)
RNN	0.067 (0.004)	0.068 (0.003)	0.114 (0.003)	0.197 (0.004)	0.134 (0.002)	0.290 (0.005)	**0.203** (0.001)	0.339 (0.005)	0.130 (0.002)	0.223 (0.004)
DF	0.064 (0.003)	0.064 (0.004)	0.110 (0.002)	0.194 (0.003)	0.135 (0.002)	0.291 (0.007)	0.213 (0.001)	0.343 (0.007)	0.130 (0.002)	0.223 (0.004)
DeepGLO	0.098 (0.001)	0.088 (0.001)	0.126 (0.001)	0.197 (0.001)	0.156 (0.001)	0.338 (0.001)	0.226 (0.001)	0.404 (0.001)	0.151 (0.001)	0.256 (0.001)
DCRNN	0.080 (0.004)	0.080 (0.005)	0.120 (0.001)	0.212 (0.002)	0.134 (0.000)	0.328 (0.000)	**0.204** (0.000)	0.389 (0.000)	0.134 (0.001)	0.252 (0.001)
L_2Emb	0.070 (0.003)	0.070 (0.003)	0.114 (0.002)	0.199 (0.004)	0.136 (0.001)	0.276 (0.006)	0.207 (0.001)	**0.321** (0.007)	0.132 (0.002)	0.216 (0.004)
SHARQ	0.088 (0.002)	0.085 (0.002)	0.142 (0.001)	0.199 (0.001)	0.156 (0.001)	0.335 (0.001)	0.230 (0.000)	0.404 (0.000)	0.154 (0.000)	0.256 (0.000)
RNN+ERM	**0.056** (0.002)	**0.058** (0.002)	0.103 (0.001)	0.185 (0.003)	0.129 (0.001)	0.283 (0.005)	0.220 (0.001)	0.348 (0.005)	0.127 (0.001)	0.219 (0.003)
HierE2E-PF	0.120 (0.005)	0.125 (0.006)	0.206 (0.003)	0.334 (0.005)	0.247 (0.002)	0.448 (0.006)	0.409 (0.007)	0.573 (0.014)	0.245 (0.004)	0.370 (0.007)
HierE2E-PF Large	0.082 (0.002)	0.077 (0.002)	0.168 (0.003)	0.263 (0.010)	0.190 (0.002)	0.360 (0.003)	0.314 (0.002)	0.440 (0.001)	0.189 (0.002)	0.285 (0.003)

Tourism	Level 0		Level 1		Level 2		Level 3		Level 4		Mean	
HiReD	**0.059** (0.001)	**0.061** (0.001)	**0.125** (0.001)	0.162 (0.003)	**0.172** (0.001)	0.225 (0.002)	**0.229** (0.001)	0.376 (0.004)	**0.347** (0.001)	**0.786** (0.007)	**0.186** (0.001)	**0.322** (0.002)
RNN	0.110 (0.001)	0.106 (0.001)	0.148 (0.001)	0.164 (0.002)	0.188 (0.001)	0.231 (0.001)	0.240 (0.000)	0.385 (0.006)	0.369 (0.001)	**0.782** (0.012)	0.211 (0.001)	0.333 (0.002)
DF	0.097 (0.003)	0.096 (0.002)	0.141 (0.002)	0.170 (0.002)	0.187 (0.001)	0.240 (0.002)	0.241 (0.001)	0.380 (0.002)	0.355 (0.000)	**0.783** (0.014)	0.204 (0.001)	0.334 (0.003)
DeepGLO	0.089 (0.0002)	0.079 (0.0002)	**0.126** (0.0001)	**0.158** (0.0001)	0.179 (0.0001)	**0.218** (0.0001)	0.234 (0.0001)	**0.372** (0.0001)	0.364 (0.0001)	0.900 (0.0002)	0.199 (0.0001)	0.346 (0.0001)
DCRNN	0.187 (0.003)	0.171 (0.003)	0.231 (0.002)	0.248 (0.003)	0.258 (0.001)	0.279 (0.002)	0.293 (0.001)	0.398 (0.001)	0.434 (0.000)	0.865 (0.000)	0.281 (0.000)	0.392 (0.001)
L_2Emb	0.114 (0.007)	0.115 (0.007)	0.153 (0.002)	0.180 (0.004)	0.192 (0.002)	0.244 (0.002)	0.245 (0.001)	0.385 (0.002)	0.372 (0.002)	**0.789** (0.010)	0.215 (0.002)	0.342 (0.003)
SHARQ	0.100 (0.005)	0.104 (0.005)	0.164 (0.002)	0.209 (0.001)	0.217 (0.003)	0.260 (0.002)	0.265 (0.003)	0.386 (0.001)	0.399 (0.003)	0.931 (0.004)	0.229 (0.001)	0.378 (0.001)
RNN+ERM	0.078 (0.005)	0.079 (0.005)	0.155 (0.003)	0.206 (0.006)	0.225 (0.004)	0.291 (0.006)	0.307 (0.006)	0.498 (0.008)	0.488 (0.009)	1.013 (0.010)	0.251 (0.005)	0.417 (0.006)
HierE2E-PF	0.110 (0.002)	0.113 (0.002)	0.143 (0.002)	0.161 (0.003)	0.187 (0.002)	0.232 (0.003)	0.240 (0.001)	0.371 (0.004)	0.358 (0.001)	0.824 (0.003)	0.208 (0.001)	0.340 (0.002)

Table 6.2: We report the test WAPE/SMAPE metrics for an ablation study on the M5 dataset, for each of the components in the HiReD model. We compare our model with two ablated variants: first, we remove the regularization ($\lambda_E = 0$), and second, we remove the BD component (TVAR only).

M5 Abl	Level 0	Level 1	Level 2	Level 3	Mean
HiReD	**0.048** / **0.048**	**0.055** / **0.053**	**0.072** / **0.077**	**0.279** / **0.511**	**0.113** / **0.172**
$\lambda_E = 0$	0.054 / 0.054	0.058 / 0.056	0.074 / 0.078	**0.279** / 0.513	0.116 / 0.175
TVAR only	0.050 / 0.049	0.064 / 0.065	0.084 / 0.086	0.288 / 0.520	0.122 / 0.180

is not the most effective way to model the tree hierarchies. We notice that HierE2E-PF performs reasonably well for the smaller Tourism while performing badly for the larger M5 and Favorita datasets - a possible explanation being that this is a VAR model that requires much more parameters to scale to large datasets. Therefore, for HierE2E-PF we perform experiments with 50× more parameters for M5 and Favorita and report the results in Table 6.1, showing that while the results improve, it still performs much worse than our model. Overall, we find that our proposed method consistently works better or at par with the other baselines at all hierarchical levels.

Ablation study. Next, we perform an ablation study of our proposed model to understand the effects of its various components, the results of which are presented in Table 6.2. We compare our proposed model, to the same model without any regularization (set $\lambda_E = 0$ in Eq. (6.5)), and a model consisting of only TVAR. We find that both these components in our model are important, and result in improved accuracy in most metrics.

Coherence. We also compare the coherence of our predictions to that of the RNN model and an ablated model with $\lambda_E = 0$. Specifically, for each node p we measure the deviation of our forecast from $\boldsymbol{c}^{(p)} = 1/\mathcal{L}(p) \sum_{i \in \mathcal{L}(p)} \widehat{\boldsymbol{y}}^{(i)}$, the mean of the leaf node predictions of the corresponding sub-tree. Perfectly coherent predictions will have a zero deviation from this quantity. In Table 6.3, we report the WAPE metric between the predictions from our model $\widehat{\boldsymbol{y}}$ and $\boldsymbol{c}$, for each of the hierarchical levels. The leaf level predictions are trivially coherent. We find that our proposed model consistently produces more coherent predictions compared to both the models, indicating that our hierarchical regularization indeed encourages coherency in predictions, in addition to improving accuracy.

Basis Visualization. We visualize the basis generated by the BD model for the M5 validation set in Fig. 6.3 (left). We notice that the bases capture various global temporal patterns in the dataset. In particular, most of the bases have a period of 7, indicating that they represent weekly patterns. We also show the predictions made from the various components of our model at all hierarchical levels, in Fig. 6.3 (right). We notice that the predictions from the BD part closely resemble the general patterns of the true time series values, where as the AR model adds further adjustments to the predictions, including a constant bias, for most time series. For the leaf level (Level 3) predictions however, the final prediction is dominated by the AR model indicating that global temporal patterns may be less useful in this case.

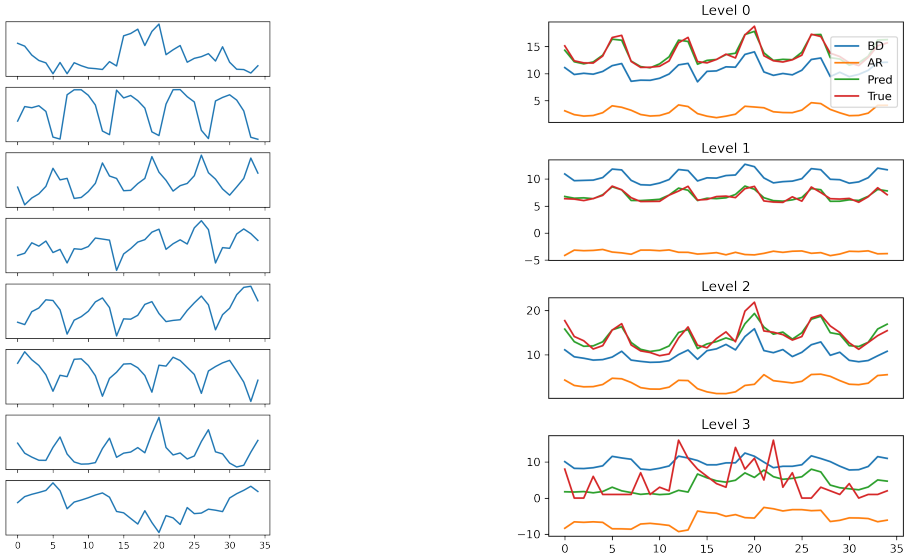


Figure 6.3: Left: Plots of the basis generated on the validation set of the M5 dataset over 35 days. Right: We plot the true time series over the same time period, and compare it with the predicted time series, AR predictions and BD predictions.

		L0	L1	L2	L3
Favorita	HiReD	**0.004**	**0.004**	**0.003**	-
	$\lambda_E = 0$	0.012	0.013	0.010	-
	RNN	0.043	0.044	0.042	-
M5	HiReD	**0.030**	**0.034**	**0.034**	-
	$\lambda_E = 0$	0.035	0.040	0.039	-
	RNN	0.042	0.057	0.047	-
Tourism	HiReD	0.092	**0.079**	**0.066**	0.060
	$\lambda_E = 0$	**0.085**	0.082	0.067	**0.059**
	RNN	0.097	0.089	0.082	0.083

Table 6.3: Coherency metric for all our datasets, at all hierarchical levels. Leaf node metrics are identically zero, and hence not reported in the table. Leaf nodes for Favorita and M5 are denoted by L3. Tourism has 5 hierarchical levels and hence L3 values are reported in this case.

Multi-Objective Considerations

So far, for our experiments, we do not perform any kind of loss function weighting for the various hierarchical levels. As a result, the higher levels are extremely under represented in the training loss due the presence of a large number of leaf nodes. In this section, we study the effect of up-sampling the higher level nodes (in the training loss) on the *mean* metric which is the average metric across all levels.

The loss function is the same as Eq. (6.5). When selecting a mini-batch, instead of randomly sampling a subset of nodes uniformly, as described in Section 6.8.1, we non-uniformly sample (with replacement) each node with probability proportional to,

$$\pi_i \propto \alpha^{-d(i)}, \quad \text{where } d(i) \text{ denotes the depth of node } i. \tag{6.6}$$

The depth of the root node is defined to be zero. The parameter α controls the how much the probability distribution π diverges from the uniform distribution. The value of $\alpha = 1$ corresponds to an uniform distribution, where as higher values of α correspond to over-sampling the higher level nodes. For a complete n-ary tree, a value of $\alpha = n$ corresponds to an uniform representation of each of the levels in the loss functions, meaning that the number of nodes in a minibatch from each of the levels is almost equal. Note that since we sample without replacement, the root is sampled more often to match the number of nodes from the other levels. The rest of the training remains the same.

The results of this experiment are shown in Fig. 6.4. For each of the datasets, we plot the level wise metrics for different values of α. Since the metrics vary widely across the levels, we normalize the values to have a unit mean. The unnormalized values of the runs are reported in Table 6.4.

We observe that in most cases, increasing the sampling rate of the higher level nodes results in improvement of performance at the higher levels. We also notice that the optimum mean performance is not necessarily achieved for $\alpha = 1$, which corresponds to uniform sampling of the nodes. The key take-away from this experiment is that, with the current approach, the metrics at all levels cannot be optimized simultaneously. In practice, different prediction model may be used for different levels.

6.6 Theoretical Justification for Hierarchical Modeling

In this section, we theoretically analyze the benefits of modeling hierarchical constraints in a much simplified setting, and show how it can result in provably improved accuracy, under some assumptions. Since analyzing our actual deep non-linear model for an arbitrary hierarchical set of time series can be complex, we make some simplifying assumptions to the problem and model. We assume that all the time series in the dataset is a linear combination of a small set of basis time series. That is, $\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{\theta} + \boldsymbol{w}$, where $\boldsymbol{B} \in \mathbb{R}^{T \times K}$ denotes the set of basis vectors, $\boldsymbol{\theta} = [\theta_1, \cdots, \theta_N] \in \mathbb{R}^{K \times N}$ denotes the set of weight vectors used in the linear combination for each time series, and $\boldsymbol{w} \in \mathbb{R}^{T \times N}$ denotes the noise matrix sampled i.i.d as $w \sim \mathcal{N}(0, \sigma^2)$ for the leaf nodes. A classical example of such a basis set can be a small subset of Fourier or Wavelet basis [254, 265] that is relevant to the dataset. Note that we ignore the TVAR model for the sake of analysis and focus mainly on the BD model which includes the hierarchical regularization.

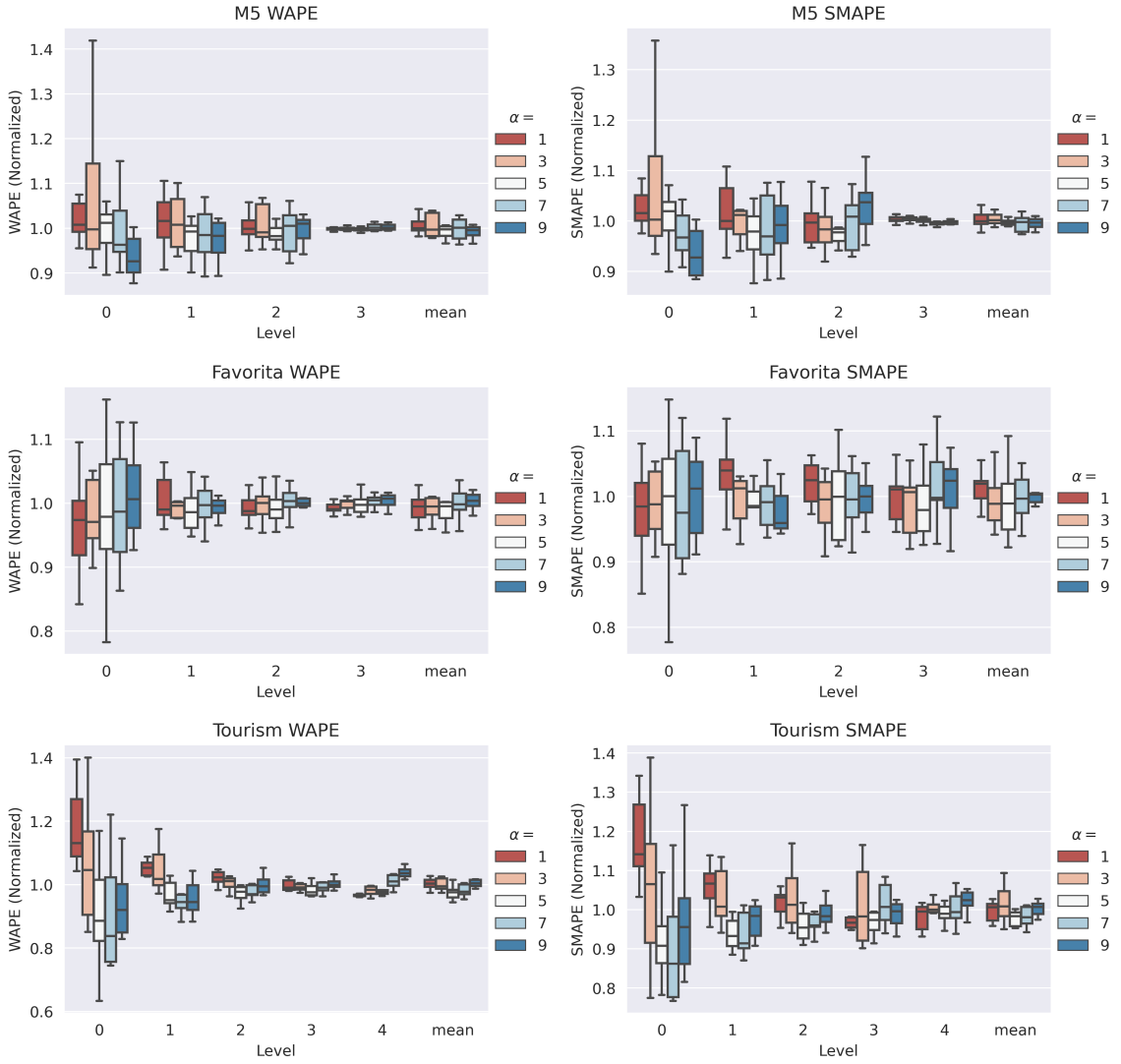


Figure 6.4: The plots show the WAPE and SMAPE metrics when training with non-uniformly sampled mini-batches. Since the metrics vary widely across levels, we normalize all the metrics runs to have a mean of 1, in order to compare the relative improvements. The bars denote 25-75% confidence intervals where as the whiskers show 5-95% confidence intervals.

Table 6.4: WAPE/SMAPE test metrics for all the three datasets, averaged over 10 runs, with non-uniform mini-batch sampling. The standard deviations are shown in the parenthesis.

M5	Level 0	Level 1	Level 2	Level 3	Mean
$\alpha = 1$	0.0488 (0.0012) / 0.0486 (0.0010)	0.0551 (0.0010) / 0.0529 (0.0010)	0.0711 (0.0009) / 0.0767 (0.0010)	0.2798 (0.0004) / 0.5127 (0.0010)	0.1137 (0.0008) / 0.1727 (0.0009)
$\alpha = 3$	0.0510 (0.0023) / 0.0504 (0.0019)	0.0565 (0.0022) / 0.0540 (0.0020)	0.0731 (0.0020) / 0.0775 (0.0022)	0.2799 (0.0003) / 0.5133 (0.0007)	0.1151 (0.0017) / 0.1738 (0.0015)
$\alpha = 5$	0.0477 (0.0008) / 0.0475 (0.0008)	0.0531 (0.0007) / 0.0507 (0.0008)	0.0699 (0.0006) / 0.0752 (0.0006)	0.2794 (0.0003) / 0.5114 (0.0009)	0.1125 (0.0005) / 0.1712 (0.0004)
$\alpha = 7$	0.0476 (0.0011) / 0.0467 (0.0009)	0.0536 (0.0010) / 0.0513 (0.0010)	0.0707 (0.0011) / 0.0765 (0.0012)	0.2809 (0.0007) / 0.5097 (0.0011)	0.1132 (0.0008) / 0.1711 (0.0008)
$\alpha = 9$	0.0449 (0.0006) / 0.0445 (0.0007)	0.0527 (0.0008) / 0.0514 (0.0010)	0.0709 (0.0006) / 0.0793 (0.0013)	0.2809 (0.0006) / 0.5095 (0.0008)	0.1123 (0.0005) / 0.1712 (0.0006)

Favorita	Level 0	Level 1	Level 2	Level 3	Mean
$\alpha = 1$	0.0581 (0.0015) / 0.0581 (0.0013)	0.0919 (0.0010) / 0.1828 (0.0030)	0.1252 (0.0008) / 0.2767 (0.0028)	0.2090 (0.0005) / 0.3256 (0.0037)	0.1211 (0.0008) / 0.2108 (0.0017)
$\alpha = 3$	0.0614 (0.0023) / 0.0609 (0.0022)	0.0919 (0.0016) / 0.1768 (0.0033)	0.1266 (0.0015) / 0.2675 (0.0038)	0.2109 (0.0010) / 0.3222 (0.0046)	0.1227 (0.0015) / 0.2069 (0.0025)
$\alpha = 5$	0.0606 (0.0028) / 0.0601 (0.0027)	0.0909 (0.0013) / 0.1742 (0.0016)	0.1257 (0.0012) / 0.2696 (0.0052)	0.2102 (0.0010) / 0.3224 (0.0054)	0.1219 (0.0015) / 0.2066 (0.0035)
$\alpha = 7$	0.0597 (0.0018) 0.0587 (0.0017)	0.0909 (0.0009) 0.1737 (0.0022)	0.1266 (0.0008) 0.2727 (0.0060)	0.2113 (0.0006) 0.3327 (0.0057)	0.1221 (0.0009) 0.2095 (0.0032)
$\alpha = 9$	0.0610 (0.0023) / 0.0597 (0.0023)	0.0908 (0.0011) / 0.1714 (0.0018)	0.1262 (0.0008) / 0.2694 (0.0037)	0.2114 (0.0007) / 0.3299 (0.0048)	0.1224 (0.0012) / 0.2076 (0.0018)

Tourism	Level 0	Level 1	Level 2	Level 3	Level 4	Mean
$\alpha = 1$	0.0636 (0.0030) / 0.0652 (0.0030)	0.1270 (0.0013) / 0.1642 (0.0027)	0.1735 (0.0011) / 0.2321 (0.0024)	0.2298 (0.0012) / 0.3860 (0.0048)	0.3499 (0.0008) / 0.7868 (0.0086)	0.1888 (0.0009) / 0.3269 (0.0026)
$\alpha = 3$	0.0583 (0.0030) / 0.0594 (0.0033)	0.1271 (0.0028) / 0.1628 (0.0056)	0.1720 (0.0019) / 0.2349 (0.0058)	0.2303 (0.0024) / 0.4017 (0.0125)	0.3593 (0.0043) / 0.8057 (0.0080)	0.1894 (0.0020) / 0.3329 (0.0047)
$\alpha = 5$	0.0502 (0.0027) / 0.0521 (0.0027)	0.1175 (0.0015) / 0.1454 (0.0019)	0.1653 (0.0012) / 0.2189 (0.0027)	0.2264 (0.0013) / 0.3911 (0.0073)	0.3574 (0.0026) / 0.7949 (0.0058)	0.1834 (0.0012) / 0.3205 (0.0019)
$\alpha = 7$	0.0504 (0.0032) / 0.0512 (0.0026)	0.1176 (0.0031) / 0.1520 (0.0070)	0.1695 (0.0033) / 0.2287 (0.0077)	0.2341 (0.0044) / 0.4138 (0.0159)	0.3718 (0.0051) / 0.8039 (0.0101)	0.1887 (0.0036) / 0.3299 (0.0076)
$\alpha = 9$	0.0534 (0.0027) / 0.0551 (0.0025)	0.1182 (0.0033) / 0.1508 (0.0021)	0.1699 (0.0015) / 0.2256 (0.0025)	0.2315 (0.0014) / 0.3958 (0.0060)	0.3765 (0.0017) / 0.8174 (0.0074)	0.1899 (0.0015) / 0.3290 (0.0019)

In this section, we consider a 2-level hierarchy of time-series, consisting of a single root node (indexed by 0) with L children (denoted by $\mathcal{L}(0)$). We will also assume that instead of learning the K basis vectors $\boldsymbol{B}$ from scratch, the K basis vectors are assumed to come from a much larger dictionary $\bar{\boldsymbol{B}} \in \mathbb{R}^{T \times D}$ of D ($\gg K$) vectors that is fixed and known to the model. While the original problem learns the basis and the coefficients $\boldsymbol{\theta}$ simultaneously, in this case the goal is to select the basis from among a larger dictionary, and learn the coefficients $\boldsymbol{\theta}$.

We analyze this problem, under the assumption that the parent embedding θ_0 is close to all the children embeddings θ_n. This assumption captures structural similarities of time series that are close to each other in the hierarchy. We show that incorporating the hierarchical constraints under such an assumption can result in a mean-square error at the leaf nodes that is a multiplicative factor L smaller than the optimal mean-square error of any model that does not use the hierarchical constraints. Our proposed HiRed model, when applied in this setting would result in the following (hierarchically) regularized regression problem:

$$\min_{\boldsymbol{\theta}} \frac{1}{NT} \|\boldsymbol{y} - \boldsymbol{B}\boldsymbol{\theta}\|_2^2 + \lambda \sum_{n \in \mathcal{L}(0)} \|\theta_0 - \theta_n\|_2^2. \tag{6.7}$$

For the sake of analysis, we instead consider a two-stage version, described in Algorithm 6.1 and Algorithm 6.2: we first recover the support of the basis using Basis Pursuit [42]. We then estimate the parameters of the root node, which is then plugged-in to solve for the parameters of the children node. We also define the baseline (unregularized) optimization

Algorithm 6.1 Basis Recovery

Input: Observed $\boldsymbol{y}$, basis dict $\bar{\boldsymbol{B}}$, regularization parameter λ_L
Output: Estimated basis $\boldsymbol{B}$
1: $\widehat{\alpha}_0 \leftarrow \underset{\alpha \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2T}\|y_0 - \bar{\boldsymbol{B}}\alpha\|_2^2 + \lambda_L \|\alpha\|_1$
2: Estimate support $\widehat{S} = \{i \mid |\widehat{\alpha}_0| > 0\}$
3: Estimate true basis $\boldsymbol{B} \leftarrow \bar{\boldsymbol{B}}_{\widehat{S}}$

Algorithm 6.2 Parameter Recovery

Input: Observed time series $\boldsymbol{y}$, estimated basis $\boldsymbol{B}$, regularization parameter λ_E
Output: Estimated parameters $\boldsymbol{\theta}$
1: $\widehat{\theta}_0 \leftarrow \operatorname{argmin}_{\theta_0} \frac{1}{T}\|y_0 - \boldsymbol{B}\theta_0\|_2^2$
2: **for** $n \in \mathcal{L}(0)$ **do**
3: $\quad \widehat{\theta}_n \leftarrow \operatorname{argmin}_{\theta_n} \frac{1}{T}\|y_n - \boldsymbol{B}\theta_n\|_2^2 + \lambda_E \|\widehat{\theta}_0 - \theta_n\|_2^2$.

problem for the leaf nodes that does not use any hierarchical information, as

$$\tilde{\theta}_n = \underset{\theta_n}{\operatorname{argmin}} \frac{1}{T}\|y_n - \boldsymbol{B}\theta_n\|_2^2 \quad \forall n \in \mathcal{L}(0). \tag{6.8}$$

The basis support recovery follows from standard analysis [273] detailed in Lemma 6.1 in the Appendix. We focus on the performance of Algorithm 6.2 here. The following theorem bounds the error of the unregularized ($\tilde{\theta}_n$) and the hierarchically-regularized ($\widehat{\theta}_n$, see Algorithm 6.2) optimization solutions. A proof of the theorem can be found in Section 6.7.2.

Theorem 6.1. *Suppose the rows of $\boldsymbol{B}$ are norm bounded as $\|\boldsymbol{B}_i\|_2 \leq r$, and $\|\theta_n - \theta_0\|_2 \leq \beta$. Define $\Sigma = \boldsymbol{B}^T\boldsymbol{B}/T$ as the empirical covariance matrix. For $\lambda_E = \frac{\sigma^2 K}{T\beta^2}$, $\widetilde{\theta}_n$ and $\widehat{\theta}_n$ can be bounded as,*

$$\mathbb{E}\|\widetilde{\theta}_n - \theta_n\|_\Sigma^2 \leq \frac{\sigma^2 K}{T}, \quad \mathbb{E}\|\widehat{\theta}_n - \theta_n\|_\Sigma^2 \leq 3\frac{\sigma^2 K}{T}\frac{1}{1 + \frac{\sigma^2 K}{Tr^2\beta^2}} + 6\frac{\sigma^2 K}{TL}. \tag{6.9}$$

In fixed design linear regression $\|\widehat{\theta}_n - \theta_n\|_\Sigma^2 = \|\boldsymbol{B}(\widehat{\theta}_n - \theta_n)\|^2$ is the population squared error (see Section 6.6.1 for a bound on the parameter estimation error). The gains due to the regularization can be understood by considering the case when β is upper bounded by a sufficiently small quantity. Note that an upper bound on β essentially implies that the children time-series have structural similarities as further elaborated in Section 6.6.1. We show that the above assumption yields a smaller upper bound on the error. In fact, if $\beta = o(\sqrt{K/T})$, then the numerator $1 + \frac{\sigma^2 K}{Tr^2\beta^2}$ in Eq. (6.9) is $\omega(1)$ resulting in $\mathbb{E}\|\widehat{\theta}_n - \theta_n\|_\Sigma^2 = o(\frac{\sigma^2 K}{T})$ which decays faster than $\frac{\sigma^2 K}{T}$. Furthermore, if β is even smaller as $\beta = O(\sqrt{K/LT})$, then following similar calculations, $\mathbb{E}\|\widehat{\theta}_n - \theta_n\|_\Sigma^2 = O(\frac{\sigma^2 K}{LT})$ which is again smaller than the unregularized bound.

6.6.1 Discussion

Upper bound $\|\theta_n - \theta_0\|_2 \leq \beta$: The upper bound β essentially bounds the distance between sibling leaf embeddings belonging to the same parent. This is directly related to

an upper bound on the distance between the parent embedding θ_0 and the leaf embeddings θ_n, as θ_n is essentially the mean of the leaf nodes (mean property). In many practical scenarios, the children time series of a parent may not have too different seasonal trends (for example power consumption of houses in the same neighborhood, or sales of items under the same category) resulting in the parent time series following similar trends as well.

Bounding $\|\theta_n - \theta_0\|_2$: In most theoretical analyses of linear regression [274], the main quantity of interest is the prediction error $\|\theta_n - \theta_0\|_\Sigma$ rather than the parameter estimation error $\|\theta_n - \theta_0\|_2$, as the former is directly related to the performance metric of the model. However, a bound on the parameter estimation error can be easily established using the property that $\|\theta_n - \theta_0\|_2 \leq \|\theta_n - \theta_0\|_\Sigma / \sqrt{C_{\min}}$, where $C_{\min}$ is the lower bound on the smallest eigenvalue of the sample covariance matrix as defined in Assumption 6.1 in Section 6.7.3.

6.7 Proofs

6.7.1 Support Recovery

Lemma 6.1. *Suppose $\boldsymbol{B}$ satisfies the lower eigenvalue condition (Assumption 6.1 in Section 6.7.3) with parameter $C_{\min}$ and the mutual incoherence condition (Assumption 6.2 in Section 6.7.3) with parameter γ. Also assume that the columns of the basis pool $\bar{\boldsymbol{B}}$ are normalized so that $\max_{j \in S^c} \|\bar{\boldsymbol{B}}^{(j)}\| \leq \sqrt{T}$, and the true parameter θ_0 of the root satisfies*

$$\|\theta_0\|_\infty \geq \lambda_L \left[\left\| \left\| \Sigma^{-1} \right\| \right\|_\infty + \frac{4\sigma}{\sqrt{LC_{\min}}} \right], \tag{6.10}$$

where $\|\|A\|\|_\infty = \max_i \sum_j |A_{ij}|$ denotes matrix operator ℓ_∞ norm, and $\Sigma = \boldsymbol{B}^T \boldsymbol{B}/T$ denotes the empirical covariance matrix. Then for $\lambda_L \geq \frac{2}{\gamma}\sqrt{\frac{2\sigma^2 \log d}{LT}}$, with probability least $1 - 4\exp(-c_1 T\lambda^2)$ (for some constant c_1), the support $\widehat{S} = \{i \mid |\widehat{\alpha}_0| > 0\}$ recovered from the Lasso solution (see Algorithm 6.1) is equal to the true support S.

Proof. We are given a pool of basis vectors $\bar{\boldsymbol{B}}$ from which the observed data is generated using a subset of K columns which we have denoted by $\boldsymbol{B}$ in the text. We denote the correct subset of columns by S and recover them from the observed data using basis pursuit - also known as the support recovery problem in the literature. Given the observed data and the pool of basis vectors $\bar{\boldsymbol{B}}$, we recover the support from the following regression problem corresponding to the root node time series.

$$y_0 = \bar{\boldsymbol{B}}\alpha + w_0, \quad w_0 \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}/L), \tag{6.11}$$

where α is K-sparse with the non-zero indices at S, and the non-zero values equal θ_0 - the true parameters of the root node. Here we have used the fact that the root node has a $1/L$ times smaller variance due to aggregation. The true support S can be recovered from the observed data y_0, by solving the sparse regression problem (Lasso) given in Algorithm 6.2. A number of standard Lasso assumptions are needed to ensure that the support is identifiable, and that the non-zero parameters are large enough to be estimated. Assuming that $\bar{\boldsymbol{B}}_S$ ($= \boldsymbol{B}$) and α satisfy all the assumptions of Theorem 6.2, the theorem ensures that the true support S is recovered with high probability. □

6.7.2 Proof of Theorem 6.1 - Error Bounds for Regularized Estimators

For this proof, we assume that the support S is recovered and the true basis functions $\boldsymbol{B}$ are known with high probability (see Section 6.7.1). We divide the proof into multiple steps.

Step I: By Corollary 6.1, the OLS estimate $\widehat{\theta}_0$ (see Algorithm 6.2) of parameters of the root node and the OLS estimate $\widetilde{\theta}_n$ (see Eq. (6.8)) can be bounded as,

$$\mathbb{E}[\|\widehat{\theta}_0 - \theta_0\|_\Sigma^2] \leq \frac{\sigma^2 K}{TL}, \quad \mathbb{E}[\|\widetilde{\theta}_n - \theta_n\|_\Sigma^2] \leq \frac{\sigma^2 K}{T} \quad \forall n \in \mathcal{L}(0). \tag{6.12}$$

Step II: Next, using change of variables, we notice that the ridge regression loss for the child nodes (see Algorithm 6.2) is equivalent to the following.

$$\widehat{\psi}_n = \underset{\psi_n}{\operatorname{argmin}} \frac{1}{T}\|y_n - \boldsymbol{B}\widehat{\theta}_0 - \boldsymbol{B}\psi_n\|_2^2 + \lambda\|\psi_n\|_2^2 \quad \forall n \in \mathcal{L}(0), \tag{6.13}$$

where $\psi_n = \theta_n - \widehat{\theta}_0$. The final estimate for the child parameters can be written as a sum of the ψ_n estimate and the root node estimate, $\widehat{\theta}_n = \widehat{\psi}_n + \widehat{\theta}_0$. We also consider a related problem that will help us in computing the errors bounds.

$$\widetilde{\psi}_n = \underset{\psi_n}{\operatorname{argmin}} \frac{1}{T}\|y_n - \boldsymbol{B}\theta_0 - \boldsymbol{B}\psi_n\|_2^2 + \lambda\|\psi_n\|_2^2 \quad \forall n \in \mathcal{L}(0). \tag{6.14}$$

Here we have replaced $\widehat{\theta}_0$ with the true value θ_0. Note that this regression problem cannot be solved in practice since we do not have access to the true value of θ_0. We will only use it to assist in the analysis. Now we will bound the difference between the estimates $\widehat{\psi}_n$ and $\widetilde{\psi}_n$. The closed form solution for ridge regression is well known in the literature.

$$\begin{aligned}
\widehat{\psi}_n &= T^{-1}(\Sigma + \lambda \boldsymbol{I})^{-1}\boldsymbol{B}^T(y_n - \boldsymbol{B}\widehat{\theta}_0) \\
\widetilde{\psi}_n &= T^{-1}(\Sigma + \lambda \boldsymbol{I})^{-1}\boldsymbol{B}^T(y_n - \boldsymbol{B}\theta_0),
\end{aligned}$$

where $\Sigma = \boldsymbol{B}^T\boldsymbol{B}/T$, as defined earlier. The norm of the difference of the estimates can be bounded as

$$\begin{aligned}
\widehat{\psi}_n - \widetilde{\psi}_n &= T^{-1}(\Sigma + \lambda \boldsymbol{I})^{-1}\boldsymbol{B}^T\boldsymbol{B}(\widetilde{\theta}_0 - \widehat{\theta}_0) \\
\implies \|\widehat{\psi}_n - \widetilde{\psi}_n\|_\Sigma^2 &= (\widetilde{\theta}_0 - \widehat{\theta}_0)^T\Sigma(\Sigma + \lambda \boldsymbol{I})^{-1}\Sigma(\Sigma + \lambda \boldsymbol{I})^{-1}\Sigma(\widetilde{\theta}_0 - \widehat{\theta}_0) \\
&= (\widetilde{\theta}_0 - \widehat{\theta}_0)^T\Sigma(\Sigma + \lambda \boldsymbol{I})^{-1}\Sigma(\Sigma + \lambda \boldsymbol{I})^{-1}\Sigma(\widetilde{\theta}_0 - \widehat{\theta}_0) \\
&= (\widetilde{\theta}_0 - \widehat{\theta}_0)^T V D\left[\frac{\lambda_i^3}{(\lambda_i + \lambda)^2}\right] V^T(\widetilde{\theta}_0 - \widehat{\theta}_0).
\end{aligned}$$

Here we have used an eigen-decomposition of the symmetric sample covariance matrix as $\Sigma = VD[\lambda_i]V^T$. We use the notation $D[\lambda_i]$ to denote a diagonal matrix with values λ_i on the diagonal. The above can be further upper bounded using the fact that $\lambda_i \leq \lambda_i + \lambda$.

$$\|\widehat{\psi}_n - \widetilde{\psi}_n\|_\Sigma^2 \leq (\widetilde{\theta}_0 - \widehat{\theta}_0)^T V D[\lambda_i] V^T(\widetilde{\theta}_0 - \widehat{\theta}_0) = \|\widetilde{\theta}_0 - \widehat{\theta}_0\|_\Sigma^2. \tag{6.15}$$

Step III: Now we will bound the error on $\widetilde{\psi}_n$ and finally use it in the next step with triangle inequality to prove our result. Note that $y_n - \boldsymbol{B}\theta_0 = \boldsymbol{B}(\theta_n - \theta_0) + w_n$. Therefore, we can see from Eq. (6.14) that $\widetilde{\psi}_n$ is an estimate for $\theta_n - \theta_0$. Using the fact that $\|\theta_n - \theta_0\|_2 \leq \beta$ and Corollary 6.1, $\widetilde{\psi}_n$ can be bounded as,

$$\mathbb{E}[\|\widetilde{\psi} - (\theta_n - \theta_0)\|_\Sigma^2] \leq \frac{r^2\beta^2\sigma^2 K}{Tr^2\beta^2 + \sigma^2 K}. \tag{6.16}$$

Finally using triangle inequality, we bound the error of our estimate $\widehat{\theta}_n$.

$$\begin{aligned}
\|\widehat{\theta}_n - \theta_n\|_\Sigma^2 &= \|\widehat{\psi}_n + \widehat{\theta}_0 - \theta_n\|_\Sigma^2 \quad \text{(Using the decomposition from Step II).} \\
&\leq 3\left(\|\widehat{\psi}_n - \widetilde{\psi}_n\|_\Sigma^2 + \|\widetilde{\psi}_n - (\theta_n - \theta_0)\|_\Sigma^2 + \|\widehat{\theta}_0 - \theta_0\|_\Sigma^2\right) \\
&\quad \text{(Using triangle and Cauchy-Schwartz inequality)} \\
&\leq 3\left(\|\widetilde{\psi}_n - (\theta_n - \theta_0)\|_\Sigma^2 + 2\|\widehat{\theta}_0 - \theta_0\|_\Sigma^2\right) \quad \text{(Using Eq. (6.15)).}
\end{aligned}$$

Taking the expectation of the both sides, and using Eqs. (6.12) and (6.16), we get the desired result.

$$\mathbb{E}\|\widehat{\theta}_n - \theta_n\|_\Sigma^2 \leq 3\frac{r^2\beta^2\sigma^2 K}{Tr^2\beta^2 + \sigma^2 K} + 6\frac{\sigma^2 K}{TL}.$$

6.7.3 Review of Sparse Linear Regression

We consider the following sparse recovery problem. We are given data $(\boldsymbol{X}, y) \in \mathbb{R}^{n\times d} \times \mathbb{R}^n$ following the observation model $y = \boldsymbol{X}\theta^* + w$, where $w \sim \mathcal{N}(\boldsymbol{0}, \sigma^2\boldsymbol{I})$, and θ^* is supported in the indices indexed by a set S (S-sparse). We estimate θ^* using the following Lagrangian Lasso program,

$$\widehat{\theta} \in \underset{\theta\in\mathbb{R}^n}{\operatorname{argmin}} \left\{ \frac{1}{2n}\|y - \boldsymbol{X}\theta\|_2^2 + \lambda_n\|\theta\|_1 \right\} \tag{6.17}$$

We consider the fixed design setting, where the matrix $\boldsymbol{X}$ is fixed and not sampled randomly. Following [273], we make the following assumptions required for recovery of the true support S of θ^*.

Assumption 6.1 (Lower eigenvalue)**.** *The smallest eigenvalue of the sample covariance sub-matrix indexed by S is bounded below:*

$$\Lambda_{\min}\left(\frac{\boldsymbol{X}_S^T\boldsymbol{X}_S}{n}\right) \geq C_{\min} > 0 \tag{6.18}$$

Assumption 6.2 (Mutual incoherence)**.** *There exists some $\gamma \in (0, 1]$ such that*

$$\left|\!\left|\!\left| \boldsymbol{X}_{S^c}^T\boldsymbol{X}_S(\boldsymbol{X}_S^T\boldsymbol{X}_S)^{-1} \right|\!\right|\!\right|_\infty \leq 1 - \gamma, \tag{6.19}$$

where $|\!|\!|A|\!|\!|_\infty = \max_i \sum_j |A_{ij}|$ denotes matrix operator ℓ_∞ norm.

Theorem 6.2 (Support Recovery, Wainwright [273]). *Suppose the design matrix satisfies Assumptions 6.1 and 6.2. Also assume that the design matrix has its n-dimensional columns normalized so that* $\max_{j \in S^c} \|X_j\|_2 \leq \sqrt{n}$. *Then for* λ_n *satisfying,*

$$\lambda_n \geq \frac{2}{\gamma} \sqrt{\frac{2\sigma^2 \log d}{n}}, \tag{6.20}$$

the Lasso solution $\widehat{\theta}$ *satisfies the following properties with a probability of at least* $1 - 4\exp(-c_1 n \lambda_n^2)$*:*

1. *The Lasso has a unique optimal solution* $\widehat{\theta}$ *with its support contained within the true support* $S(\widehat{\theta}) \subseteq S(\theta^*)$ *and satisfies the* ℓ_∞ *bound*

$$\|\widehat{\theta}_S - \theta^*_S\|_\infty \leq \underbrace{\lambda_n \left[\left\| \left(\frac{\boldsymbol{X}_S^T \boldsymbol{X}_S}{n} \right)^{-1} \right\|_\infty + \frac{4\sigma}{\sqrt{C_{\min}}} \right]}_{g(\lambda_n)}, \tag{6.21}$$

2. *If in addition, the minimum value of the regression vector* θ^* *is lower bounded by* $g(\lambda_n)$*, then it recovers the exact support.*

6.7.4 Review of Ridge Regression

In this section we review the relevant background from [95] on fixed design ridge regression. As usual, we assume data $(\boldsymbol{X}, y) \in \mathbb{R}^{n \times d} \times \mathbb{R}^n$ following the observation model $y = \boldsymbol{X}\theta^* + w$, where $w \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$. Define the *ridge estimator* $\widehat{\theta}$ as the minimizer of the ℓ_2 regularized mean squared error,

$$\widehat{\theta} \in \underset{\theta \in \mathbb{R}^n}{\operatorname{argmin}} \left\{ \frac{1}{n} \|y - \boldsymbol{X}\theta\|_2^2 + \lambda \|\theta\|_2^2 \right\} \tag{6.22}$$

We denote the sample covariance matrix by $\Sigma = \boldsymbol{X}^T \boldsymbol{X}/n$. Then for any parameter θ, the expected ℓ_2 prediction error is given by, $\|\theta - \theta^*\|_\Sigma^2 = \|\boldsymbol{X}(\theta - \theta^*)\|_2^2/n$. We also assume the standard ridge regression setting of bounded $\|\theta^*\|_2 \leq B$. We have the following proposition from Hsu et al. [95] on expected error bounds for ridge regression.

Proposition 6.1 (Hsu et al. [95]). *For any regularization parameter* $\lambda > 0$*, the expected prediction loss can be upper bounded as*

$$\mathbb{E}[\|\widehat{\theta} - \theta^*\|_\Sigma^2] \leq \sum_j \frac{\lambda_j}{(\lambda_j/\lambda + 1)^2} \theta_j^{*2} + \frac{\sigma^2}{n} \sum_j \left(\frac{\lambda_j}{\lambda_j + \lambda} \right)^2, \tag{6.23}$$

where λ_i *denote the eigenvalues of the empirical covariance matrix* Σ*.*

Using the fact that $\lambda_j \leq \mathbf{tr}(\Sigma)$, and $x/(x+c)$ is increasing in x for $x \geq 0$, the above bound can be simplified as,

$$\begin{aligned} \mathbb{E}[\|\widehat{\theta} - \theta^*\|_\Sigma^2] &\leq \frac{\mathbf{tr}(\Sigma)}{(\mathbf{tr}(\Sigma)/\lambda + 1)^2} \|\theta^*\|^2 + \frac{\sigma^2 d}{n} \left(\frac{\mathbf{tr}(\Sigma)}{\mathbf{tr}(\Sigma) + \lambda} \right)^2 \\ &\leq \frac{\mathbf{tr}(\Sigma) B^2 \lambda^2 + \mathbf{tr}(\Sigma)^2 \sigma^2 d/n}{(\mathbf{tr}(\Sigma) + \lambda)^2} \end{aligned}$$

Assuming that the covariate vectors $\boldsymbol{X}_i$ are norm bounded as $\|\boldsymbol{X}_i\|_2 \leq r$, and using the fact that $\mathbf{tr}(\Sigma) \leq r^2$, gives us the following corollary.

Corollary 6.1. *When choosing $\lambda = \frac{\sigma^2 d}{nB^2}$, the prediction loss can be upper bounded as,*

$$\mathbb{E}[\|\widehat{\theta} - \theta^*\|_{\Sigma}^2] \leq \frac{r^2 B^2 \sigma^2 d}{nr^2 B^2 + \sigma^2 d}. \tag{6.24}$$

The usual ordinary least squares bound of $\frac{\sigma^2 d}{n}$ can be derived when considering the limit $B \to \infty$, corresponding to $\lambda = 0$.

6.8 Further Experimental Details

6.8.1 Training Details

Table 6.5 presents all the hyperparameters used in our proposed model. All models were trained via SGD using the Adam optimizer [126], and the training data was standardized to mean zero and unit variance. The datasets were split into train, validation, and test sets, the sizes of which are given in the Table 6.5. We used learning rate decay and early stopping using the Mean WAPE score on the validation set, with a patience of 10 for all models. We tuned the model hyper-parameters using the same metric. The various model hyper-parameters are given in Table 6.5.

All our experiments were implemented in Tensorflow 2, and run on a Titan Xp GPU with 12GB of memory. The computing server we used, had 256GB of memory, and 32 CPU cores, however, our code did not seem to use more than 10GB of memory and 4 CPU cores.

Mini-Batching: During each training iteration, we sample a minibatch of time series for computing the gradients. For constructing a minibatch, first a time window (of length $H + F$) is uniformly randomly selected from the training time steps, after which a subset of nodes is sampled from the hierarchy tree. The time series data corresponding to this subset of nodes and the sampled time window constitutes our minibatch.

Predicting for new time steps: Once trained, our model can be used to predict for new datapoints without any retraining. However, in practice it may be beneficial to retrain the model with newer data to improve performance, even though this is not a constraint for our proposed approach.

Knowledge of the hierarchy: Our proposed approach requires the hierarchy to be known during training to be able to regularize the embeddings. In scenarios where new aggregations are presented during test time, a reasonable prediction can be produced using the following strategy: the embedding of the new aggregation is set to the mean of the embeddings of nodes in that aggregation - the rest of the model remains the same. Predictions for unseen aggregations are made in the same way as with seen aggregations. This idea is beyond the scope of this thesis, and hence left for future exploration.

Table 6.5: Final model hyperparameters for various datasets tuned using the Mean WAPE metric on the validation set.

Model hyperparameters	M5	Favorita	Tourism
LSTM hidden dim	42	24	14
Embedding dim K	8	8	6
NMF rank R	12	4	6
Multi-Horizon decoder hidden dim	24	16	12
Embedding regularization λ_E	3.4e-6	4.644e-4	7.2498e-8
History length H and forecast horizon F	(28, 7)	(28, 7)	(24, 4)
No. of rolling val/test windows	5	5	3
Initial learning rate	0.004	0.002	0.07
Decay rate and decay interval	(0.5, 6)	(0.5, 6)	(0.5, 6)
Early stopping patience	10	10	10
Training epochs	40	40	40
Batch size	512	512	512
Total #params	80k	80k	8k

6.8.2 Further details about global state Z

In many practical scenarios, the evolving global state of the set of time series may not be captured by the global covariates $\boldsymbol{X}$ only. For instance, when there is an overall increase/decrease in sales across all time series, it is captured in the past values $\boldsymbol{Y}$ rather than $\boldsymbol{X}$. As a result, it may be required to feed in past values of time series to the BD model. However, since we cannot feed in the whole set of time series (order of 1000s) without leading to scalability issues, we choose a small set of representative time series using convex Non-negative Matrix Factorization, thus approximating the global state. NMF assumes that the columns of the time series matrix $\boldsymbol{Y} \in \mathbb{R}^{T \times N}$ lies in the convex set spanned by a small *subset* of columns, that is $\boldsymbol{Y} = \boldsymbol{Z}\boldsymbol{W} + \epsilon$, where the columns of $\boldsymbol{Z} \in \mathbb{R}^{T \times R}$ are a subset of the columns of $\boldsymbol{Y}$, and $\boldsymbol{W} \in \mathbb{R}_{+}^{K \times N}$ denote weights of the convex combination. Since this assumption may not hold exactly for most datasets, ϵ is used to account for the modelling error resulting in an approximate factorization. We use $\boldsymbol{Z}$ as an approximation to the global state, and compute it using the Algorithm of Gillis and Vavasis [69]. We would also like to emphasize that our prediction model only sees the past values $\boldsymbol{Z}_{\mathcal{H}}$ as input since the future time series values are unknown.

While NMF is one of the choices for $\boldsymbol{Z}$, it is definitely not the only choice. Another option, PCA (or equivalently SVD), may lead to latent vectors which do not have any temporal dependencies thus requiring additional temporal regularization [239, 286]. One may also use more sophisticated methods such as Temporal Latent Auto-Encoders [189]. We leave this idea for future exploration.

Part III

Conclusion

7 | Conclusion and Epilogue

Optimization in general is a significant step in many scientific and engineering fields. From continuous optimization used for training neural networks [228] to discrete optimization for chip placement [174], optimization problems occur in a wide variety flavors and settings. Efficient solutions to such optimization problems have the potential to positively impact a wide range of fields. In this thesis, we studied optimization problems broadly falling into two categories: black-box optimization and multi-objective optimization. We proposed solutions to a number of problems in each of the categories, and also demonstrated their effectiveness on real world problems.

We summarize our a general summary, limitations, and possible future directions about each of the topics below.

Black-box Optimization. As emphasized earlier, black-box optimization occurs in many real world problems, from hyper-parameter optimization of machine learning algorithms [246] to design of novel materials [49]. Gaussian process based Bayesian optimization is often the method of choice, when it comes to optimization of low-dimensional black box functions. Several Bayesian optimization approaches have been studied in the literature, with expected improvement [112] being the most popular due to its superior performance and lack of any hyper-parameters. A lot of the techniques in BO are inspired from the bandit literature [229, 251], and depend on creating the right balance between exploration and exploitation.

Since, GP based approaches do not scale to high dimensional or structured data, high dimensional approaches often use models based on decision trees [98] or neural networks [220], and are predominantly greedy approaches due to the lack of efficient methods for uncertainty quantification in high dimensions. Recent theoretical works on high-dimensional bandits [85, 150] have shown that in the case of sparse linear bandits, greedy approaches are often optimal in the the data poor regime. The sample complexity typically grows exponentially with the dimension of the input space for problems without any structure. However, work on sparse linear bandits indicate that greedy approaches may in fact be optimal for certain high-dimensional and structured problems. The data rich setting lies on the other end of the spectrum, where high-dimensional models can shine because of the abundance of data. In such cases, exploration does not lead to much improvement beyond just exploitation (a.k.a. the greedy approach).

Black-box problems also vary in the core problem setting. Multi-objective black box functions produce multiple output values when queried. Another setting, multi-fidelity opti-

mization allows for function queries at lower fidelities (returns an approximation of the true value), at a lower cost. In this thesis, we study multi-objective and cost-aware variants of black-box optimization. While the cost-aware setting is related to the multi-fidelity setting, they differ significantly as will be discussed below.

In Chapter 2, we proposed a flexible approach to multi-objective Bayesian optimization for efficient exploration of desired regions of the Pareto front. We show that our algorithm can successfully sample from a specified region of the Pareto front as is required in many applications, but is still flexible enough to sample from the whole Pareto front. Compared to several other approaches [17, 89, 207], our algorithm is conceptually much simpler, computationally cheap and scales linearly with the number of objectives. Our approach also lends itself to a notion of regret in the MO setting that also captures user preferences; with the regret being high if not sampled from the specified region or sampled outside of it. We provided a theoretical proof of the fact that our algorithm achieves a zero regret in the limit under necessary regularity assumptions. Finally, we performed experiments with synthetic and real problems, and showed that our approach leads to a smaller or comparable multi-objective regret compared to the baselines. An implementation can be found in Dragonfly[1], a publicly available python library for scalable Bayesian optimization [119].

A limitation of the above approach is that user preferences are often not explicitly known (even by the user), and often need to be elicited [76, 227, 303]. While we use a simple heuristic to specify the sampling distribution for our experiments, in practice, human feedback will be required to guide the algorithm to sample from the desired region of the Pareto front.

In Chapter 4, we study black-box optimization with varying evaluation costs. While this setup is similar in flavor to multi-fidelity optimization [115], it is different in the sense that, in the latter the function can be evaluated at the same point at multiple fidelities (with differing costs). Where as in the former, the fidelity remains the same and the cost varies. We proposed a simple approach to cost aware Bayesian optimization based on Information Directed Sampling [229], and theoretically showed that our algorithm is provably no-regret. We also showed promising results on a synthetic function. We leave further experiments on real functions to future work.

A limitation of the above approach is that, in the case where the cost of an evaluation is arbitrarily close to zero, the algorithm will evaluate that point indefinitely as the information gain from that point is quite large compared to the cost of evaluation. A potential way of solving this problem is using a similar technique as Astudillo et al. [9]. More specifically, instead of balancing the regret incurred with the information gained, a modified goal could be to select a batch of candidates and balance the regret incurred with the information gained from all of them, while ensuring that the total cost of evaluation is smaller than the total allowed budget for the batch.

In Chapter 3, we study the effectiveness of greedy approaches for black-box optimization. We presented two simple neural-greedy algorithms for global optimization of expensive BB functions. Unlike prior works, our algorithms do not explicitly construct confidence intervals or perform posterior sampling. We propose a simple approach that greedily fits a surrogate model to the observed data and use the learned model as the acquisition function

[1]Dragonfly, `https://github.com/dragonfly/dragonfly`

to determine the next query point. Our main approach SimpleSMB added random perturbations to the target variable before training to account for noise in the observations, whereas our other approach, PosteriorSMB added random perturbations to both the target variable and the output.

We derived theoretical regret bounds by using recently studied connections between GPs and training dynamics of wide neural networks. In particular, we showed that PosteriorSMB implicitly performs Thompson sampling, and rely on this connection to derive a bound on its regret. Empirically, we showed that our algorithms have better performance than other neural baselines and achieve similar performance as GP-EI, which is the gold standard for blackbox optimization.

In our experiments, SimpleSMB achieved slightly better performance than PosteriorSMB. This is worth investigating further SimpleSMB doesn't perform Thompson sampling like PosteriorSMB. In the future, we aim to derive regret bounds for SimpleSMB and understand the settings in which it achieves better performance than PosteriorSMB. There have been criticisms of the NTK regime as not capturing the behavior of regular neural networks [67]. An interesting future direction would be to study our algorithms in the non-NTK regime and derive their regret bounds. Another important problem is to propose an approach for automatically tuning γ, the initialization variance of the neural network parameters. We noticed a significant impact of γ on the performance of our algorithm. An important future direction is to rely on the connection between our algorithms and GPs to automatically set this hyper-parameter using maximum likelihood estimation.

The core idea behind our method was to inflate the initialization variances of the neural network model. While this works for shallow neural networks as in our experiments, for deeper neural networks, this may cause significant problems due to exploding gradients during training. Instead of inflating the variance of all the layers, one may consider inflating the variance of only the last few layers. More extensive experiments have to be performed in order to make any conclusions about best practices for neural network BBO.

Applications: Scaling black-box optimization algorithms to higher dimensions can lead to a significant impact on a variety of applications from scientific discovery to large scale recommender systems, requiring methods that scales to tens of thousands of observations. Examples include molecular discovery [49, 196, 297], and controllers for nuclear fusion [39, 170].

Multi-Objective Optimization. As discussed earlier, multi-objective optimization problems appear frequently in many machine learning applications such as multi-task learning [240], imbalanced classification [41, 87], balancing competing objectives in public policy [225], and neural architecture search [57]. The standard goal in MOO problems is to optimize several competing objectives. Since they cannot be optimized simultaneously, instead of looking for an optimum, the goal is to look for a set of Pareto optimal solutions. Each Pareto optimal solution is essentially an optimal trade-off between the objectives.

Multi-objective optimization problems can vary widely in their characteristics. As discussed above, MOO problems can also be black-boxes, several aspects of which are studied in Chapter 2. The second part of this thesis is about strategies for differentiable MOO. A standard solution to differentiable MOO is to linearly weight the objectives and optimize

the resulting scalar. However, as we see in Chapter 5, directly applying weight multipliers to the objectives does not always lead to the most Pareto optimal solution. We showed that an annealing schedule to increase the weights slowly works well in practice.

As discussed in Section 1.1.2, MOO problems can also appear in the form of constrained optimization problems. Although in many cases constrained optimization problems can be written in the form of a Lagrangian, the parameters of the Lagrangian cannot usually be computed in closed form. One way of setting the parameters automatically is to perform alternating gradient ascent-descent to solve the min max problem,

$$\max_{x} \min_{\lambda \geq 0} f(x) + \lambda^T (c - g(x)) \quad \text{(see Eq. (1.6) for notation)}. \tag{7.1}$$

Alternating gradient ascent descent on the above is also known as the differential method of multipliers. It ensures that the weights of the constraint functions increase when the constraints are violated. However, as beautifully summarized in a blog[2] the most basic version of the algorithm may not converge because of oscillations. A modified differential method of multipliers [206] which includes a dampening strategy leads to a more robust convergence. The modified method of multipliers combined with an annealing schedule can be useful in many practical scenarios, including the problem of constrained language generation [139, 140]. Next we summarize the applications of differentiable MOO that we studied in this thesis.

In Chapter 5, we proposed a novel approach to learn high dimensional embeddings with the goal of improving efficiency of retrieval tasks. We worked with two objectives, one the retrieval accuracy, and the other the retrieval speed. Our approach integrates the FLOPs incurred during retrieval into the loss function and optimizes it directly through a continuous relaxation. We provide further insight into our approach by showing that the proposed approach favors an even distribution of the non-zero activations across all the dimensions. In order for our method to work in practice, we proposed a regularization annealing schedule in order to ensure that the loss does not get stuck in a local minima. Overall we were able to show that sparse embeddings can be around 50× faster compared to dense embeddings without a significant loss of accuracy[3].

The above work also raises several questions about the effectiveness of sparse embeddings. Several works have motivated the benefit of high-dimensional and sparse embeddings for information retrieval [48, 108]. In particular, Dasgupta et al. [48] show that fruit flies effectively make use of wide and sparse representations to identify odors, essential for their survival. In a way, this suggests that even from a evolutionary standpoint high dimensional sparse embeddings are optimal. High-dimensionality retains the representational capacity, where as sparsity helps in improving the retrieval speed.

Learning binary code representations [92] are another popular way of learning sparse representations for data. One possible hypothesis is that binary codes are able to effectively encode the contents of the data by encoding the various independent features present in the data in different dimensions of the embedding. As a result the retrieval problem is reduced to computing the number of common features (equivalent to a set intersection) between the query embedding and the database embeddings. We use the term *features* here loosely

[2] `https://www.engraved.blog/how-we-can-make-machine-learning-algorithms-tunable/`

[3] The sparse embeddings code is available at `https://github.com/biswajitsc/sparse-embed`

to denote the active bits of the binary embedding. The hypothesis is that each active bit encodes the presence of a feature in the original data. Our proposed approach, on the other hand, learns real valued sparse embeddings, and generalizes the above set intersection to a dot product. This idea is quite reminiscent of TF-IDF [113, 211], where each feature is weighted according to their degree of occurrence in the data. An interesting future direction is to interpret the sparse features with the goal of understanding what each of the dimensions correspond to.

In Chapter 6, we proposed a method for hierarchical time series forecasting. Our model consists of two components, the time-varying auto-regressive (TVAR) model, and the basis decomposition (BD) model. The TVAR model is coherent by design, whereas we regularize the BD model to impose approximate coherency. Our model is fully differentiable and is trainable via SGD, while also being scalable with respect to the number of nodes in the hierarchy. Furthermore, it also does not require any additional pre-processing steps.

We empirically evaluated our method on three benchmark datasets and showed that our model consistently improved over state of the art baselines for most levels of the hierarchy. We perform an ablation study to justify the important components of our model and also show empirically that our forecasts are more coherent than the RNN baseline. We also consider the multi-objective aspect of the level-wise metrics, by up-sampling the higher level nodes.

Our proposed model optimizes the training loss for all the hierarchical levels simultaneously. Another class of methods known as *top-down* approaches operate in a layer-wise fashion. They start by learning a model for the top-most (root) level and for each level of the hierarchy tree, they make use of the predictions from the upper level to make predictions for the current level. In a follow up work [47], we propose a top-down approach for probabilistic coherent hierarchical forecasting. Our currently proposed model in this thesis, produces point estimates only, in contrast to probabilistic predictions which predict a distribution of values. An interesting direction would be to extend our current model to probabilistic forecasting.

General Takeaways. At a high level, in this thesis we aimed at providing a range of simple but practical strategies that can be a part of the scientists' or engineers' optimization toolbox. Even though our methods are simple, they are often more effective than other more sophisticated approaches which require a lot more hyper-parameters. We also provide theoretical analysis for most of our contributions, thus showing that simple and practical approaches can also be theoretically sound. Overall, progress in optimization techniques, both black box and differential, can have a large impact on many other scientific and engineering domains.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016. 5.5, 6.5

[2] Alekh Agarwal, Dean P Foster, Daniel J Hsu, Sham M Kakade, and Alexander Rakhlin. Stochastic convex optimization with bandit feedback. In *Advances in Neural Information Processing Systems*, pages 1035–1043, 2011. 3.1

[3] Rajeev Agrawal. Sample mean based index policies by o (log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995. 3

[4] Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International conference on machine learning*, pages 127–135. PMLR, 2013. 3.1

[5] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. *NeurIPS*, 2015. 2.4, 5.2

[6] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. PMLR, 2019. 3.1

[7] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *Advances in Neural Information Processing Systems*, 32, 2019. 3.1, 3.2, 3.4.1

[8] Devansh Arpit, Yingbo Zhou, Hung Ngo, and Venu Govindaraju. Why regularized auto-encoders learn sparse representation? *arXiv preprint arXiv:1505.05561*, 2015. 5.2

[9] Raul Astudillo, Daniel Jiang, Maximilian Balandat, Eytan Bakshy, and Peter Frazier. Multi-step budgeted bayesian optimization with unknown evaluation costs. *Advances in Neural Information Processing Systems*, 34, 2021. 2, 7

[10] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 2002. 2.1

[11] Alex Auvolat, Sarath Chandar, Pascal Vincent, Hugo Larochelle, and Yoshua Bengio.

Clustering is efficient for approximate maximum inner product search. *arXiv preprint arXiv:1507.05910*, 2015. 5.2

[12] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013. 5.2

[13] Francis Bach and Vianney Perchet. Highly-smooth zero-th order online optimization. In *Conference on Learning Theory*, pages 257–283. PMLR, 2016. 3.1

[14] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993. 1

[15] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17804–17815. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/ce1aad92b939420fc17005e5461e6f48-Paper.pdf`. 6.1

[16] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. Revisiting the inverted indices for billion-scale approximate nearest neighbors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 202–216, 2018. 5.2

[17] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Max-value entropy search for multi-objective bayesian optimization with constraints. *arXiv preprint arXiv:2009.01721*, 2020. 7

[18] Alexandre Belloni, Tengyuan Liang, Hariharan Narayanan, and Alexander Rakhlin. Escaping the local minima via simulated annealing: Optimization of approximately convex functions, 2015. 3.1

[19] Souhaib Ben Taieb and Bonsoo Koo. Regularized regression for hierarchical forecasting without unbiasedness conditions. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1337–1347, 2019. 6.1, 8

[20] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Bernie Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, et al. Neural forecasting: Introduction and literature overview. *arXiv preprint arXiv:2004.10240*, 2020. 6.1

[21] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002. 5.2

[22] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2464–2473, 2020. 6.1

[23] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017. 6.4

[24] Joos-Hendrik Böse, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Dustin Lange, David Salinas, Sebastian Schelter, Matthias Seeger, and Yuyang Wang. Probabilistic demand forecasting at scale. *Proceedings of the VLDB Endowment*, 10(12): 1694–1705, 2017. 6.1

[25] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 1.1.2, 5.5

[26] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`. 3.7.1

[27] Jürgen Branke. Consideration of partial user preferences in evolutionary multiobjective optimization. In *Multiobjective optimization*. Springer, 2008. 2.1

[28] Jürgen Branke and Kalyanmoy Deb. Integrating user preferences into evolutionary multi-objective optimization. In *Knowledge incorporation in evolutionary computation*. Springer, 2005. 2.1

[29] Sébastien Bubeck, Gilles Stoltz, Csaba Szepesvári, and Rémi Munos. Online optimization in x-armed bandits. In *Advances in Neural Information Processing Systems*, pages 201–208, 2009. 1, 3.1

[30] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 2012. 2.2.2, 2.3, 4.1

[31] Daniele Calandriello, Luigi Carratino, Alessandro Lazaric, Michal Valko, and Lorenzo Rosasco. Near-linear time gaussian process optimization with adaptive batching and resparsification. In *International Conference on Machine Learning*, pages 1295–1305. PMLR, 2020. 3.1

[32] Paolo Campigotto, Andrea Passerini, and Roberto Battiti. Active learning of Pareto fronts. *IEEE transactions on neural networks and learning systems*, 2014. 2.1

[33] Guillermo Canas and Lorenzo Rosasco. Learning probability measures with respect to optimal transport metrics. In *Advances in Neural Information Processing Systems*, 2012. 2.5.3

[34] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Spectral temporal graph neural network for multivariate time-series forecasting. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17766–17778. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/cdf6581cb7aca4b7e19ef136c6e601a5-Paper.pdf`. 6.1

[35] Yuan Cao, Zhiying Fang, Yue Wu, Ding-Xuan Zhou, and Quanquan Gu. Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198*, 2019. 3.4.1

[36] Yue Cao, Mingsheng Long, Jianmin Wang, Han Zhu, and Qingfu Wen. Deep quantization network for efficient image retrieval. *AAAI*, 2016. 5.2

[37] Yue Cao, Mingsheng Long, Bin Liu, and Jianmin Wang. Deep cauchy hashing for hamming space retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1229–1237, 2018. 5.1

[38] Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical Science*, pages 273–304, 1995. 1

[39] Ian Char, Youngseog Chung, Willie Neiswanger, Kirthevasan Kandasamy, Andrew O Nelson, Mark Boyer, Egemen Kolemen, and Jeff Schneider. Offline contextual bayesian optimization. *Advances in Neural Information Processing Systems*, 32, 2019. 7

[40] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002. 5.1

[41] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002. 7

[42] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001. 6.6

[43] Sheng Chen, Yang Liu, Xiang Gao, and Zhen Han. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. *arXiv preprint arXiv:1804.07573*, 2018. 5.5

[44] Sayak Ray Chowdhury and Aditya Gopalan. On kernelized multi-armed bandits. In *International Conference on Machine Learning*, pages 844–853. PMLR, 2017. 3.1, 3.4.1, 3.6.3

[45] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214. JMLR Workshop and Conference Proceedings, 2011. 3

[46] Ying Cui, Ruofei Zhang, Wei Li, and Jianchang Mao. Bid landscape forecasting in online ad exchange marketplace. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 265–273, 2011. 6.1

[47] Abhimanyu Das, Weihao Kong, Biswajit Paria, and Rajat Sen. A top-down approach to hierarchically coherent probabilistic forecasting. *arXiv preprint arXiv:2204.10414*, 2022. 7

[48] Sanjoy Dasgupta, Charles F Stevens, and Saket Navlakha. A neural algorithm for a fundamental computing problem. *Science*, 2017. 5.2, 7

[49] Adarsh Dave, Jared Mitchell, Kirthevasan Kandasamy, Han Wang, Sven Burke, Biswajit Paria, Barnabás Póczos, Jay Whitacre, and Venkatasubramanian Viswanathan. Autonomous discovery of battery electrolytes with robotic experimentation and machine learning. *Cell Reports Physical Science*, 1(12):100264, 2020. 7

[50] Emmanuel de Bézenac, Syama Sundar Rangapuram, Konstantinos Benidis, Michael Bohlke-Schneider, Richard Kurle, Lorenzo Stella, Hilaf Hasson, Patrick Gallinari, and Tim Januschowski. Normalizing kalman filters for multivariate time series analysis. In *NeurIPS*, 2020. 6.1

[51] Kalyanmoy Deb and J Sundar. Reference point based multi-objective optimization using evolutionary algorithms. In *Genetic and evolutionary computation*. ACM, 2006. 2.1

[52] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002. 1

[53] Jiankang Deng, Jia Guo, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *arXiv preprint arXiv:1801.07698*, 2018. 5.5, 5.5

[54] Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in neural information processing systems*, 32, 2019. 3.1

[55] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014. 1.1.1

[56] Ronen Eldan, Dan Mikulincer, and Tselil Schramm. Non-asymptotic approximations of neural networks by gaussian processes. In *Conference on Learning Theory*, pages 1754–1775. PMLR, 2021. 3.1, 3.4.1

[57] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018. 7

[58] Michael Emmerich and Jan-willem Klinkenberg. The computation of the expected improvement in dominated hypervolume of Pareto front approximations. *Rapport technique, Leiden University*, 2008. 2.1

[59] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. *CVPR*, 2015. 5.2

[60] Favorita. Favorita forecasting dataset. `https://www.kaggle.com/c/favorita-grocery-sales-forecast`, 2017. 6.5

[61] Sarah Filippi, Olivier Cappe, Aurélien Garivier, and Csaba Szepesvári. Parametric bandits: The generalized linear case. In *Advances in Neural Information Processing Systems*, pages 586–594, 2010. 3.1

[62] Alexander IJ Forrester, András Sóbester, and Andy J Keane. Multi-fidelity optimization via surrogate modelling. *Proceedings of the royal society a: mathematical, physical and engineering sciences*, 463(2088):3251–3269, 2007. 4.1

[63] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018. 1

[64] Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. Predictive entropy search for multi-objective Bayesian optimization with constraints. *arXiv preprint arXiv:1609.01051*, 2016. 2.1

[65] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. *CVPR*, 2013. 5.2

[66] Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, and Basri Ronen. On the similarity between the laplace and neural tangent kernels. *Advances in Neural Information Processing Systems*, 33:1451–1461, 2020. 3.4.1, 3.1

[67] Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Linearized two-layers neural networks in high dimension. *The Annals of Statistics*, 49 (2):1029–1054, 2021. 7

[68] Subhashis Ghosal and Anindya Roy. Posterior consistency of Gaussian process prior for nonparametric binary regression. *The Annals of Statistics*, 2006. 2.5.2

[69] Nicolas Gillis and Stephen A Vavasis. Fast and robust recursive algorithmsfor separable nonnegative matrix factorization. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):698–714, 2013. 6.4, 6.8.2

[70] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. *VLDB*, 1999. 5.2

[71] Jeffrey L Gleason. Forecasting hierarchical time series with a regularized embedding space. *San Diego*, 7, 2020. 6.2, 5

[72] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011. 5.1, 5.2

[73] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1487–1495, 2017. 2, 3

[74] Javier González, Joseph Longworth, David C James, and Neil D Lawrence. Bayesian optimization for synthetic gene design. *arXiv preprint arXiv:1505.01627*, 2015. 1

[75] Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011. 3.1

[76] Shengbo Guo, Scott Sanner, and Edwin V Bonilla. Gaussian process preference elicitation. *Advances in neural information processing systems*, 23, 2010. 7

[77] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. MS-Celeb-1M: A dataset and benchmark for large scale face recognition. In *European Conference on Computer Vision*. Springer, 2016. 5.5

[78] M Hadi Kiapour, Xufeng Han, Svetlana Lazebnik, Alexander C Berg, and Tamara L Berg. Where to buy it: Matching street clothing photos in online shops. In *Proceedings of the IEEE international conference on computer vision*, pages 3343–3351, 2015. 5.1

[79] Patrick Haffner. Fast transpose methods for kernel learning on sparse data. In *Proceedings of the 23rd international conference on Machine learning*, pages 385–392. ACM, 2006. 5.2

[80] Jussi Hakanen and Joshua D Knowles. On using decision maker preferences with ParEGO. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2017. 2.1, 2.2.3, 2.4

[81] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254. IEEE, 2016. 5.3

[82] Xing Han, Sambarta Dasgupta, and Joydeep Ghosh. Simultaneously reconciled quantile forecasting of hierarchically related time series. In *International Conference on Artificial Intelligence and Statistics*, pages 190–198. PMLR, 2021. 6.2, 6

[83] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003. 1

[84] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021. 3.5

[85] Botao Hao, Tor Lattimore, and Mengdi Wang. High-dimensional sparse linear bandits. *Advances in Neural Information Processing Systems*, 33:10753–10763, 2020. 7

[86] Bobby He, Balaji Lakshminarayanan, and Yee Whye Teh. Bayesian deep ensembles via the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:1010–1022, 2020. 2, 3, 3.3, 3.4, 3.4.1, 3.6.2

[87] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009. 7

[88] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 5.5

[89] Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. Predictive entropy search for multi-objective Bayesian optimization. In *International Conference on Machine Learning*, 2016. 3, 1, 2.1, 2.1, 7

[90] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Pre-

dictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, 2014. 4.1, 4.2

[91] José Miguel Hernández-Lobato, James Requeima, Edward O Pyzer-Knapp, and Alán Aspuru-Guzik. Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In *International Conference on Machine Learning*. JMLR, 2017. 1

[92] Geoffrey Hinton and Ruslan Salakhutdinov. Discovering binary codes for documents by learning deep generative models. *Topics in Cognitive Science*, 3(1):74–91, 2011. 7

[93] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006. 5.2

[94] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 6.4

[95] Daniel Hsu, Sham M Kakade, and Tong Zhang. Random design analysis of ridge regression. In *Conference on learning theory*, pages 9–1. JMLR Workshop and Conference Proceedings, 2012. 6.7.4, 6.7.4, 6.1

[96] Deng Huang, Theodore T Allen, William I Notz, and Ning Zeng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of global optimization*, 34(3):441–466, 2006. 1

[97] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008. 5.8.2

[98] 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*. Springer, 2011. 1, 7

[99] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019. 1

[100] Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media, 2008. 6.1

[101] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018. 6.1

[102] Rob J Hyndman and Shu Fan. Density forecasting for long-term peak electricity demand. *IEEE Transactions on Power Systems*, 25(2):1142–1153, 2009. 6.1

[103] Rob J Hyndman, Roman A Ahmed, George Athanasopoulos, and Han Lin Shang. Optimal combination forecasts for hierarchical time series. *Computational statistics & data analysis*, 55(9):2579–2589, 2011. 6.1

[104] Rob J Hyndman, Alan J Lee, and Earo Wang. Fast computation of reconciled fore-

casts for hierarchical and grouped time series. *Computational statistics & data analysis*, 97:16–32, 2016. 6.1

[105] Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima. Evolutionary many-objective optimization: A short review. In *IEEE Congress on Evolutionary Computation*. IEEE, 2008. 2.1

[106] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018. 3.1, 3.2

[107] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *TPAMI*, 2011. 5.1, 5.2

[108] Yeonwoo Jeong and Hyun Oh Song. Efficient end-to-end learning for quantizable representations. *ICML*, 2018. 5.1, 5.2, 5.4, 5.5, 5.5, 5.8.3, 7

[109] Yushi Jing, David Liu, Dmitry Kislyuk, Andrew Zhai, Jiajing Xu, Jeff Donahue, and Sarah Tavel. Visual search at pinterest. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1889–1898. ACM, 2015. 5.1

[110] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017. 5.5

[111] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of optimization Theory and Applications*, 1993. 2.7

[112] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998. 1, 3, 1, 3.7.1, 3.7.1, 4.2, 7

[113] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972. 7

[114] Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. High dimensional Bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning*, 2015. 2.7

[115] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabás Póczos. Multi-fidelity bayesian optimisation with continuous approximations. In *International Conference on Machine Learning*, pages 1799–1808. PMLR, 2017. 1, 4.1, 4.2, 7

[116] Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. Query efficient posterior estimation in scientific experiments via Bayesian active learning. *Artificial Intelligence*, 2017. 1

[117] Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabás Póczos. Parallelised Bayesian optimisation via Thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, 2018. 2.3, 2.5.2, 4.4

[118] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and

Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31, 2018. 1, 3

[119] Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with dragonfly. *arXiv preprint arXiv:1903.06694*, 2019. 1, 7

[120] Sampath Kannan, Jamie H Morgenstern, Aaron Roth, Bo Waggoner, and Zhiwei Steven Wu. A smoothed analysis of the greedy algorithm for the linear contextual bandit problem. *Advances in neural information processing systems*, 31, 2018. 3.3

[121] Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. Fast inference in sparse coding algorithms with applications to object recognition. *arXiv preprint arXiv:1010.3467*, 2010. 5.2

[122] Ira Kemelmacher-Shlizerman, Steven M Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale. *CVPR*, 2016. 5.1, 5.5

[123] Douwe Kiela and Léon Bottou. Learning image embeddings using convolutional neural networks for improved multi-modal semantics. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 36–45, 2014. 5.2

[124] Jong-Hwan Kim, Ji-Hyeong Han, Ye-Hoon Kim, Seung-Hwan Choi, and Eun-Soo Kim. Preference-based solution selection algorithm for evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 2012. 2.1

[125] Samuel Kim, Peter Y Lu, Charlotte Loh, Jamie Smith, Jasper Snoek, and Marin Soljačić. Scalable and flexible deep bayesian optimization with auxiliary information for scientific problems. *arXiv preprint arXiv:2104.11667*, 2021. 1, 2, 3, 3, 3.1, 4

[126] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3.7.1, 6.8.1

[127] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 5.2

[128] Johannes Kirschner and Andreas Krause. Information directed sampling and bandits with heteroscedastic noise. *arXiv preprint arXiv:1801.09667*, 2018. 4.2

[129] Johannes Kirschner, Tor Lattimore, and Andreas Krause. Information directed sampling for linear partial monitoring. *arXiv preprint arXiv:2002.11182*, 2020. 4.2

[130] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 681–690, 2008. 1, 3.1

[131] Joshua Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 2006. 1, 2.1, 2.1, 2.1, 2.2.3

[132] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks

for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015. 5.1

[133] Deguang Kong, Ryohei Fujimaki, Ji Liu, Feiping Nie, and Chris Ding. Exclusive feature learning on arbitrary structures via $\ell_{1,2}$-norm. In *Advances in Neural Information Processing Systems*, pages 1655–1663, 2014. 5.2

[134] Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. *Advances in Neural Information Processing Systems*, 28, 2015. 3.1

[135] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6, 2009. 5.8.3

[136] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 2012. 5.3

[137] Marcin Krotkiewski and Marcin Dabrowski. Parallel symmetric sparse matrix–vector product on scalar multi-core cpus. *Parallel Computing*, 36(4):181–198, 2010. 5.2

[138] Taku Kudo and Yuji Matsumoto. Fast methods for kernel-based text analysis. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 24–31. Association for Computational Linguistics, 2003. 5.2

[139] Sachin Kumar, Eric Malmi, Aliaksei Severyn, and Yulia Tsvetkov. Controlled text generation as continuous optimization with multiple constraints. *Advances in Neural Information Processing Systems*, 34:14542–14554, 2021. 7

[140] Sachin Kumar, Biswajit Paria, and Yulia Tsvetkov. Constrained sampling from language models via langevin dynamics in embedding spaces. *arXiv preprint arXiv:2205.12558*, 2022. 7

[141] Branislav Kveton, Manzil Zaheer, Csaba Szepesvari, Lihong Li, Mohammad Ghavamzadeh, and Craig Boutilier. Randomized exploration in generalized linear bandits. In *International Conference on Artificial Intelligence and Statistics*, pages 2066–2076. PMLR, 2020. 3.1, 3.3, 3.6.3

[142] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017. 3.1

[143] Jeffrey Larson, Matt Menickelly, and Stefan M Wild. Derivative-free optimization methods. *Acta Numerica*, 28:287–404, 2019. 1

[144] Eric Hans Lee, Valerio Perrone, Cedric Archambeau, and Matthias Seeger. Cost-aware bayesian optimization. *arXiv preprint arXiv:2003.10870*, 2020. 2, 4.1, 4.2

[145] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2008. 5.2

[146] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Penning-

ton, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017. 3.1, 3.2

[147] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019. 2, 3, 3.1, 3.2, 3.6.1

[148] Qi Li, Zhenan Sun, Ran He, and Tieniu Tan. Deep supervised discrete hashing. *NeurIPS*, 2017. 5.2

[149] Shibo Li, Wei Xing, Robert Kirby, and Shandian Zhe. Multi-fidelity bayesian optimization via deep neural networks. *Advances in Neural Information Processing Systems*, 33:8521–8531, 2020. 1

[150] Wenjie Li, Adarsh Barik, and Jean Honorio. A simple unified framework for high dimensional bandit problems. *arXiv preprint arXiv:2102.09626*, 2021. 7

[151] Wenye Li, Jingwei Mao, Yin Zhang, and Shuguang Cui. Fast similarity search via optimal sparse lifting. *NeurIPS*, 2018. 5.1, 5.2

[152] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017. 6.1, 6.1, 3

[153] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015. 5.2

[154] Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31(11):4405–4423, 2020. 3, 3.1

[155] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, volume 2, page 7, 2016. 5.2

[156] Daniel James Lizotte. *Practical Bayesian optimization*. University of Alberta, 2008. 2.4

[157] Andrea Locatelli and Alexandra Carpentier. Adaptivity to smoothness in x-armed bandits. In *Conference on Learning Theory*, pages 1463–1492. PMLR, 2018. 1

[158] I Loshchilov and T Glasmachers. Black-box optimization competition (bbcomp), 2015. 3.5

[159] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 2004. 5.1

[160] M5. M5 forecasting dataset. `https://www.kaggle.com/c/m5-forecasting-accuracy/`, 2020. 6.5

[161] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 5.2

[162] Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013. 5.2

[163] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 2014. 5.2

[164] Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *TPAMI*, 2018. 5.2

[165] Ruben Martinez-Cantin, Nando de Freitas, Arnaud Doucet, and José A Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems*, 2007. 1

[166] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951. 5.4

[167] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017. URL `http://jmlr.org/papers/v18/16-537.html`. 1

[168] Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018. 3.1

[169] ED McKenzie. General exponential smoothing and the equivalent arma process. *Journal of Forecasting*, 3(3):333–344, 1984. 6.1

[170] Viraj Mehta, Biswajit Paria, Jeff Schneider, Stefano Ermon, and Willie Neiswanger. An experimental design perspective on model-based reinforcement learning. In *International Conference on Learning Representations*, 2021. 7

[171] Viraj Mehta, Biswajit Paria, Jeff Schneider, Willie Neiswanger, and Stefano Ermon. An experimental design perspective on model-based reinforcement learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=0no8Motr-zO`. 1

[172] Lukas Meier, Sara Van De Geer, and Peter Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008. 5.2

[173] John Mellor-Crummey and John Garvin. Optimizing sparse matrix–vector product computations using unroll and jam. *The International Journal of High Performance Computing Applications*, 18(2):225–236, 2004. 5.2

[174] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021. 7

[175] Konstantin Mishchenko, Mallory Montgomery, and Federico Vaggi. A self-supervised approach to hierarchical forecasting with applications to groupwise synthetic controls. *arXiv preprint arXiv:1906.10586*, 2019. 6.2, 5

[176] Jonas Mockus. The bayesian approach to global optimization. In *System Modeling and Optimization*, pages 473–481. Springer, 1982. 1

[177] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2498–2507. JMLR. org, 2017. 5.2

[178] Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie royale des sciences de Paris*, 1781. 2.5.3

[179] Stylianos Moschoglou, Athanasios Papaioannou, Christos Sagonas, Jiankang Deng, Irene Kotsia, and Stefanos Zafeiriou. Agedb: the first manually collected, in-the-wild age database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop*, volume 2, page 5, 2017. 5.8.2

[180] Yair Movshovitz-Attias, Alexander Toshev, Thomas K Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. *arXiv preprint arXiv:1703.07464*, 2017. 5.2

[181] Vaishnavh Nagarajan and J Zico Kolter. Generalization in deep networks: The role of distance from initialization. *arXiv preprint arXiv:1901.01672*, 2019. 3.3

[182] Hirotaka Nakayama, Yeboon Yun, and Min Yoon. *Sequential approximate multiobjective optimization using computational intelligence*. Springer Science & Business Media, 2009. 2.2.3, 2.4

[183] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012. 3.1

[184] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157, 2016. 5.1

[185] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965. 1

[186] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. In *International Conference on Machine Learning*, pages 1926–1934, 2015. 5.2

[187] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011. 5.2

[188] Hong-Wei Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. In *2014 IEEE international conference on image processing (ICIP)*, pages 343–347. IEEE, 2014. 5.5

[189] Nam Nguyen and Brian Quanz. Temporal latent auto-encoder: A method for prob-

abilistic multivariate time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9117–9125, 2021. 6.1, 6.8.2

[190] Nikolay Nikolov, Johannes Kirschner, Felix Berkenkamp, and Andreas Krause. Information-directed exploration for deep reinforcement learning. *arXiv preprint arXiv:1812.07544*, 2018. 4.2

[191] Qingqun Ning, Jianke Zhu, Zhiyuan Zhong, Steven CH Hoi, and Chun Chen. Scalable image retrieval by sparse product quantization. *IEEE Transactions on Multimedia*, 19(3):586–597, 2016. 5.2

[192] Mohammad Norouzi, David J Fleet, and Ruslan R Salakhutdinov. Hamming distance metric learning. *NeurIPS*, 2012. 5.2

[193] Hyun Oh Song, Stefanie Jegelka, Vivek Rathod, and Kevin Murphy. Deep metric learning via facility location. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5382–5390, 2017. 5.1

[194] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997. 5.2

[195] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437*, 2019. 6.1

[196] Aini Palizhati, Steven B Torrisi, Muratahan Aykol, Santosh K Suram, Jens S Hummelshøj, and Joseph H Montoya. Agents for sequential learning using multiple-fidelity data. *Scientific reports*, 12(1):1–13, 2022. 7

[197] Anastasios Panagiotelis, Puwasala Gamakumara, George Athanasopoulos, Rob J Hyndman, et al. Probabilistic forecast reconciliation: Properties, evaluation and score optimisation. *Monash econometrics and business statistics working paper series*, 26: 20, 2020. 6.1

[198] Theodore Papalexopoulos, Christian Tjandraatmadja, Ross Anderson, Juan Pablo Vielma, and David Belanger. Constrained discrete black-box optimization using mixed-integer programming. *arXiv preprint arXiv:2110.09569*, 2021. 1, 3.3

[199] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 27–40. IEEE, 2017. 5.3

[200] Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. A flexible framework for multi-objective bayesian optimization using random scalarizations. In *Uncertainty in Artificial Intelligence*, pages 766–776. PMLR, 2019. 3, 1, 4.2

[201] Biswajit Paria, Willie Neiswanger, Ramina Ghods, Jeff Schneider, and Barnabás Póczos. Cost-aware bayesian optimization via information directed sampling. *Adaptive Experimental Design and Active Learning in the Real World Workshop at ICML*,

2020. URL https://realworldml.github.io/files/cr/29_Cost_Aware_BO_ICML_2020_Workshop.pdf. 2, 3

[202] Biswajit Paria, Chih-Kuan Yeh, Ian E.H. Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. Minimizing flops to learn efficient sparse representations. In *International Conference on Learning Representations*, 2020. 1

[203] Sungrae Park, JunKeon Park, Su-Jin Shin, and Il-Chul Moon. Adversarial dropout for supervised and semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 5.2

[204] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *empirical methods in natural language processing (EMNLP)*, 2014. 2.4

[205] Victor Picheny. Multiobjective optimization using Gaussian process emulators via stepwise uncertainty reduction. *Statistics and Computing*, 2015. 2.1

[206] John Platt and Alan Barr. Constrained differential optimization. In *Neural Information Processing Systems*, 1987. 7

[207] Wolfgang Ponweiser, Tobias Wagner, Dirk Biermann, and Markus Vincze. Multiobjective optimization on a limited budget of evaluations using model-assisted $\mathcal{S}$-metric selection. In *International Conference on Parallel Problem Solving from Nature*. Springer, 2008. 1, 2.1, 2.1, 7

[208] Michael JD Powell. A view of algorithms for optimization without derivatives. *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications*, 43(5): 170–174, 2007. 1

[209] Viktor K Prasanna and Gerald R Morris. Sparse matrix computations on reconfigurable hardware. *Computer*, 40(3):58–64, 2007. 5.2

[210] Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. *NeurIPS*, 2009. 5.2

[211] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011. 7

[212] Parikshit Ram and Alexander G Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 931–939. ACM, 2012. 5.2

[213] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. *Advances in neural information processing systems*, 31:7785–7794, 2018. 6.1

[214] Syama Sundar Rangapuram, Lucien D Werner, Konstantinos Benidis, Pedro Mercado, Jan Gasthaus, and Tim Januschowski. End-to-end learning of coherent probabilistic forecasts for hierarchical time series. In *International Conference on Machine Learning*, pages 8832–8843. PMLR, 2021. 6.2, 7

[215] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann L Cun. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2007. 5.2

[216] Marc'Aurelio Ranzato, Y-Lan Boureau, and Yann L Cun. Sparse feature learning for deep belief networks. In *Advances in neural information processing systems*, pages 1185–1192, 2008. 5.2

[217] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003. 3.2, 3.7.1

[218] Carl Edward Rasmussen and Christopher KI Williams. Gaussian processes for machine learning. 2006. *The MIT Press, Cambridge, MA, USA*, 2006. 1, 1.1.1, 2.7

[219] Kashif Rasul, Abdul-Saboor Sheikh, Ingmar Schuster, Urs Bergmann, and Roland Vollgraf. Multivariate probabilistic time series forecasting via conditioned normalizing flows. *arXiv preprint arXiv:2002.06103*, 2020. 6.1

[220] Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018. 3, 3, 3.3, 3.3.1, 3.7.1, 7

[221] R Tyrrell Rockafellar. *Convex Analysis*, volume 36. Princeton University Press, 1970. 1.1.2

[222] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 2013. 2.1, 2.1

[223] Diederik M Roijers, Luisa M Zintgraf, and Ann Nowé. Interactive Thompson sampling for multi-objective multi-armed bandits. In *International Conference on Algorithmic DecisionTheory*. Springer, 2017. 2.1, 2.2.3

[224] Diederik M Roijers, Luisa M Zintgraf, Pieter Libin, and Ann Nowé. Interactive multi-objective reinforcement learning in multi-armed bandits for any utility function. In *ALA workshop at FAIM*, 2018. 2.1, 2.2.3

[225] Esther Rolf, Max Simchowitz, Sarah Dean, Lydia T Liu, Daniel Bjorkegren, Moritz Hardt, and Joshua Blumenstock. Balancing competing objectives with noisy data: Score-based classifiers for welfare-aware machine learning. In *International Conference on Machine Learning*, pages 8158–8168. PMLR, 2020. 7

[226] Philip A Romero, Andreas Krause, and Frances H Arnold. Navigating the protein fitness landscape with gaussian processes. *Proceedings of the National Academy of Sciences*, 110(3):E193–E201, 2013. 1

[227] Constantin A Rothkopf and Christos Dimitrakakis. Preference elicitation and inverse reinforcement learning. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 34–48. Springer, 2011. 7

[228] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 7

[229] Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 2014. 3, 2.3, 2.5, 3, 4.1, 4.3, 4.3, 4.4, 7

[230] Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014. 1.1.1, 4.2

[231] Daniel Russo and Benjamin Van Roy. An information-theoretic analysis of thompson sampling. *The Journal of Machine Learning Research*, 17(1):2442–2471, 2016. 4.2, 4.3

[232] Alexandre Sablayrolles, Matthijs Douze, Nicolas Usunier, and Hervé Jégou. How should we evaluate supervised hashing? In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1732–1736. IEEE, 2017. 5.4

[233] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Spreading vectors for similarity search. *ICLR*, 2019. 5.4

[234] David Salinas, Michael Bohlke-Schneider, Laurent Callot, Roberto Medico, and Jan Gasthaus. High-dimensional multivariate forecasting with low-rank gaussian copula processes. *arXiv preprint arXiv:1910.03002*, 2019. 6.1, 6.2

[235] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020. 6.1

[236] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018. 5.4

[237] Hiroyuki Sato, Hernán E Aguirre, and Kiyoshi Tanaka. Controlling dominance area of solutions and its impact on the performance of moeas. In *International conference on evolutionary multi-criterion optimization*. Springer, 2007. 2.1

[238] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015. 5.2, 5.8.3

[239] Rajat Sen, Hsiang-Fu Yu, and Inderjit Dhillon. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. *arXiv preprint arXiv:1905.03806*, 2019. 6.1, 6.4, 2, 6.8.2

[240] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *arXiv preprint arXiv:1810.04650*, 2018. 7

[241] Ohad Shamir. On the complexity of bandit and derivative-free stochastic convex optimization. In *Conference on Learning Theory*, pages 3–24. PMLR, 2013. 3.1

[242] Devashish Shankar, Sujay Narumanchi, HA Ananya, Pramod Kompalli, and Krishnendu Chaudhury. Deep learning based large scale visual recommendation and search for e-commerce. *arXiv preprint arXiv:1703.02344*, 2017. 5.1

[243] K Sharman and Benjamin Friedlander. Time-varying autoregressive modeling of a

class of nonstationary signals. In *ICASSP'84. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 9, pages 227–230. IEEE, 1984. 6.1

[244] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pages 2321–2329, 2014. 5.1

[245] Aleksandrs Slivkins. Introduction to multi-armed bandits. *arXiv preprint arXiv:1904.07272*, 2019. 3.3

[246] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, 2012. 1, 1, 7

[247] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015. (document), 1, 3, 3.1, 3.1

[248] Nimit Sharad Sohoni, Christopher Richard Aberger, Megan Leszczynski, Jian Zhang, and Christopher Ré. Low-memory neural network training: A technical report. *arXiv preprint arXiv:1904.10631*, 2019. 5.3

[249] Jialin Song, Yuxin Chen, and Yisong Yue. A general framework for multi-fidelity bayesian optimization with gaussian processes. *arXiv preprint arXiv:1811.00755*, 2018. 4.1, 4.2

[250] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. *Advances in neural information processing systems*, 29, 2016. 3, 3, 3.1

[251] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009. 1, 3.1, 3.7.1, 4.2, 4.4, 4.5, 7

[252] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *International Conference on Machine Learning*. JMLR, 2010. 1.1.1, 2.3, 2.5.1, 2.7

[253] 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. 5.2

[254] Gilbert Strang. Wavelet transforms versus fourier transforms. *Bulletin of the American Mathematical Society*, 28(2):288–305, 1993. 6.6

[255] Sonja Surjanovic and Derek Bingham. Virtual library of simulation experiments: test functions and datasets. *Simon Fraser University, Burnaby, BC, Canada*, 13:2015, 2013. URL `https://www.sfu.ca/~ssurjano/hart6.html`. (document), 3.7.2, 3.6

[256] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NIPS 2014)*, 2014. 6.1

[257] Souhaib Ben Taieb, James W Taylor, and Rob J Hyndman. Coherent probabilistic forecasts for hierarchical time series. In *International Conference on Machine Learning*, pages 3348–3357. PMLR, 2017. 6.1

[258] Shion Takeno, Hitoshi Fukuoka, Yuhki Tsukada, Toshiyuki Koyama, Motoki Shiga, Ichiro Takeuchi, and Masayuki Karasuyama. Multi-fidelity bayesian optimization with max-value entropy search. *arXiv preprint arXiv:1901.08275*, 2019. 4.1, 4.2

[259] Lothar Thiele, Kaisa Miettinen, Pekka J Korhonen, and Julian Molina. A preference-based evolutionary algorithm for multi-objective optimization. *Evolutionary computation*, 2009. 2.1

[260] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 1933. 1, 1.1.1, 2.1, 3

[261] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 5.2

[262] Tourism. Tourism forecasting dataset. `https://robjhyndman.com/publications/mint/`, 2019. 6.5

[263] Tsuyoshi Ueno, Trevor David Rhone, Zhufeng Hou, Teruyasu Mizoguchi, and Koji Tsuda. Combo: An efficient bayesian optimization library for materials science. *Materials discovery*, 4:18–21, 2016. 1

[264] Sattar Vakili, Kia Khezeli, and Victor Picheny. On information gain and regret bounds in gaussian process bandits. In *International Conference on Artificial Intelligence and Statistics*, pages 82–90. PMLR, 2021. 3.4.1

[265] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *9th ISCA Speech Synthesis Workshop*, pages 125–125, 2016. 6.6

[266] Mark Van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional gaussian processes. *Advances in Neural Information Processing Systems*, 30, 2017. 3

[267] Tim Van Erven and Jairo Cugliari. Game-theoretically optimal reconciliation of contemporaneous hierarchical time series forecasts. In *Modeling and stochastic learning for forecasting in high dimensions*, pages 297–317. Springer, 2015. 6.1

[268] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017. 6.4

[269] Francisco Vazquez, G Ortega, José-Jesús Fernández, and Ester M Garzón. Improving the performance of the sparse matrix vector product with gpus. In *2010 10th IEEE International Conference on Computer and Information Technology*, pages 1146–1151. IEEE, 2010. 5.2

[270] Francisco Vázquez, José-Jesús Fernández, and Ester M Garzón. A new approach for sparse matrix vector product on nvidia gpus. *Concurrency and Computation: Practice and Experience*, 23(8):815–826, 2011. 5.2

[271] Cédric Villani. *Optimal transport: old and new*. Springer Science & Business Media, 2008. 2.5.3

[272] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition*. IEEE, 2001. 2.4

[273] Martin J Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1-constrained quadratic programming (lasso). *IEEE transactions on information theory*, 55(5):2183–2202, 2009. 6.6, 6.7.3, 6.2

[274] Martin J Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*. Cambridge University Press, 2019. 6.6.1

[275] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):769–790, 2018. 5.2

[276] Xueyi Wang. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1293–1299. IEEE, 2011. 5.1

[277] Yuyang Wang, Alex Smola, Danielle Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. Deep factors for forecasting. In *International Conference on Machine Learning*, pages 6607–6617. PMLR, 2019. 6.1, 6.4, 4

[278] Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635. PMLR, 2017. 4.1, 4.2

[279] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009. 5.1, 5.2

[280] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. *NeurIPS*, 2009. 5.2

[281] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *NeurIPS Time Series Workshop*, 2017. 6.4

[282] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016. 5.2

[283] Shanika L Wickramasuriya, George Athanasopoulos, Rob J Hyndman, et al. Forecasting hierarchical and grouped time series through trace minimization. *Department of Econometrics and Business Statistics, Monash University*, 105, 2015. 6.1

[284] Shanika L Wickramasuriya, George Athanasopoulos, and Rob J Hyndman. Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. *Journal of the American Statistical Association*, 114(526):804–819, 2019. 8

[285] Shanika L Wickramasuriya, Berwin A Turlach, and Rob J Hyndman. Optimal non-negative forecast reconciliation. *Statistics and Computing*, 30(5):1167–1182, 2020. 6.1

[286] Kevin W Wilson, Bhiksha Raj, and Paris Smaragdis. Regularized non-negative matrix factorization with temporal dependencies for speech denoising. In *Interspeech*, pages 411–414, 2008. 6.8.2

[287] Jian Wu, Saul Toscano-Palmerin, Peter I Frazier, and Andrew Gordon Wilson. Practical multi-fidelity bayesian optimization for hyperparameter tuning. In *Uncertainty in Artificial Intelligence*, pages 788–798. PMLR, 2020. 1

[288] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 753–763, 2020. 6.1

[289] Pan Xu, Zheng Wen, Handong Zhao, and Quanquan Gu. Neural contextual bandits with deep representation and shallow exploration. *arXiv preprint arXiv:2012.01780*, 2020. 2, 3, 3.1, 2, 3.5

[290] Anna K Yanchenko, Di Daniel Deng, Jinglan Li, Andrew J Cron, and Mike West. Hierarchical dynamic modeling for individualized bayesian forecasting. *arXiv preprint arXiv:2101.03408*, 2021. 6.2

[291] Kaifeng Yang, Longmei Li, André Deutz, Thomas Back, and Michael Emmerich. Preference-based multiobjective optimization using truncated expected hypervolume improvement. In *Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2016. 2.1

[292] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017. 6.1

[293] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 2007. 2.1

[294] Qingfu Zhang, Wudong Liu, Edward Tsang, and Botond Virginas. Expensive multiobjective optimization by moea/d with Gaussian process model. *IEEE Transactions on Evolutionary Computation*, 2010. 2.1, 2.1

[295] Weitong Zhang, Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural thompson sampling. *arXiv preprint arXiv:2010.00827*, 2020. 3, 3.1, 3.1, 3.5

[296] Yan Zhang, Yasser H Shalabi, Rishabh Jain, Krishna K Nagar, and Jason D Bakos. Fpga vs. gpu for sparse matrix vector multiply. In *2009 International Conference on Field-Programmable Technology*, pages 255–262. IEEE, 2009. 5.2

[297] Miao Zhong, Kevin Tran, Yimeng Min, Chuanhao Wang, Ziyun Wang, Cao-Thang Dinh, Phil De Luna, Zongqian Yu, Armin Sedighian Rasouli, Peter Brodersen, et al. Accelerated discovery of co2 electrocatalysts using active machine learning. *Nature*, 581(7807):178–183, 2020. 7

[298] Dawei Zhou, Lecheng Zheng, Yada Zhu, Jianbo Li, and Jingrui He. Domain adaptive multi-modality neural attention network for financial forecasting. In *Proceedings of The Web Conference 2020*, pages 2230–2240, 2020. 6.1

[299] Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural contextual bandits with ucb-based exploration. In *International Conference on Machine Learning*, pages 11492–11502. PMLR, 2020. (document), 2, 3, 3.1, 3, 3.1, 3.1, 3, 3.7.1, 3.7.1

[300] Yang Zhou, Rong Jin, and Steven Chu-Hong Hoi. Exclusive lasso for multi-task feature selection. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 988–995, 2010. 5.2

[301] Ling Zhuo and Viktor K Prasanna. Sparse matrix-vector multiplication on fpgas. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 63–74. ACM, 2005. 5.2

[302] Luisa M Zintgraf, Timon V Kanters, Diederik M Roijers, Frans A Oliehoek, and Philipp Beau. Quality assessment of MORL algorithms: A utility-based approach. In *Belgian-Dutch Conference on Machine Learning*, 2015. 2.1, 2.1, 2.2.2

[303] Luisa M Zintgraf, Diederik M Roijers, Sjoerd Linders, Catholijn M Jonker, and Ann Nowé. Ordered preference elicitation strategies for supporting multi-objective decision making. *arXiv preprint arXiv:1802.07606*, 2018. 2.1, 7

[304] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer, 1998. 1

[305] Marcela Zuluaga, Guillaume Sergent, Andreas Krause, and Markus Püschel. Active learning for multi-objective optimization. In *International Conference on Machine Learning*, 2013. 1, 2.1, 2, 2.1

[306] Marcela Zuluaga, Andreas Krause, and Markus Püschel. ε-pal: an active learning approach to the multi-objective optimization problem. *Journal of Machine Learning Research*, 2016. 2.1