Practical Optimization for Structured Machine Learning Problems

by

Mohamed Osama Ahmed

B.Sc., Cairo University, 2006
M.Sc., Cairo University, 2010
MASc., The University of British Columbia, 2013

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver, Canada)

December 2018

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the dissertation entitled:

Practical Optimization for Structured Machine Learning Problems

submitted by **Mohamed Osama Ahmed** in partial fulfillment of the requirements for the

degree of **Doctor of Philosophy** in **Computer Science**

Examining Committee:

Mark Schmidt, Supervisor

Giuseppe Carenini, Supervisory Committee Member

Mike Gelbart, Supervisory Committee Member

Leonid Sigal, University Examiner

Alexandra Fedorova, University Examiner

Abstract

Recent years have witnessed huge advances in machine learning (ML) and its applications, especially in image, speech, and language applications. Optimization in ML is a key ingredient in both the training and hyperparameter tuning steps, and it also influences the test phase. In this thesis, we make improvements to optimization of all of these three problems. The first part of the thesis considers the training problem. We present the first linearly-convergent stochastic gradient method to train conditional random fields (CRFs) using the stochastic average gradient (SAG) method. Our method addresses the memory issues required for SAG and proposes a better non-uniform sampling (NUS) technique. The second part of the thesis deals with memory-free linearly-convergent stochastic gradient method for training ML models. We consider the stochastic variance reduced gradient (SVRG) algorithm, which normally requires occasional full-gradient evaluations. We show that we can decrease the number of gradient evaluations using growing batches (which improves performance in the early iterations) and support vectors (which improves performance in the later iterations). The third part of this thesis switches to the problem of hyper-parameter tuning. Although it is a lower dimensional problem, it can be more difficult as it is a non-convex global optimization problem. First, we propose a harmless global optimization algorithm to ensure that the performance is not worse than using random search. Moreover, we explore the use of gradient-based Bayesian optimization (BO) to improve the performance. We propose the use of directional derivatives to address the memory issues of BO. Finally, in the fourth part of this thesis, we propose a novel optimization method that combines the advantages of BO and Lipschitz optimization (LO).

Lay Summary

Machine learning is helping advance a lot of technologies. Building powerful machine learning (ML) models requires solving challenging mathematical problems known as optimization. Optimization for ML is a hard problem, especially with the recent trend of using big datasets. This work focuses on improving the optimization algorithms that are used to build ML models. With our proposed methods, one can train a model in less time and achieve better results. The second part of our work focuses on tuning the "mysterious" parameters of ML models known as "hyperparameters". Our work will help researchers build ML models without having to make hard choices about which hyperparameter configuration to use. In this thesis, we improve over the conventional hyperparameter tuning methods.

Preface

The work presented in this thesis is a result of four separate research projects that are based on collaborative work that has been accepted or under review.

- Chapter 2 resulted in the following paper: "M. Schmidt, R. Babanezhad, M. O. Ahmed, A. Defazio, and A. Sarkar. Non-uniform stochastic average gradient method for training conditional random fields. In AISTATS,2015." My role was developing the new SAG for CRF algorithm. I was responsible for all the implementation details and the simulation results. I explored the use of different non-uniform sampling techniques and different step-size methods. The theory was developed by Reza Babanezhad and Mark Schmidt. I was responsible for writing the experiment section. The rest of the co-authors contributed to some of the experiments.

- Chapter 3 resulted in the following paper: "R. Harikandeh, M. O. Ahmed, A. Virani, M. Schmidt, J. Konecny, and S. Sallinen. Stop wasting my gradients: Practical SVRG." In Advances in Neural Information Processing Systems, 2015." I proposed the use of mixed SVRG and support vectors and I explored the use of various mini-batching methods. I was responsible for developing the algorithm, all the implementation details, and the simulation results. Also, I did the writing of the experimental results section. The theory was developed by Reza Harikandeh and Mark Schmidt. The rest of the co-authors contributed to some of the experiments.

- Chapter 4 resulted in the following paper: "M. O. Ahmed, B. Shahriari, and M. Schmidt. Do we need harmless Bayesian optimization and first-order Bayesian optimization?" in NIPS Bayesian Optimization workshop, 2016." I proposed the harmless BO method and the use of directional derivatives with BO. I developed the method, conducted all the experiments, and I provided a state-of-the art BO implementation. The theory was developed by Mark Schmidt. Bobak Shahriari contributed with the pybo package. As for the writing, it was a joint effort between all authors.

- Chapter 5 resulted in the following paper: "M. O. Ahmed, S. Vaswani, and M. Schmidt. Combining Bayesian Optimization and Lipschitz Optimization." Submitted to AISTATS, 2019. I developed the method and conducted all the experiments. I proposed the different methods to combine Lipschitz bounds with BO. I did all the work except the theory (done by Mark Schmidt and Sharan Vaswani). As for the writing, it was a joint effort between all authors.

Table of Contents

Appendices

List of Tables

List of Figures

List of Abbreviations

BO	Bayesian Optimization
CRF	Conditional Random Field
FOBO	First Order Bayesian Optimization
HBO	Harmless Bayesian Optimization
LBO	Lipschitz Bayesian Optimization
SAG	Stochastic Average Gradient
SAGA	Stochastic Average Gradient Amélioré
SGD	Stochastic Gradient Descent
SVRG	Stochastic Variance Reduced Gradient

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisor Mark Schmidt for his guidance, patience and support throughout the thesis. I learned a lot while working with you. Your constant encouragement helped me overcome several obstacles. To my supervisory committee, Giuseppe Carenini, Sohrab Shah, and Mike Gelbart – thank you for your time, efforts and helpful feedback. To my examiners, Leonid Sigal, Alexandra Fedorova, and Ruben Martinez-Cantin – thank you for your valuable feedback. To my Master thesis supervisor, Lutz Lampe – thank you for supporting my decision to switch from studying communications to machine learning. To my colleagues at UBC – thank you for the wonderful discussions. To my coauthors, Reza, Bobak, and Sharan – thank you for the great team work. To all the support staff at UBC, especially Joyce Poon and Kath Imhiran – thank you for your patience and all your help with logistics and paper work.

To my bigger family in Egypt, my aunt Somaya, and my uncles Mostafa and Magdy – thank you for being my second parents. To my family in Ottawa, Zeinab, Mokbel, Sarah, Nadia, and Nora Sayed – thank you for making me feel home. To my friend, Ayman – thank you for all your help, long walks, and wonderful food. To my friends Azab and Anas – thank you for listening when I had tough times. To my friends at UBC and Vancouver, Ramy, Shady, Hazim, Belal, and Radwa – thank you for all your support, the wonderful times and the weekend gatherings. You made my last years at UBC amazing.

Last but most importantly, special thanks to my father *Osama*, my mother *Nagwa*, my sister *Nora* and my brother *Ahmed* for their continuous support and encouragement during tough times. None of the work in this thesis, or anything else I have ever achieved, could have been possible without you.

To the memory of my grandmothers, Monira Owida and Thuraya Zayed.

Chapter 1

Introduction

"Artificial intelligence (AI) is the new electricity" - Andrew Ng. AI is playing a big role in the current technology revolution not only in speech recognition [Hinton et al., 2012] and image recognition [Krizhevsky et al., 2012], but also in health care [Thodoroff et al., 2016], physics [Baldi et al., 2014], and even in music generation [Boulanger-Lewandowski et al., 2012]. AI is opening doors for new opportunities in various fields. Machine learning (ML) is a subfield of AI that deals with how to teach machines to learn like humans using data. ML has empowered a lot of aspects in our life such as natural language processing [Collobert et al., 2011], time series forecasting [Längkvist et al., 2014], and the soon-to-arrive self-driving cars [Teichmann et al., 2016]. We are living the age of data. A lot of data are collected through phones, computers, and other electronic devices. Such large amounts of data enable us to build more accurate and more powerful machine learning models. One key ingredient in building ML models is optimization. With the growing sizes of datasets, designing more practical optimization algorithms is crucial. This is clear in two main problems: training and hyperparameter tuning.

1.1 Training a Machine Learning Model

Given a dataset of n training example pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, the training problem can be formulated as finding the model parameters w that minimizes f the loss function that measures how well the model fits the data. So the training optimization problem can be written as:

$$\min_{w \in \mathbb{R}^d} f(w) = \min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(w, x_i, y_i) \tag{1.1}$$

where i represents the index for the training example.

A huge proportion of the model-fitting procedures in machine learning can be mapped to this problem. This includes classic models like least squares and logistic regression but also includes more advanced methods like conditional random fields and deep neural network models. In the high-dimensional setting (large d), the common approaches for solving (1.1) are:

1. Full gradient (FG) methods.
2. Stochastic gradient (SG) methods

3. Variance-reduced stochastic gradient methods

We now review the core ideas behind each of these methods.

1.1.1 Gradient Descent

This is the classic first order method that uses gradients to minimize f. As the gradient points in the direction of maximizing increase in f, if we take a step in the negative direction of f, we can find a point with a lower function value. The update can be written as:

$$w^{t+1} = w^t - \alpha_t \nabla f(w^t) = w^t - \frac{\alpha_t}{n} \sum_{i=1}^{n} \nabla f_i(w), \tag{1.2}$$

where t is the iteration number, w^t is the current point, w^{t+1} is the next point chosen, ∇f is the gradient of f, and α_t is the current step-size. Choosing the step size is crucial to the performance. It determines how fast the algorithm converges to the solution. If it is too small, it will take too many iterations to reach the solution, while a too large step-size can cause the algorithm to diverge. The choice of α can be either constant, adaptive (determined with line-search) [Nocedal and Wright, 2006], or a hyperparameter that is determined through hyperparameter optimization techniques. For (1.1), FG provides linear convergence rates [Nesterov, 2004]. However, it requires evaluating the gradients f_i for all n examples on every iteration.

1.1.2 Stochastic Gradient Descent

SG methods provide a cheaper alternative to train ML models. In contrast to FG, SG evaluates the gradient for only one training example, chosen at random, at each iteration and then takes a step as follows:

$$w^{t+1} = w^t - \alpha_t \nabla f_i(w^t), \tag{1.3}$$

where ∇f_i represents the gradient of the cost function that corresponds to the i^{th} training example. If we have a data set with n training examples, the iterations of stochastic gradient methods are n times faster than deterministic methods. However, the number of stochastic gradient iterations required might be very high. This has been studied in the optimization community, which considers the problem of finding the minimum number of iterations t so that we can guarantee that we reach an accuracy of ε, meaning that

$$f(w^t) - f(w^*) \leq \varepsilon,$$

where w^* is the parameter vector minimizing the training objective function. SG makes rapid initial progress as it only uses a single gradient on each iteration but ultimately has slower sublinear convergence rates than FG. SG can be improved by using mini-batching which evaluates the gradient for m training examples at each iteration as follows:

$$w^{t+1} = w^t - \frac{\alpha_t}{m} \sum_{i=1}^{m} \nabla f_i(w^t), \tag{1.4}$$

where $m < n$. This decreases the variance of the gradient estimate, at the cost of additional gradient calculations.

1.1.3 Variance-reduced Stochastic Gradient Descent

Le Roux et al. [2012] proposed the first general method, *stochastic average gradient* (SAG), that only considers one training example on each iteration but still achieves a linear convergence rate. The update of the parameters can be written as follows:

$$w^{t+1} = w^t - \frac{\alpha_t}{n} \sum_{i=1}^{n} g_i^t. \tag{1.5}$$

For FG, $g_i^t = \nabla f_i(w)$. The SAG algorithm uses this same iteration, but instead of updating g_i^t for all n data points on every iteration, it simply sets $g_i^t = \nabla f_i(w)$ for *one randomly chosen* data point and keeps the remaining g_i^t at their value from the previous iteration (stored in memory). Thus the SAG algorithm is a randomized version of the gradient algorithm where we use the gradient of each example from the last iteration where it was selected. The surprising aspect of the work of Le Roux et al. [2012] is that this simple *delayed* gradient algorithm achieves a similar convergence rate to the classic FG algorithm despite the iterations being n times faster. Other methods have subsequently been shown to have this property [Defazio et al., 2014a, Mairal, 2013, Shalev-Schwartz and Zhang, 2013], but these all require storing a previous evaluation of the gradient ∇f_i or the dual variables for each i. For many objectives this only requires $O(n)$ space, but for general problems this requires $O(nd)$ space making them impractical. Recently, several methods have been proposed with similar convergence rates to SAG but without the memory requirements such as *stochastic variance-reduced gradient* (SVRG).

The first part of this thesis, focuses on the training part of the ML pipeline. We consider work that applies to the SAG and the SVRG method.

1.2 Hyperparameter Tuning

Hyperparameter tuning is another example where optimization plays a big role in building ML models. Training an ML model assumes that we make certain choices about the model. For example, before training a neural network, we have to choose the number of layers, number of units per layer, type of nonlinearity, and learning rate (step size). These parameters are known as hyperparameters. Finding good values of these parameters is not an easy problem. It is generally nonconvex and with big models or large datasets, evaluating the loss function is expensive. Thus a variety of methods have been proposed to address this problem.

Typically, we divide the data available for an ML model into 3 parts: training, validation, and test. The training data are used to solve (1.1). The validation data are used to pick the hyperparameters of the model. The test data are used to report the expected performance

of the model on unseen data. The most common approaches for hyperparameter tuning are grid search, random search, and Bayesian optimization.

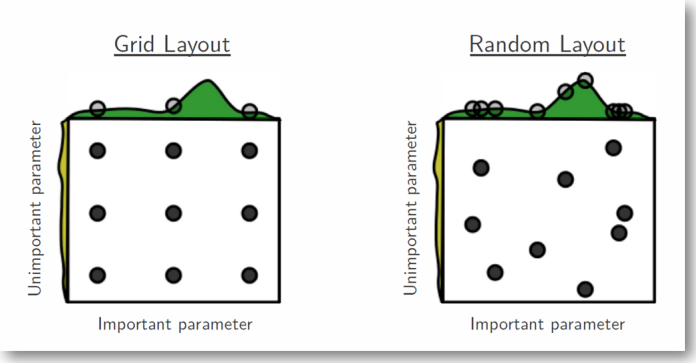


Figure 1.1: Grid search vs. Random search [Bergstra and Bengio, 2012]

1.2.1 Grid Search

This is a classic hyperparameter search method. The idea is to have a grid of hyperparameter configurations to evaluate. The cost function, which in the case of hyperparameter search is the validation loss, is calculated for each configuration. The hyperparameters are chosen to provide the best performance on the validation set.

1.2.2 Random Search

Bergstra and Bengio [2012] proposed the use of random search instead of grid search. Instead of specifying a fixed grid, we specify a probability distribution to sample from for each hyperparameter. The motivation is that grid search can waste a lot of unnecessary function evaluations. For example, if one of the hyperparameters is not important, then there is no point in trying different values for this parameter. Having a random grid increases the chances of sampling better configurations compared to grid search. Figure 1.1 shows an example where grid search is wasting a lot of its budget on an unimportant hyperparameter. Grid search only tries 3 values for the important parameter out of 9 possible configurations. On the other hand, random search uses the same budget of 9 configurations, however, it tries more values for the important parameter.

1.2.3 Bayesian Optimization

Although random search is better than grid search, it does not make use of the previous points evaluated. For example, if we are training a neural network with 1 hidden layer with 100 units, there is no point to try the same architecture but with 101 units instead of 100. We

expect the performance to be somewhat similar. Instead of searching randomly, we want a procedure that tries out a set of hyper-parameters that provides the most information at each step. Bayesian optimization (BO) is a technique that uses a model based optimization approach. Shahriari et al. [2016] provides a recent review on various BO techniques. BO provides a framework that enables us to pick the configurations to evaluate sequentially. BO performs this task by starting with a prior belief about the function to be optimized f. This prior is usually a Gaussian process. This is a flexible prior that allows us to model a large classes of functions. With each observation $(x_t, f(x_t))$, BO refines its belief about f (the posterior). This posterior can be used to determine which configuration to evaluate next using the *acquisition function*. There is a famous trade-off between *exploration* and *exploitation*. On one hand, we want to sample points of hight posterior uncertainty to learn more about f (exploration). On the other, our main goal is to find the configuration with the best performance (exploitation). Depending on the choice of acquisition functions, BO can favor exploration or exploitation. One of the main problems with BO is to tune this trade off. In certain experimental settings, BO can be outperformed by using random search for twice as many iterations [Li et al., 2017]. However, in other works and many of our experiments, BO outperforms random by a substantial margin. This leads us to consider designing better acquisition functions and better BO methods that perform no worse than random.

The second part of the thesis focuses on designing better BO methods in terms of the acquisition function and also that perform no worse than random search. The BO methods designed in the second part of the thesis can be used to improve the optimization methods proposed in the first part of the thesis. For example, α_t in (1.2) can be set by BO.

1.3 Thesis Outline

In this work, we consider both parts of the ML pipeline; the training and the hyperparameter search.

- Chapter 2 studies the SAG problem. We consider the challenging problem of applying SAG for conditional random fields (CRFs). This is the first work that tackles this problem with the challenging memory requirements. Moreover, we propose a non-uniform sampling scheme that substantially improves the practical performance of the method.

- Chapter 3 considers SVRG which does not have the same expensive memory requirements as SAG. This comes at the expense of having to do more gradient calculations. The purpose of the work presented in this chapter is to decrease the number of gradient calculations required by SVRG while maintaining the performance.

- Chapter 4 switches to the more difficult problem of hyperparameter search. Our focus is on building better BO algorithms. We propose two important directions for the BO community to explore to improve the performance of these methods. The first is to use harmless BO. This is an algorithm that does no worse than random in

the worst case, but that might still take advantage of certain structure of the function, for example, a high degree of smoothness in case of BO. The second contribution is to investigate the use of first-order BO methods. These are methods that take into consideration the gradient information. We also propose using directional derivatives. This is cheaper than using the full gradients and provides comparable performance.

- Chapter 5 proposes using an idea from the global optimization literature to design a better BO algorithm. We propose using Lipschitz bounds to help speed up BO. The intuition here is that BO has a trade off between exploration and exploitation. This exploration part requires evaluating points in the parts of the space with high uncertainty to learn more about the function. Lipschitz bounds eliminates the infeasible parts of the space. We use the Lipschitz bounds to reduce the size of the space that BO has to search and we introduce a new algorithm, Lipschitz Bayesian optimization (LBO).

Chapter 2

Stochastic Average Gradient for Conditional Random Fields

Conditional random fields (CRFs) [Lafferty et al., 2001] are a ubiquitous tool in natural language processing. They are used for part-of-speech tagging [McCallum et al., 2003], semantic role labeling [Cohn and Blunsom, 2005], topic modeling [Zhu and Xing, 2010], information extraction [Peng and McCallum, 2006], shallow parsing [Sha and Pereira, 2003], named-entity recognition [Settles, 2004], as well as a host of other applications in natural language processing and in other fields such as computer vision [Nowozin and Lampert, 2011]. Similar to generative Markov random field (MRF) models, CRFs allow us to model probabilistic dependencies between output variables. The key advantage of discriminative CRF models is the ability to use a very high-dimensional feature set, without explicitly building a model for these features (as required by MRF models). Despite the widespread use of CRFs, a major disadvantage of these models is that they can be very slow to train and the time needed for numerical optimization in CRF models remains a bottleneck in many applications.

Due to the high cost of evaluating the CRF objective function on even a single training example, it is now common to train CRFs using stochastic gradient methods [Vishwanathan et al., 2006]. These methods are advantageous over deterministic methods because on each iteration they only require computing the gradient of a single example (and not *all* example as in deterministic methods). Thus, if we have a data set with n training examples, the iterations of stochastic gradient methods are n times faster than deterministic methods. However, the number of stochastic gradient iterations required might be very high. This has been studied in the optimization community, which considers the problem of finding the minimum number of iterations t so that we can guarantee that we reach an accuracy of ε, meaning that

$$f(w^t) - f(w^*) \leq \varepsilon, \text{ and } \|w^t - w^*\|^2 \leq \varepsilon,$$

where f is our training objective function, w^t is our parameter estimate on iteration t, and w^* is the parameter vector minimizing the training objective function. For strongly-convex objectives like ℓ_2-regularized CRFs, stochastic gradient methods require $O(1/\varepsilon)$ iterations [Nemirovski et al., 2009]. This is in contrast to traditional deterministic methods which only require $O(\log(1/\varepsilon))$ iterations [Nesterov, 2004]. However, this much lower number of iterations comes at the cost of requiring us to process the entire data set on each iteration.

For problems with a finite number of training examples, Le Roux et al. [2012] recently

proposed the stochastic average gradient (SAG) algorithm which combines the advantages of deterministic and stochastic methods: it only requires evaluating a single randomly-chosen training example on each iteration, and only requires $O(\log(1/\varepsilon))$ iterations to reach an accuracy of ε. Beyond this faster convergence rate, the SAG method also allows us to address two issues that have traditionally frustrated users of stochastic gradient methods: *setting the step-size* and *deciding when to stop*. Implementations of the SAG method use both an adaptive step-size procedure and a cheaply-computable criterion for deciding when to stop. Le Roux et al. [2012] show impressive empirical performance of the SAG algorithm for binary classification. Recently, Defazio et al. [2014a] proposed the SAGA algorithm. SAGA is easier to analyze. One of the contributions of the work in this chapter is to propose an extension of SAGA called SAGA2. It is a variant of SAGA that requires two training samples at each iteration but can improve the convergence rate.

Another important contribution in this chapter focuses on non-uniform sampling (NUS). Recently, several works show that we can improve the convergence rates of randomized optimization algorithms by using non-uniform sampling (NUS) schemes. This includes randomized Kaczmarz [Strohmer and Vershynin, 2009], randomized coordinate descent [Nesterov, 2012], and stochastic gradient methods [Needell et al., 2014]. The key idea behind all of these NUS strategies is to *bias the sampling towards the Lipschitz constants of the gradients*, so that gradients that change quickly get sampled more often and gradients that change slowly get sampled less often.

We summarize the contributions of the work presented in this chapter as follows:

1. By addressing its huge memory requirements, this is the first work to apply SAG algorithm to train CRFs.

2. We also give an improved NUS strategy that adaptively estimates how frequently we should sample each data point. This reduces the number of backtracking iterations (which are expensive for CRFs unlike linear models)

3. We study the effect of the different uniform and non-uniform sampling strategies on the performance of the three algorithms: SAG, SAGA, and SAGA2.

4. We study the effect of the different step size choices on the performance of the three algorithms: SAG, SAGA, and SAGA2.

Our experiments compare the SAG algorithm with a variety of competing deterministic, stochastic, and semi-stochastic methods on benchmark data sets for four common tasks: part-of-speech tagging, named entity recognition, shallow parsing, and optical character recognition. Our results indicate that the SAG algorithm with NUS outperforms previous methods by an order of magnitude in terms of the training objective and, despite not requiring us to tune the step-size, performs as well or better than optimally tuned stochastic gradient methods in terms of the test error.

This Chapter is organized as follows. In Section 2.1, we introduce the concept of conditional random fields. We then present the related work in Section 2.2. Stochastic average

gradient (SAG), SAGA and SAGA2 methods are presented in Sections 2.3 and 2.5, respectively. Different sampling schemes and step size strategies are described Section 2.6. Experimental results are presented in Section 2.7.2, and final remarks in Section 2.8 conclude this chapter.

2.1 Conditional Random Fields

CRFs model the conditional probability of a structured output $y \in \mathcal{Y}$ (such as a sequence of labels) given an input $x \in \mathcal{X}$ (such as a sequence of words) based on features $F(x, y)$ and parameters w using

$$p(y|x, w) = \frac{\exp(w^T F(x, y))}{\sum_{y'} \exp(w^T F(x, y'))}. \tag{2.1}$$

Given n pairs $\{x_i, y_i\}$ comprising our training set, the standard approach to training the CRF is to minimize the ℓ_2-regularized negative log-likelihood,

$$\min_w f(w) = \frac{1}{n} \sum_{i=1}^{n} -\log p(y_i|x_i, w) + \frac{\lambda}{2}\|w\|^2, \tag{2.2}$$

where $\lambda > 0$ is the strength of the regularization parameter. Unfortunately, evaluating $\log p(y_i|x_i, w)$ is expensive due to the summation over all possible configurations y'. For example, in chain-structured models the forward-backward algorithm is used to compute $\log p(y_i|x_i, w)$ and its gradient. A second problem with solving (2.2) is that the number of training examples n in applications is constantly-growing, and thus we would like to use methods that only require a few passes through the data set.

2.2 Related Work

Lafferty et al. [2001] proposed an iterative scaling algorithm to solve problem (2.2), but this proved to be inferior to generic deterministic optimization strategies like the limited-memory quasi-Newton algorithm L-BFGS [Sha and Pereira, 2003, Wallach, 2002]. The bottleneck in these methods is that we must evaluate $\log p(y_i|x_i, w)$ and its gradient for all n training examples on every iteration. This is very expensive for problems where n is very large, so to deal with this problem stochastic gradient methods were examined [Finkel et al., 2008, Vishwanathan et al., 2006]. However, traditional stochastic gradient methods require $O(1/\varepsilon)$ iterations rather than the much smaller $O(\log(1/\varepsilon))$ required by deterministic methods.

There have been several attempts at improving the cost of deterministic methods or the convergence rate of stochastic methods. For example, the exponentiated gradient method of Collins et al. [2008] processes the data online and only requires $O(\log(1/\varepsilon))$ iterations to reach an accuracy of ε in terms of the dual objective. However, this does not guarantee

good performance in terms of the primal objective or the weight vector. Although this method is highly-effective if λ is very large, our experiments and the experiments of others show that the performance of online exponentiated gradient degrades substantially if a small value of λ is used (which may be required to achieve the best test error), see Collins et al. [2008, Figures 5-6 and Table 3] and Lacoste-Julien et al. [2013, Figure 1]. In contrast, SAG degrades more gracefully as λ becomes small, even achieving a convergence rate faster than classic SG methods when $\lambda = 0$ [Schmidt et al., 2017]. Lavergne et al. [2010] consider using multiple processors and vectorized computation to reduce the high iteration cost of quasi-Newton methods, but when n is enormous these methods still have a high iteration cost. Friedlander and Schmidt [2012] explore a hybrid deterministic-stochastic method that slowly grows the number of examples that are considered in order to achieve an $O(\log(1/\varepsilon))$ convergence rate with a decreased cost compared to deterministic methods.

2.3 Stochastic Average Gradient

Le Roux et al. [2012] introduce the SAG algorithm, a simple method with the low iteration cost of stochastic gradient methods but that only requires $O(\log(1/\varepsilon))$ iterations. To motivate this new algorithm, we write the classic gradient descent iteration as

$$w^{t+1} = w^t - \frac{\alpha}{n}\sum_{i=1}^{n} s_i^t, \tag{2.3}$$

where α is the step-size and at each iteration we set the 'slope' variables s_i^t to the gradient with respect to training example i at w^t, so that $s_i^t = -\nabla \log p(y_i|x_i, w^t) + \lambda w^t$. The SAG algorithm uses this same iteration, but instead of updating s_i^t for all n data points on every iterations, it simply sets $s_i^t = -\nabla \log p(y_i|x_i, w^t) + \lambda w^t$ for *one randomly chosen* data point and keeps the remaining s_i^t at their value from the previous iteration. Thus the SAG algorithm is a randomized version of the gradient algorithm where we use the gradient of each example from the last iteration where it was selected. The surprising aspect of the work of Le Roux et al. [2012] is that this simple *delayed* gradient algorithm achieves a similar convergence rate to the classic full gradient algorithm despite the iterations being n times faster. The intuition can be that the gradients stored in memory provides a better approximation than SG which just evaluates one gradient per iteration.

2.3.1 Implementation for CRFs

Unfortunately, a major problem with applying (2.3) to CRFs is the requirement to store s_i^t. While the CRF gradients $\nabla \log p(y_i|x_i, w^t)$ have a nice structure (see Section 2.3.2), s_i^t includes λw^t for some previous t, which is dense and unstructured. To get around this issue, instead of using (2.3) we use the following SAG-like update [Le Roux et al., 2012, Section 4]

$$w^{t+1} = w^t - \alpha(\frac{1}{m}\sum_{i=1}^{n} g_i^t + \lambda w^t)$$

$$= w^t - \alpha(\frac{1}{m}d + \lambda w^t)$$
$$= (1 - \alpha\lambda)w^t - \frac{\alpha}{m}d, \tag{2.4}$$

where g_i^t is the value of $-\nabla \log p(y_i|x_i, w^k)$ for the last iteration k where i was selected and d is the sum of the g_i^t over all i. Thus, this update uses the exact gradient of the regularizer and only uses an approximation for the (structured) CRF log-likelihood gradients. Since we don't yet have any information about these log-likelihoods at the start, we initialize the algorithm by setting $g_i^0 = 0$. But to compensate for this, we track the number of examples seen m, and normalize d by m in the update (instead of n). In Algorithm 1, we summarize this variant of the SAG algorithm for training CRFs.[1]

In many applications of CRFs the g_i^t are very sparse, and we would like to take advantage of this as in stochastic gradient methods. Fortunately, we can implement (2.4) without using dense vector operations by using the representation $w^t = \beta^t v^t$ for a scalar β^t and a vector v^t, and using 'lazy updates' that apply d repeatedly to an individual variable when it is needed [Le Roux et al., 2012].

Also following Le Roux et al. [2012], we set the step-size to $\alpha = 1/L$, where L is an approximation to the maximum Lipschitz constant of the gradients. This is the smallest number L such that

$$\|\nabla f_i(w) - \nabla f_i(v)\| \leq L\|w - v\|, \tag{2.5}$$

for all i, w, and v. This quantity is a bound on how fast the gradient can change as we change the weight vector. The Lipschitz constant with respect to the gradient of the regularizer is simply λ. This gives $L = L_g + \lambda$, where L_g is the Lipschitz constant of the gradient of the log-likelihood. Unfortunately, L_g depends on the covariance of the CRF and is typically too expensive to compute. To avoid this computation, as in Le Roux et al. [2012] we approximate L_g in an online fashion using the standard backtracking line-search given by Algorithm 2 [Beck and Teboulle, 2009]. The test used in this algorithm is faster than testing (2.5), since it uses function values (which only require the forward algorithm for CRFs) rather than gradient values (which require the forward and backward steps). Algorithm 2 monotonically increases L_g, but we also slowly decrease it in Algorithm 1 in order to allow the possibility that we can use a more aggressive step-size as we approach the solution.

A classic problem associated with SG methods is deciding when to terminate the iterations. The step-size must go to zero and it is therefore difficult to decide if the algorithm is close to the optimal value or if we simply require a small step-size to continue making progress. This is a problem for SAG. Since the solution is the only stationary point, we must have $\nabla f(w^t) = 0$ at the solution. Further, the value $\frac{1}{n}d + \lambda w^t$ converges to $\nabla f(w^t)$ so we can use the size of this value to decide when to stop the algorithm (although we also require that $m = n$ to avoid premature stopping before we have seen the full data set).

[1] If we solve the problem for a sequence of regularization parameters, we can obtain better performance by warm-starting g_i^0, d, and m.

Algorithm 1 SAG algorithm for training CRFs

Require: $\{x_i, y_i\}$, λ, w, δ

$m \leftarrow 0$, $g_i \leftarrow 0$ for $i = 1, 2, \ldots, n$
$d \leftarrow 0$, $L_g \leftarrow 1$
while $m < n$ and $\|\frac{1}{n}d + \lambda w\|_\infty \geq \delta$ **do**
  Sample i from $\{1, 2, \ldots, n\}$
  $f \leftarrow -\log p(y_i|x_i, w)$
  $g \leftarrow -\nabla \log p(y_i|x_i, w)$
  if this is the first time we sampled i **then**
    $m \leftarrow m + 1$
  end if
  Subtract old gradient g_i, add new gradient g:
  $d \leftarrow d - g_i + g$
  Replace old gradient of example i:
  $g_i \leftarrow g$
  if $\|g_i\|^2 > 10^{-8}$ **then**
    $L_g \leftarrow$lineSearch$(x_i, y_i, f, g_i, w, L_g)$
  end if
  $\alpha \leftarrow 1/(L_g + \lambda)$
  $w \leftarrow (1 - \alpha\lambda)w - \frac{\alpha}{m}d$
  $L_g \leftarrow L_g \cdot 2^{-1/n}$
end while

2.3.2 Reducing the Memory Requirements

The modified update in the previous section drastically reduces the memory requirements of applying SAG to CRFs. But even if the gradients g_i^t are not sparse, we can often reduce the memory requirements of Algorithm 1 because it is known that the CRF gradients only depend on w through marginals of the features. Specifically, the gradient of the log-likelihood under model (2.1) with respect to feature j is given by

$$\begin{aligned}\nabla_j \log p(y|x, w) &= F_j(x, y) - \frac{\sum_{y'} \exp(F(x,y'))}{\sum_{y'} F(x,y')} F_j(x, y') \\ &= F_j(x, y) - \sum_{y'} p(y'|x, w) F_j(x, y') \\ &= F_j(x, y) - \mathbb{E}_{y'|x,w}[F_j(x, y')]\end{aligned}$$

Typically, each feature j only depends on a small 'part' of y. For example, we typically include features of the form $F_j(x, y) = F(x)\mathbb{I}[y_k = s]$ for some function F, where k is an element of y and s is a discrete state that y_k can take. In this case, the gradient can be written in terms of the marginal probability of element y_k taking state s,

$$\nabla_j \log p(y|x, w) = F(x)\mathbb{I}[y_k = s] - \mathbb{E}_{y'|x,w}[F(x)\mathbb{I}[y_k = s]]$$

Algorithm 2 Lipschitz line-search algorithm

Require: x_i, y_i, f, g_i, w, L_g.

$f' = -\log p(y_i|x_i, w - \frac{1}{L_g} g_i)$
while $f' \geq f - \frac{1}{2L_g}\|g_i\|^2$ **do**
  $L_g = 2L_g$
  $f' = -\log p(y_i|x_i, w - \frac{1}{L_g} g_i)$
end while
return L_g.

$$= F(x)(\mathbb{I}[y_k = s] - \mathbb{E}_{y'|x,w}[\mathbb{I}[y_k = s])$$
$$= F(x)(\mathbb{I}[y_k = s] - p(y_k = s|x, w)).$$

Notice that Algorithm 1 only depends on the old gradient through its difference with the new gradient (line 10), which in this example gives

$$\nabla_j \log p(y|x, w) - \nabla_j \log p(y|x, w_{\text{old}}) =$$
$$F(x)(p(y_k = s|x, w_{\text{old}}) - p(y_k = s|x, w)),$$

where w is the current parameter vector and w_{old} is the old parameter vector. Thus, to perform this calculation the only thing we need to know about w_{old} is the unary marginal $p(y_k = s|x, w_{\text{old}})$, which will be *shared* across features that only depend on the event that $y_k = s$. Similarly, features that depend on pairs of values in y will need to store pairwise marginals, $p(y_k = s, y'_k = s'|x, w_{\text{old}})$. For general pairwise graphical model structures, the memory requirements to store these marginals will thus be $O(VK + EK^2)$, where V is the number of vertices and E is the number of edges. This can be an enormous reduction since *it does not depend on the number of features*. Further, since computing these marginals is a by-product of computing the gradient, this potentially-enormous reduction in the memory requirements comes at no extra computational cost.

2.4 Non-Uniform Sampling

As pointed out in the introduction to this Chapter, non uniform sampling (NUS) can help speed up the convergence of randomized algorithm. In this work, we propose a NUS scheme that depends on the Lipschitz constant L. Specifically, we maintain a Lipschitz constant L_i for each training example i and, instead of the usual sampling strategy $p_i = 1/n$, we bias towards the distribution $p_i = L_i / \sum_j L_j$. Similar to (2.5), L_i is the smallest number such that

$$\|\nabla f_i(w) - \nabla f_i(v)\| \leq L_i \|w - v\|, \tag{2.6}$$

for all w and v, where $\nabla f_i(w)$ is the gradient with respect to example i . In these various contexts, NUS allows us to improve the dependence on the values L_i in the convergence

rate, since the NUS methods depend on $\overline{L} = (1/n)\sum_j L_j$, which may be substantially smaller than the usual dependence on $L = \max_j\{L_j\}$. Schmidt et al. [2017] argue that faster convergence rates might be achieved with NUS for SAG since it allows a larger step size α that depends on $\overline{L}$ instead of L.[2] In this work, we consider 2 possible NUS techniques:

- The scheme for SAG proposed by Schmidt et al. [2017, Section 5.5] uses a fairly complicated adaptive NUS scheme and step-size, but the key ingredient is estimating each constant L_i using Algorithm 2. Our experiments show this method already improves on state of the art methods for training CRFs by a substantial margin, but we found we could obtain improved performance for training CRFs using a simple NUS scheme for SAG.

- As in Needell et al. [2014], with probability 0.5 choose i uniformly and with probability 0.5 sample i with probability $L_i/(\sum_j L_j)$ (restricted to the examples we have previously seen).[3] We also use a step-size of $\alpha = \frac{1}{2}\left(1/L + 1/\overline{L}\right)$, since the faster convergence rate with NUS is due to the ability to use a larger step-size than $1/L$. This simple step-size and sampling scheme contrasts with the more complicated choices described by Schmidt et al. [2017, Section 5.5], that make the degree of non-uniformity grow with the number of examples seen m. This prior work initializes each L_i to 1, and updates L_i to $0.5L_i$ each subsequent time an example is chosen. In the context of CRFs, this leads to a large number of expensive backtracking iterations. To avoid this, we initialize L_i with $0.5\overline{L}$ the first time an example is chosen, and decrease L_i to $0.9L_i$ each time it is subsequently chosen.

2.5 Stochastic Average Gradient Variants

Several works have introduced variants of the SAG algorithm. In the second part of the experiments we study the effect of the sampling scheme and the step-size choices on the performance of 2 algorithms: SAGA and SAGA2. It is important to note that the memory-savings introduced in Section 2.3.2 also works for these SAG variants.

2.5.1 SAGA

This algorithm was introduced by Defazio et al. [2014a]. SAGA provides a simpler theoretical analysis than SAG. The key to this improvement is that SAGA uses an unbiased estimate of the gradient as compared to SAG which uses a biased estimate. SAGA is described in Algorithm 3. It can be noticed that the only difference between SAG and SAGA is how and when the weights are updated and the weighting of the terms. The intuition

[2] An interesting difference between the SAG update with NUS and NUS for stochastic gradient methods is that the SAG update does not seem to need to decrease the step-size for frequently-sampled examples (since the SAG update does not rely on using an unbiased gradient estimate).

[3] Needell et al. [2014] only analyze the basic stochastic gradient method and thus require $O(1/\varepsilon)$ iterations.

Algorithm 3 SAGA algorithm for training CRFs

Require: $\{x_i, y_i\}$, λ, w, δ

$m = 0$, $g_i = 0$ for $i = 1, 2, \ldots, n$
$d = 0$, $L_g = 1$
while $m < n$ and $\|\frac{1}{n}d + \lambda w\|_\infty \geq \delta$ **do**
  Sample i from $\{1, 2, \ldots, n\}$ according to the sampling scheme.
  $f = -\log p(y_i|x_i, w)$
  $g = -\nabla \log p(y_i|x_i, w)$
  if this is the first time we sampled i **then**
    $m = m + 1$
  end if
  if L_g is needed **then**
    if $\|g_i\|^2 > 10^{-8}$ **then**
      $L_g =$lineSearch$(x_i, y_i, f, g_i, w, L_g)$
    end if
  end if
  Pick the step size α.
  For NUS, Calculate bias correction factor:
  $\beta = mL_i / \sum_j L_j$
  $w = (1 - \alpha\lambda)w - \frac{\alpha}{\beta}((g - g_i) + \frac{1}{m}d)$
  $d = d + g - g_i$
  $g_i = g$
  $L_g = L_g \cdot 2^{-1/n}$
end while

behind this update is that it allows for an unbiased estimate of the gradient in contrast to the biased estimate calculated by SAG.

2.5.2 SAGA2

In this chapter, we suggest a further modification to SAGA. We refer to this algorithm as SAGA2. The algorithm is described in Algorithm 4. SAGA2 is similar to SAGA in the weight updates. However, when the gradients are updated another training example is sampled uniformly. This makes analyzing NUS easier for SAGA2 as it allows showing a faster convergence rate that depends on the average Lipschitz constant rather than the maximum. The full details of SAGA2 are described in Schmidt et al. [2015].

2.6 SAG Implementation Details

Many works have explored theoretical properties of methods like SAG, but there are fewer works trying to find the best empirical choices for this method. In this section we discuss

Algorithm 4 SAGA2 algorithm for training CRFs

Require: $\{x_i, y_i\}$, λ, w, δ

$m = 0$, $g_i = 0$ for $i = 1, 2, \ldots, n$
$d = 0$, $L_g = 1$
while $m < n$ and $\|\frac{1}{n}d + \lambda w\|_\infty \geq \delta$ **do**
  Sample i from $\{1, 2, \ldots, n\}$ according to the sampling scheme.
  $f = -\log p(y_i|x_i, w)$
  $g = -\nabla \log p(y_i|x_i, w)$
  if this is the first time we sampled i **then**
    $m = m + 1$
  end if
  if L_g is needed **then**
    if $\|g_i\|^2 > 10^{-8}$ **then**
      $L_g =$lineSearch$(x_i, y_i, f, g_i, w, L_g)$
    end if
  end if
  Pick the step size α.
  For NUS, Calculate bias correction factor:
  $\beta = mL_i / \sum_j L_j$
  $w = (1 - \alpha\lambda)w - \frac{\alpha}{\beta}((g - g_i) + \frac{1}{m}d)$
  Sample i from $\{1, 2, \ldots, n\}$ Uniformly.
  $f = -\log p(y_i|x_i, w)$
  $g = -\nabla \log p(y_i|x_i, w)$
  $d = d + g - g_i$
  $g_i = g$
  $L_g = L_g \cdot 2^{-1/n}$
end while

some of the variations that are possible and introduce new variations that work better. The performance of SAG, SAGA, and SAGA2 depend on the choice of 2 main factors: the sampling strategy and the step-size choice. This is the first work that presents a systematic comparison of the different combinations of possible choices for these factors.

2.6.1 Effect of Sampling Strategies

The following presents the possible choices for the sampling strategies:

1. Uniform sampling (U): the training sample at each iteration is chosen randomly according to a uniform distribution.

2. Random permutation (RP): This method pick an order for the training examples randomly every epoch, and this order is fixed throughout the epoch. Several papers

suggested that RP is better than cyclic and uniform sampling [Defazio et al., 2014b, Recht and Ré, 2012].

3. Cyclic: This method process the training examples one by one with a fixed order $(1 : n)$.

4. Cyclic with two orders (Cyclic2orders): this method uses 2 fixed orders of the training examples and the algorithm keeps alternating between them. The method is trying to simulate the RP method. However, the advantage here is that the data can be stored twice with the different orders and the algorithm can access the memory sequentially. This allows for better memory management than RP. However, the price to pay is storing the data twice.

5. Non-uniform Sampling (NUS):

   Two Non-uniform sampling schemes were tried:

   - Pure Lipschitz (PL) Sampling: Schmidt et al. [2017] proposed this heuristic NUS scheme: with probability $(n - m)/n$, we sample a previously unseen example uniformly, and with probability m/n we sample a previously seen example with probability $L_i / \sum_j L_j$ to pick example i. In this algorithm the L_i are initialized to one, we use Algorithm 2 to update the L_i when a data point is sampled, and when we re-visit an example we double L_i.
   - Mixed Sampling (MS): This is the same as PL. The only difference is that with probability 0.5 we sample a previously unseen example uniformly, and with probability 0.5 we sample a previously seen example proportional to $L_i / \sum_j L_j$.

To summarize, the sampling schemes that were tried are: 'U', 'RP', 'Cyclic', 'Cyclic2orders', 'PL', and 'MS'.

2.6.2 Effect of Step Size

Another important tuning factor in the optimization algorithm is the step-size. If it is too small, the algorithm may take too many iterations to converge to the right solution. On the other hand, if the step size is too large, the algorithm may not converge at all. The goal of these experiments is to try to find (empirically) the best step size for the SAG, SAGA, and SAGA2 algorithms. This will provide more understanding of why certain choices work while others do not. The methods for choosing the step sizes are:

1. Constant (const): The step size is constant for each iteration. In order to pick the best step size, a grid search is performed. If the smallest step size or the largest step size is chosen as the best value, the grid is extended to try more values. This is not a practical strategy. However, it provides an estimate of the best possible performance.

2. Lmean: the step size is chosen according to the mean of the Lipschitz constants as $\alpha = 1/L_{\text{mean}}$, where L_{mean} is the average of the L_i. The intuition behind this choice is if L_{mean} is small, then the rate of the change of the gradient is small, so a large step size can be chosen. This is the step size used in the theory of SAGA2.

3. Lmax: the step size is chosen according to the maximum of the Lipschitz constants as $\alpha = 1/L_{\text{max}}$. This is the step size used in the theory of SAG and SAGA.

4. Hedge: this method uses the average of the two previous step sizes, so $\alpha = \frac{1}{2L_{\text{max}}} + \frac{1}{2L_{\text{mean}}}$.

5. optimum (opt): the step size is chosen as $\alpha = \frac{2}{(L_{\text{max}}+n\mu)}$, where μ is the strong convexity parameter, and it is approximated by $\mu = \lambda$ for this problem. This step size is inspired by the optimal step size for $\mu-$strongly convex function with constant step size [Nesterov, 2004]

6. averagehedgeopt1: the step size is chosen as $\alpha = \frac{2}{(0.5(L_{\text{mean}}+L_{\text{max}})+\mu)}$

7. averagehedgeopt2: the step size is chosen as $\alpha = 0.5(\frac{2}{(L_{\text{max}}+\mu)} + \frac{2}{(L_{\text{mean}}+\mu)})$.

   The last two choices of the step size are different variations of the opt and hedge choices.

To summarize, the step sizes that were tried are: ‘const’, ‘Lmean’, ‘Lmax’, ‘hedge’, ‘opt’, ‘averagehedgeopt1’ and ‘averagehedgeopt2’.

2.7 Experiments

In order to evaluate the performance of the various methods, we performed the experiments on different tasks with the following datasets:

1. OCR: This dataset was proposed by Taskar et al. [2003] for optical character recognition (OCR). The optimal character recognition dataset labels the letters in images of words. The goal is to recognize the letters given the images.

2. CoNLL: This is the CoNLL-2000 shallow parse chunking dataset.[4] Chunking yields a lightweight parse of a sentence; it segments a sentence into syntactic chunks by tagging each sentence token with a chunk tag corresponding to its constituent type (e.g., ‘NP’, ‘VP’, etc.) and location (e.g., beginning, inside, ending, or outside any constituent).

3. NER: This is the CoNLL-2002 Dutch named-entity recognition (NER) dataset.[5] For the NER task, the goal is to identify named entities and correctly classify them as

[4] http://www.cnts.ua.ac.be/conll2000/chunking

[5] http://www.cnts.ua.ac.be/conll2002/ner

persons, organizations, locations, times or quantities. We use standard n-gram and part of speech (POS) tag features, as well as word shape features over the case of the characters in the token.

4. POS: This is the part of speech tagging task using the Penn Treebank Wall Street Journal data (POS-WSJ). We use standard n-gram and part of speech (POS) tag features [Sha and Pereira, 2003].

Table 2.1 summarizes the information about the datasets used.

Table 2.1: The datasets used to conduct the experiments

Name	**Number of training examples**	$f_{\textbf{opt}}$	**Number of labels**	**Number of features**	**Dim**(w)
OCR	6251	1.5×10^4	27	128	4,082
CoNLL-2000	8936	8.9×10^3	24	19	1,643,026
CoNLL-2002	15806	5.1×10^3	10	20	2,798,955
POS-WSJ	38219	7.2×10^4	46	13	8,572,770

2.7.1 Stochastic Average Gradient for Conditional Random Fields results

To quantify the memory savings given by the choices in Section 2.3, below we report the size of the memory required for these datasets under different memory-saving strategies divided by the *Absolute* memory required by the naive SAG algorithm. *Sparse* refers to only storing non-zero gradient values, *Marginals* refers to storing all unary and pairwise marginals, and *Mixed* refers to storing node marginals and the gradient with respect to pairwise features.

Dataset	Sparse	Marginals	Mixed	Absolute (GB)
OCR	7.8×10^{-1}	1.1×10^{0}	2.1×10^{-1}	0.2
CoNLL-2000	4.8×10^{-3}	7.0×10^{-3}	6.1×10^{-4}	110
CoNLL-2002	6.4×10^{-4}	3.8×10^{-4}	7.0×10^{-5}	330
POS-WJ	1.3×10^{-3}	5.5×10^{-3}	3.6×10^{-4}	2440

On these datasets we compared the performance of a set of competitive methods, including five variants on classic stochastic gradient methods:

1. *Pegasos* which is a standard stochastic gradient method with a step-size of $\alpha = \eta/\lambda t$ on iteration t [Shalev-Shwartz et al., 2011],[6]

2. a basic stochastic gradient (*SG*) method where we use a constant $\alpha = \eta$,

[6]We also tested *Pegasos* with averaging but it always performed worse than the non-averaged version.

3. an averaged stochastic gradient (*ASG*) method where we use a constant step-size $\alpha = \eta$ and average the iterations,[7]

4. *AdaGrad* where we use the per-variable $\alpha_j = \eta/(\delta + \sqrt{\sum_{i=1}^{t} \nabla_j \log p(y_i|x_i, w^i)^2})$ and the proximal-step with respect to the ℓ_2-regularizer [Duchi et al., 2011]

5. stochastic meta-descent (*SMD*) where we initialize with $\alpha_j = \eta$ and dynamically update the step-size [Vishwanathan et al., 2006].[8]

Since setting the step-size is a notoriously hard problem when applying stochastic gradient methods, we let these classic stochastic gradient methods cheat by choosing the η which gives the best performance among powers of 10 on the training data (for SMD we additionally tested the four choices among the paper and associated code of Vishwanathan et al. [2006], and we found $\delta = 1$ worked well for *AdaGrad*).

Our comparisons also included a deterministic *L-BFGS* algorithm and the *Hybrid* L-BFGS/stochastic algorithm of Friedlander and Schmidt [2012]. We also included the online exponentiated gradient *OEG* method of [Collins et al., 2008], using the heuristics in their code. Finally, we included the *SAG* algorithm as described in Section 2.3, the *SAG-NUS* (SAG-PL) variant of Schmidt et al. [2017], and our proposed *SAG-NUS** (SAG-MS) strategy from Section 2.6.1.[9]

Figure 2.1 shows the result of our experiments on the training objective and Figure 2.2 shows the result of tracking the test error. Here we measure the number of 'effective passes', meaning $(1/n)$ times the number of times we performed the bottleneck operation of computing $\log p(y_i|x_i, w)$ and its gradient. This is an implementation-independent way to compare the convergence of the different algorithms (whose runtimes differ only by a small constant), but we have included the performance in terms of runtime in Figures 2.4 and 2.5. For the different *SAG* methods that use a line-search we count the extra 'forward' operations used by the line-search as full evaluations of $\log p(y_i|x_i, w)$ and its gradient, even though these operations are cheaper because they do not require the backward pass nor computing the gradient. In these experiments we used $\lambda = 1/n$, which yields a value close to the optimal test error across all data sets. To approximate the unique minimum value we ran L-BFGS for up to 1000 iterations, which always gave a value of w satisfying $\|\nabla f(w)\|_\infty \leq 1.4 \times 10^{-7}$, indicating that this is a very accurate approximation of the true solution. In the test error plots, we have excluded the *SAG* and *SAG-NUS* methods to keep the plots interpretable.

In the test error plots, we have also plotted as dotted lines the performance of the classic stochastic gradient methods when the second-best step-size is used.

[7] We also tested *SG* and *ASG* with decreasing step-sizes of either $\alpha_t = \eta/\sqrt{t}$ or $\alpha_t = \eta/(\delta + t)$, but these gave worse performance than using a constant step size.

[8] Because of the extra implementation effort required to implement it efficiently, we did not test SMD the POS dataset.

[9] We also tested *SG* with the proposed NUS scheme, but the performance was similar to the regular SG method. This is consistent with the analysis of Needell et al. [2014, Corollary 3.1] showing that NUS for regular *SG* only improves the non-dominant term.

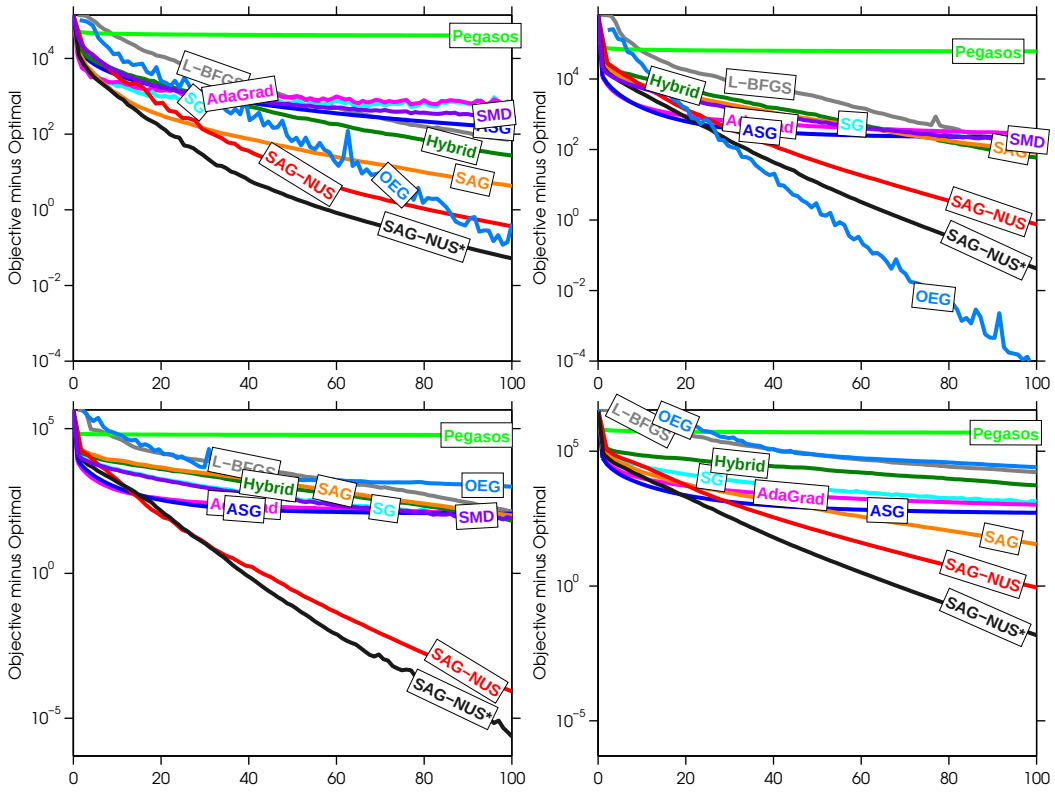


Figure 2.1: Objective minus optimal objective value against effective number of passes for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

We make several observations based on these experiments:

- *SG* outperformed *Pegasos*. Pegasos is known to move exponentially away from the solution in the early iterations [Bach and Moulines, 2011], meaning that $\|w^t - w^*\| \geq \rho^t \|w^0 - w^*\|$ for some $\rho > 1$, while SG moves exponentially towards the solution ($\rho < 1$) in the early iterations [Nedic and Bertsekas, 2000].
- *ASG* outperformed *AdaGrad* and *SMD* (in addition to *SG*). ASG methods are known to achieve the same asymptotic efficiency as an optimal stochastic Newton method [Polyak and Juditsky, 1992], while AdaGrad and SMD can be viewed as approximations to a stochastic Newton method. Vishwanathan et al. [2006] did not compare to ASG, because applying ASG to large/sparse data requires the recursion of Xu [2010].
- *Hybrid* outperformed *L-BFGS*. The hybrid algorithm processes fewer data points in the early iterations, leading to cheaper iterations.
- *OEG* performed extremely well on two of the data sets (in terms of the training ob-

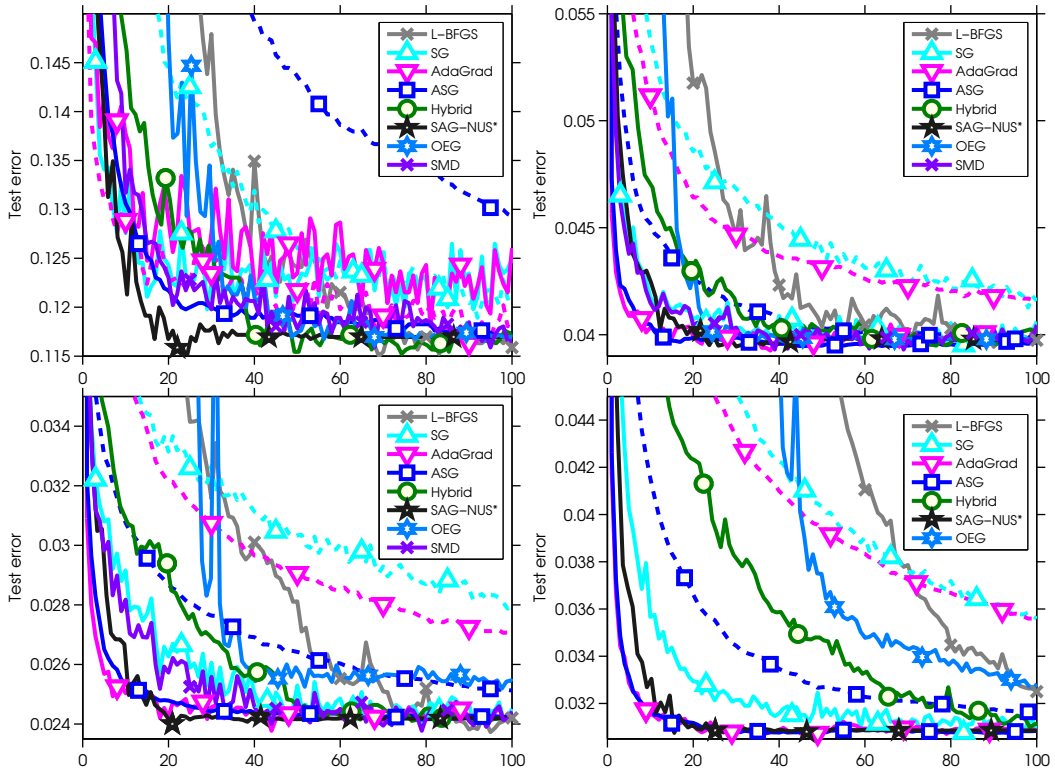


Figure 2.2: Test error against effective number of passes for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ. The dotted lines show the performance of the classic stochastic gradient methods when the optimal step-size is not used. *Note that the performance of all classic stochastic gradient methods is much worse when the optimal step-size is not used*, whereas the SAG methods have an adaptive step-size so are not sensitive to this choice.

jective), but much worse on the other two.

- None of the three competitive algorithms *ASG*/*Hybrid*/*SAG* dominated the others: the relative ranks of these methods changed based on the data set and whether we could choose the optimal step-size.
- Both *SAG-NUS* methods outperform all other methods by a substantial margin based on the training objective, and are always among the best methods in terms of the test error. Further, our proposed *SAG-NUS** always performed as well or better than *SAG-NUS*.

On three of the four data sets, the best classic stochastic gradient methods (*AdaGrad* and *ASG*) seem to reach the optimal test error with a similar speed to the *SAG-NUS** method,

although they require many passes to reach the optimal test error on the OCR data. Further, we see that the good test error performance of the *AdaGrad* and *ASG* methods *is very sensitive to choosing the optimal step-size*, as the methods perform much worse if we don't use the optimal step-size (dashed lines in Figure 2.2). In contrast, SAG uses an adaptive step-size and has virtually identical performance even if the initial value of L_g is too small by several orders of magnitude (the line-search quickly increases L_g to a reasonable value on the first training example, so the dashed black line in Figure 2.2 would be on top of the solid line).

Test Error Plots for All Methods

In Figure 2.3, we plot the test error of all methods. Note that Pegasos does not appear on the plot (despite being in the legend) because its values exceed the maximum plotted values. In these plots we see that the SAG-NUS methods perform similarly to the best among the optimally-tuned stochastic gradient methods in terms of test error, despite the lack of tuning required to apply these methods.

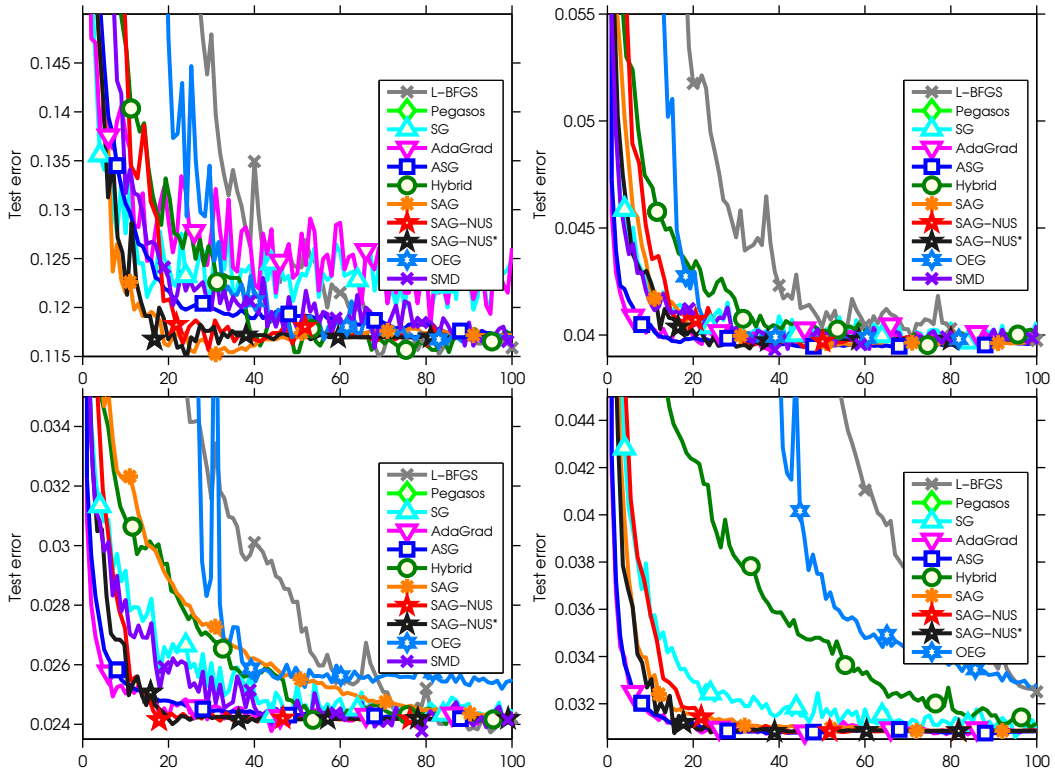


Figure 2.3: Test error against effective number of passes for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

Runtime Plots

In the previous figures, we plot the performance against the effective number of passes as an implementation-independent way of comparing the different algorithms. In all cases except OEG and SMD, we implemented a C version of the method and also compared the running times of our different implementations. This ties the results to the hardware used to perform the experiments, and thus says little about the runtime in different hardware settings, but does show the practical performance of the methods in this particular setting. We plot the training objective against runtime in Figure 2.4 and the testing objective in Figure 2.5. In general, the runtime plots show the exact same trends as the plots against the effective number of passes. However, we note several small differences:

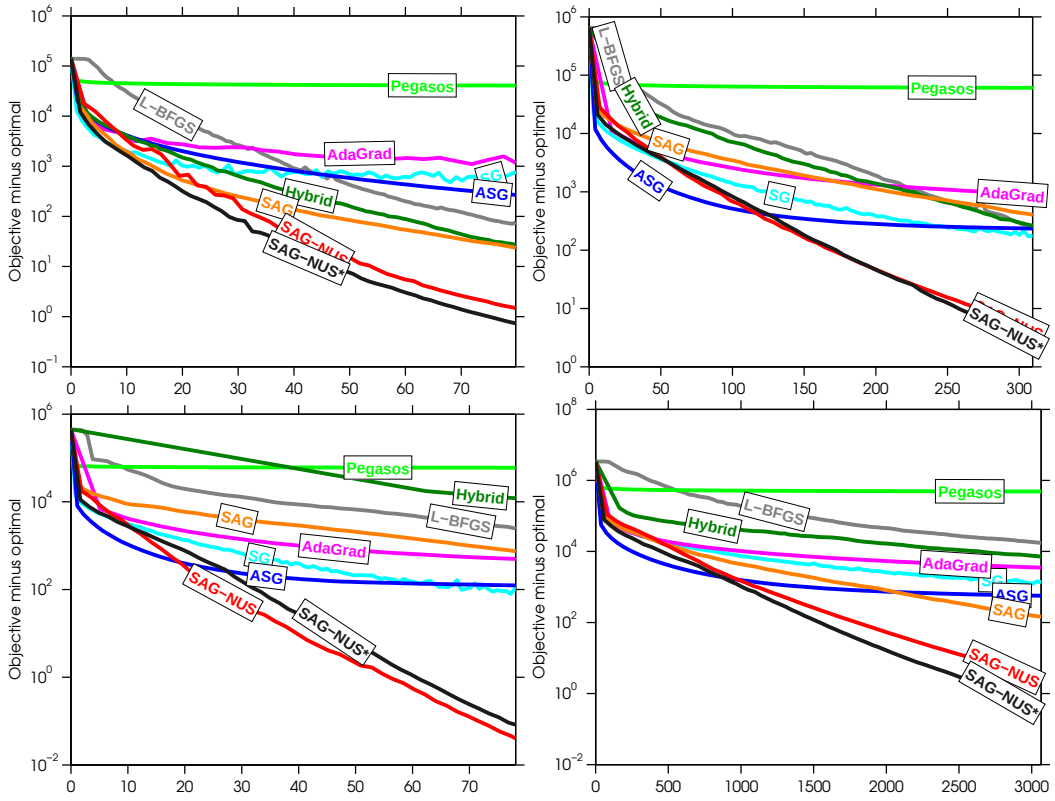


Figure 2.4: Objective minus optimal objective value against time for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

- *AdaGrad* performs slightly worse in terms of runtime, and was always worse than the basic *SG* method. This seems to be due to the extra square root operators needed to implement the method.

- *Hybrid* performs slightly worse in terms of runtime, although it was still faster than the *L-BFGS* method. This seems to be due to the higher cost of applying the L-BFGS update when the batch size is small.

- *SAG* performs slightly worse in terms of runtime, though it remains among the other top performing methods *Hybrid* and *ASG*. This seems to be due to the higher cost of the memory update associated with the algorithm.

- Although both *SAG-NUS* methods still dominate all other methods by a substantial margin, the performance of the new *SAG-NUS** and the existing *SAG-NUS* is much closer in terms of runtime. This seems to be because, although the *SAG-NUS* method does much more backtracking than *SAG-NUS**, these backtracking steps are much cheaper because they only require the forward pass of the forward-backward algorithm. If we compared these two algorithms under more complicated inference schemes, we would expect the advantage of *SAG-NUS** to appear in the runtime, too.

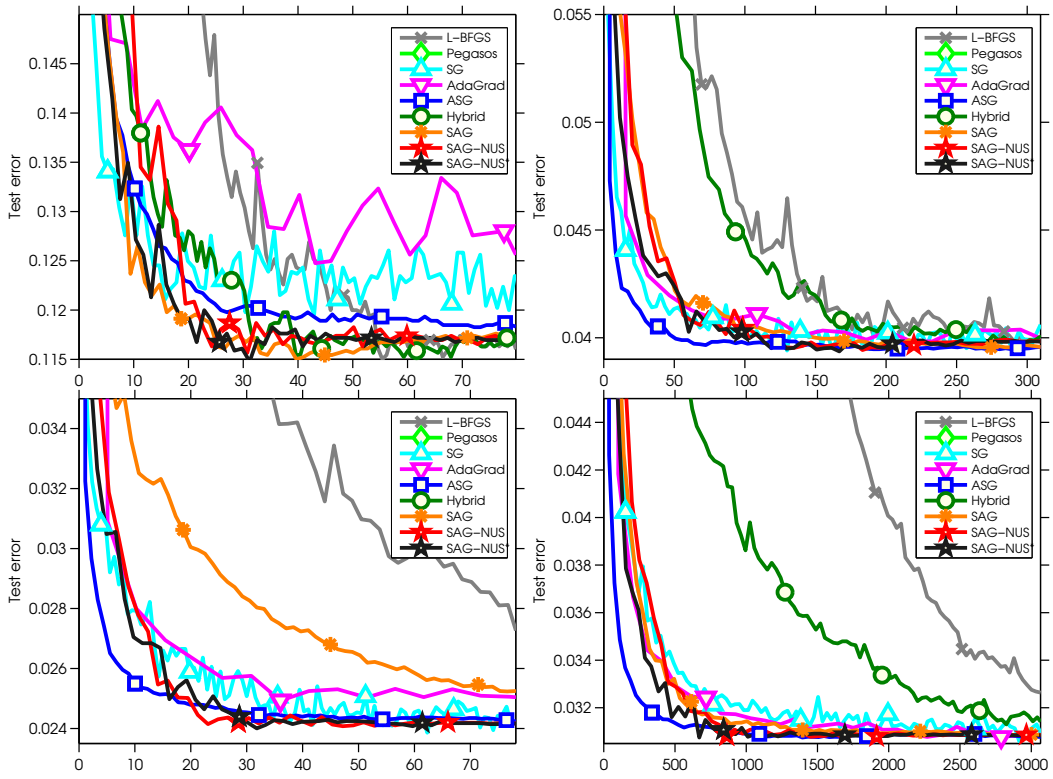


Figure 2.5: Test error against time for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

Table 2.2: Summary of the conducted experiments

Algorithm	**Sampling Schemes**	**Step Size**
SAG	U	const
SAGA	PL	Lmean
SAGA2	MS	Lmax
-	RP	hedge
-	Cyclic	opt
-	Cyclic2orders	averagehedgeopt1
-	-	averagehedgeopt2

2.7.2 Sampling Schemes and Step-size Experiments

The second part of the experiments studies the performance of the SAG, SAGA, and SAGA2 algorithms for different choices of the sampling schemes and the step sizes.

Effect of the step size and the sampling scheme

Table 2.2 summarizes the experiments that were performed. The performance of the three algorithms on the OCR dataset is presented in Figures 2.6, 2.7, and 2.8. The x-axis shows the different step size methods and the bar colors represent the sampling scheme. The height of the bar represents the value of the objective after the final iteration as compared to the optimal objective $\left|f_{\text{opt}} - f_{\text{final}}\right|$. Missing values correspond to the methods that diverge. The algorithm is considered to diverge if ($\left|f_{\text{opt}} - f_{\text{final}}\right| \geq |f_{\text{initial}} - f_{\text{final}}|$). The following summarizes the conclusions from the conducted experiments:

- From Figure 2.6, it is clear that SAG only works for PL, MS, and U sampling. However, U sampling requires certain choices of the step size to perform well.
- None of the cyclic, RP and cyclic2orders sampling work for SAG no matter which step size is chosen.
- SAGA seems to work with more choices of the sampling schemes and the step sizes as can be noticed in Figure 2.7. Cyclic ordering does not work for any choice of the step size (it is important to note that this may be due to the specific order used).
- RP and cyclic2orders work for several choices of the step size. In fact, these two sampling schemes provides the best performance for SAGA when used with the hedge step size. As discussed in section 2.6.2, cyclic2orders may be preferred over RP due to better memory management.
- From Figure 2.8, we can conclude that SAGA2 works for most of the sampling schemes and the step sizes similar to SAGA. It is important to note that SAGA2

picks a random training sample uniformly to update the gradient. So even for cyclic, there is a random part introduced in the algorithm. This allows cyclic sampling to work for SAGA2. Although SAGA2 requires two training samples at each iteration, it can compensate for this by taking larger steps ($\alpha = 1/L_{\text{mean}}$). This allows SAGA2 to compete with SAG and SAGA2.

Based on the initial results from the smallest dataset (OCR), we decided to perform the experiments on the larger datasets only for the best methods. Figure 2.9 presents the results for the best five configurations of SAG for all the datasets. The legend on the figure represents the algorithm used, the sampling scheme, and the step size. For example, "SAG-U-opt" is the method that uses the SAG algorithm with uniform sampling and the opt step size.

From Figures 2.9, it is clear that MS sampling with the hedge step size is the best choice for SAG, especially for the NER task which outperforms all the other configurations. Figure 2.10 represents the test errors for the best SAG configurations. These confirm that MS sampling with the hedge step size is the best choice for SAG. One of the main advantages of SAG is that the best method is the same for all the datasets which is SAG with MS and hedge.

As for SAGA, Figures 2.11 and 2.12 present the results of the best methods on all four datasets. Although SAGA works for more choices of the step size and the sampling schemes, there is no clear winner on all the data sets. The closest configuration to achieving the best performance on all the datasets is SAGA with PL sampling and the hedge step size. This can be a problem as the algorithm may require changes to work better on different datasets. It is interesting to see that the hedge step size is providing great performance on NER as it did with SAG. Furthermore, RP and cyclic2orders sampling did well on the OCR and the POS tasks. However, they did not work on the NER task in terms of the training objective.

Finally for SAGA2, the results are presented in Figures 2.13 and 2.14. We compare SAGA2 with PL sampling and the Lmean step size to the best methods of SAG and SAGA. This choice was made for SAGA2 because we provided the theory behind it in Schmidt et al. [2015]. SAGA2 is doing as well as the best SAG and SAGA methods on the OCR and POS tasks. However, for the text chunking and the NER tasks, SAG and SAGA performed much better than SAGA2. Even the hedge step size could not fix the performance of SAGA2 on the NER task as it did for SAGA. This can be due to the large step size, but it requires further investigation.

It is important to note that we conducted a lot of experiments in order to find the best methods. Some of the experiments that did not improve the performance are not reported here such as: non-uniform sampling for stochastic gradient method.

2.8 Discussion

Applying SAG to CRFs is a very challenging task due to the large memory requirements of SAG. We have presented the first implementation of SAG for CRFs. Our experiments show that performance gain from using SAG is huge, especially with non-uniform sampling.

As for the implementation details, our experiments show that SAG with MS sampling and the hedge step size is the best method. However, the proof of convergence for SAG with NUS is still missing. On the other hand, there are theoretical guarantees that SAGA will converge with NUS [Schmidt et al., 2015]. However, the configurations of SAGA (sampling scheme and step size) may need to be changed depending on the task. As for SAGA2, the theoretical guarantees for convergence were provided in Schmidt et al. [2015]. However, we notice that SAGA2 with PL sampling and Lmean step size diverge on the NER task. This requires further investigation. As for the step sizes, we observe that hedge provides the best performance for SAG and SAGA on all the data sets. Lmax seems to produce a small step size, while averagehedgeopt1 and averagehedgeopt2 turned out to be bad choices.

We have shown that SAGA and SAGA2 works with more configurations when compared to SAG. However, when using the MS non-uniform sampling strategy with the hedge step size, SAG provides the best performance on all the tasks considered, especially for NER. Since the analysis of SAG is hard, SAGA with PL sampling and hedge step size may provide a promising alternative. To facilitate reproducible research, the code is available at `http://www.cs.ubc.ca/~schmidtm/Software/SAG4CRF.html`.

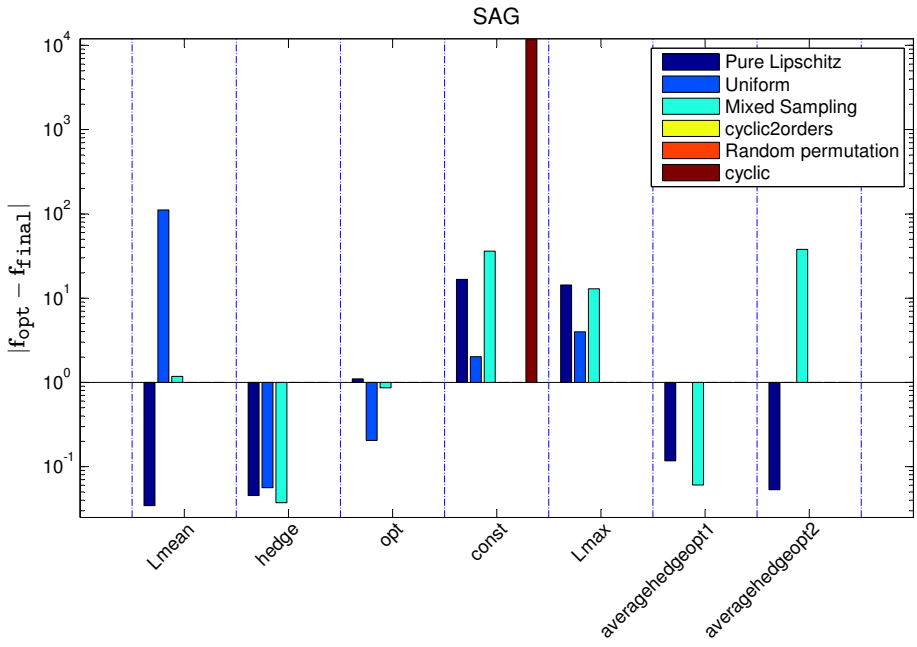


Figure 2.6: SAG Objective minus optimal objective value for OCR data for different choices of the step size and sampling scheme. The lower values are better and the missing columns represents the methods that diverge. PL with Lmean is the best.

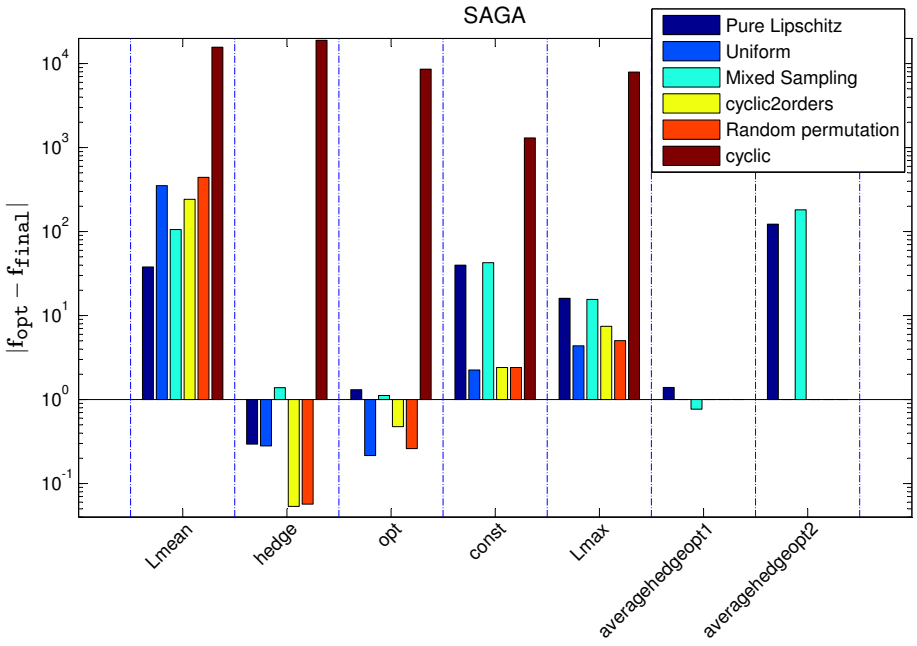


Figure 2.7: SAGA Objective minus optimal objective value for OCR data for different choices of the step size and sampling scheme. The lower values are better and the missing columns represents the methods that diverge. Hedge with wither Cyclic or RP are the best

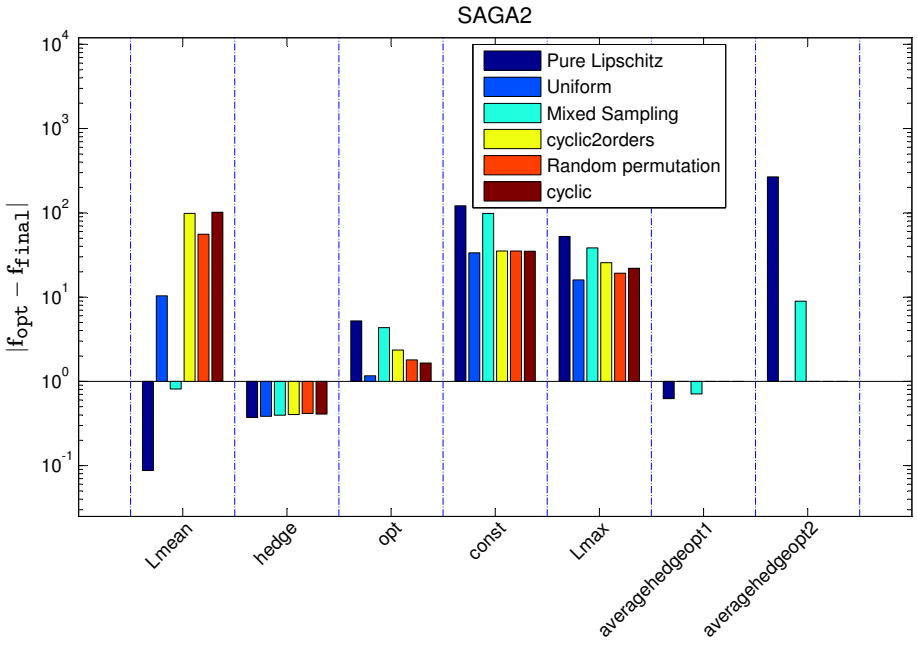


Figure 2.8: SAGA2 Objective minus optimal objective value for OCR data for different choices of the step size and sampling scheme. The lower values are better and the missing columns represents the methods that diverge. PL with Lmean is the best.

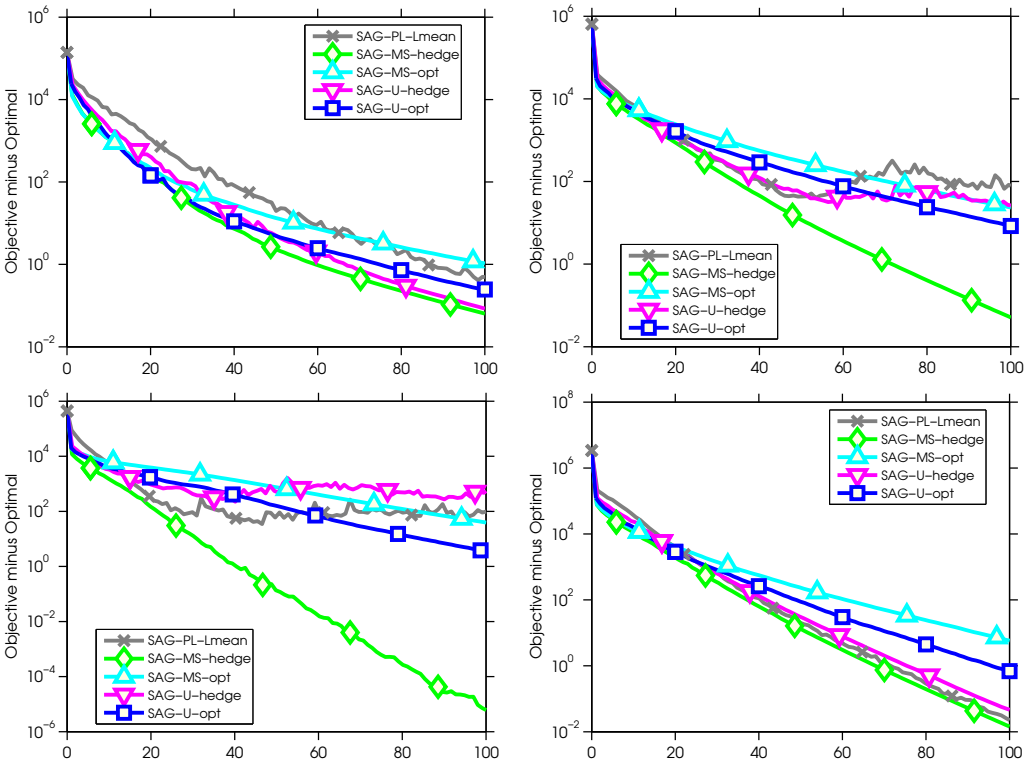


Figure 2.9: Objective minus optimal objective value against effective number of passes for different SAG optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

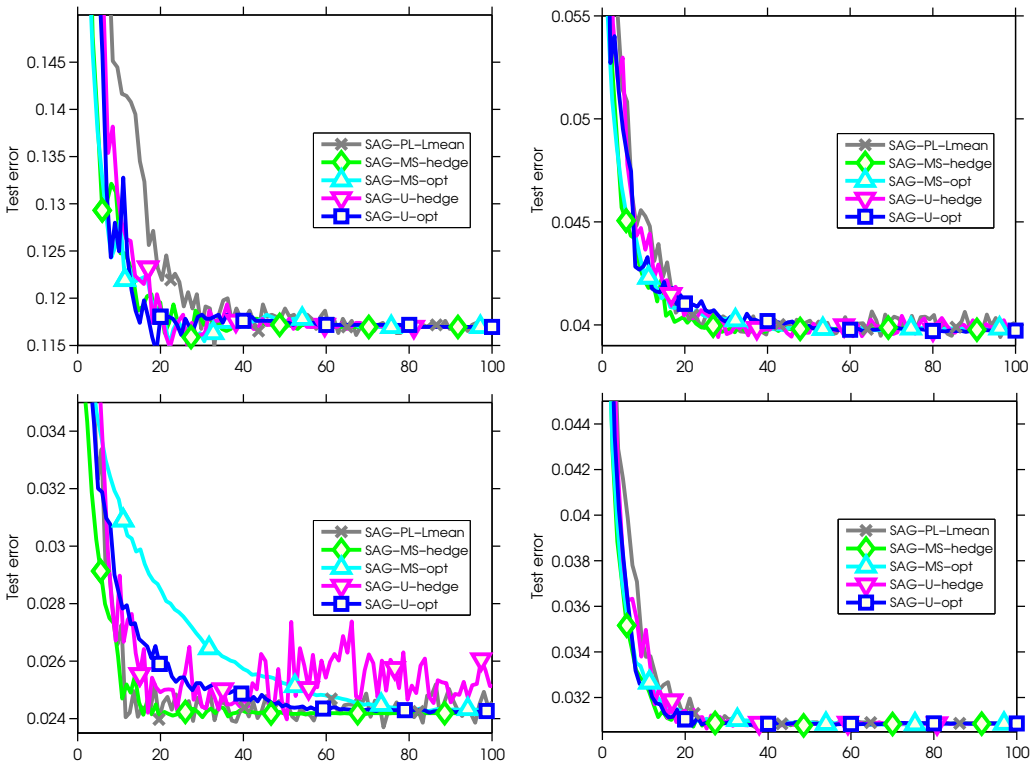


Figure 2.10: Test error against effective number of passes for the best SAG optimization configurations. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

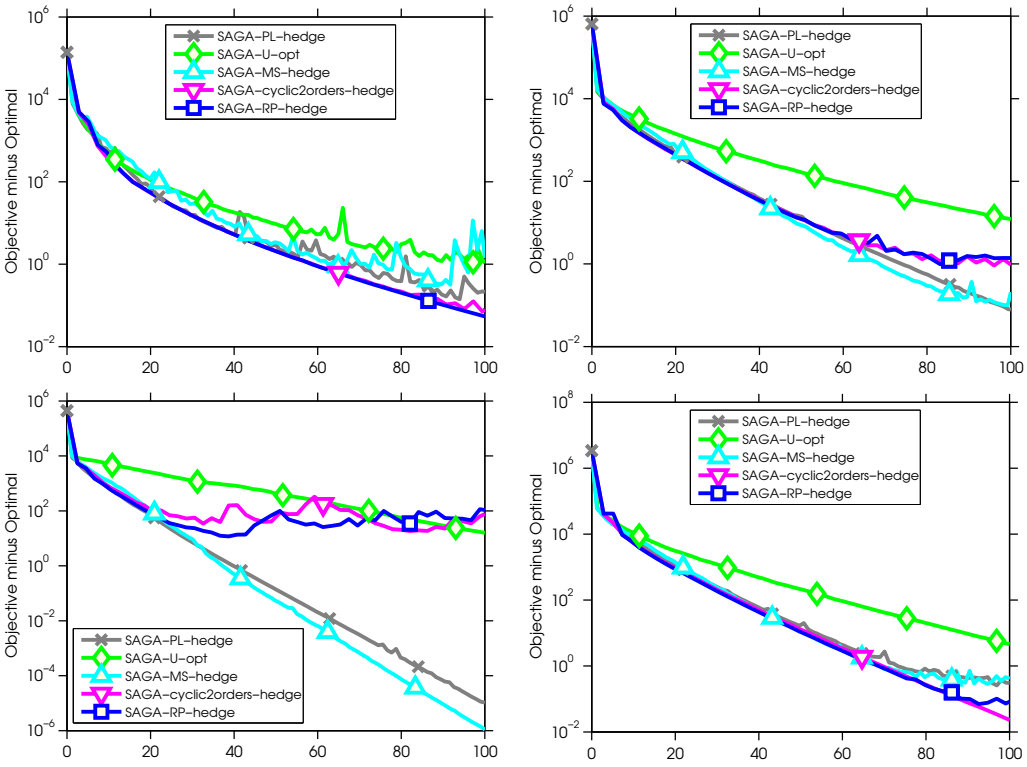


Figure 2.11: Objective minus optimal objective value against effective number of passes for different SAGA optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

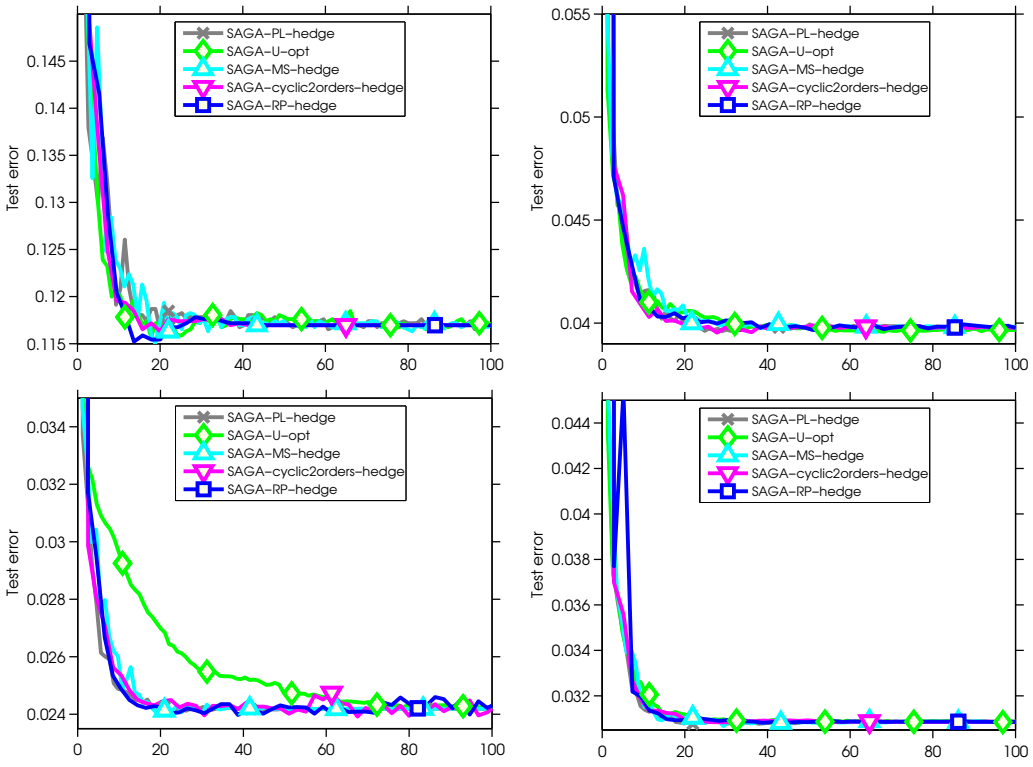


Figure 2.12: Test error against effective number of passes for the best SAGA optimization configurations. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

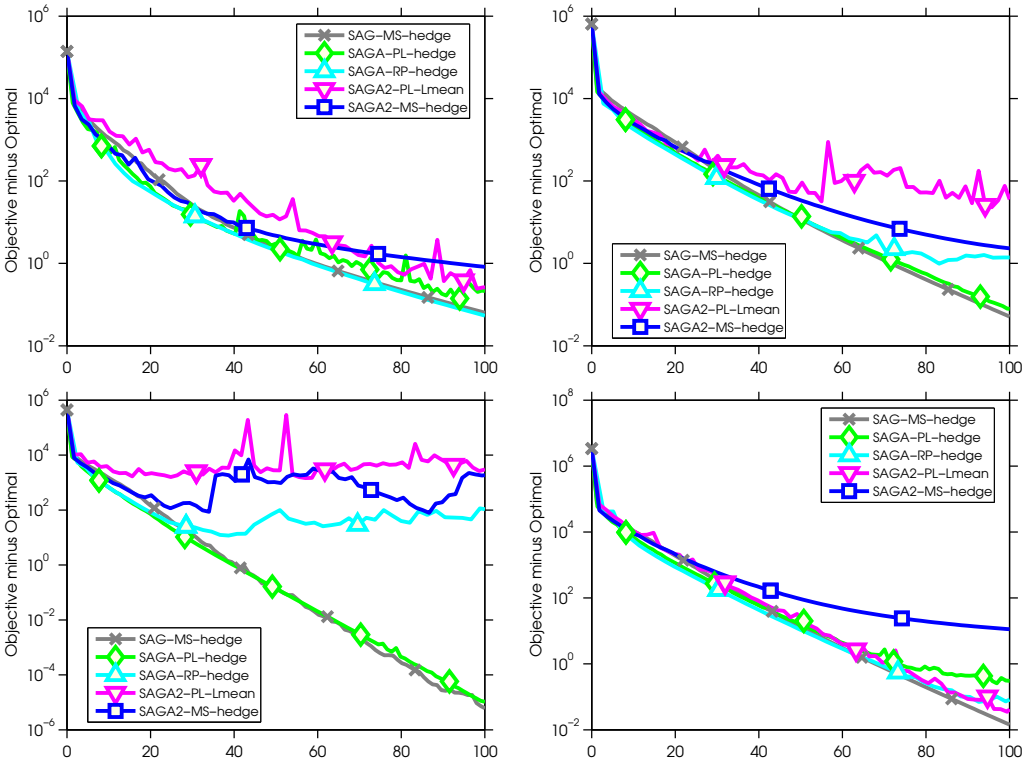


Figure 2.13: Objective minus optimal objective value against effective number of passes for the best SAG, SAGA, and SAGA2 optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

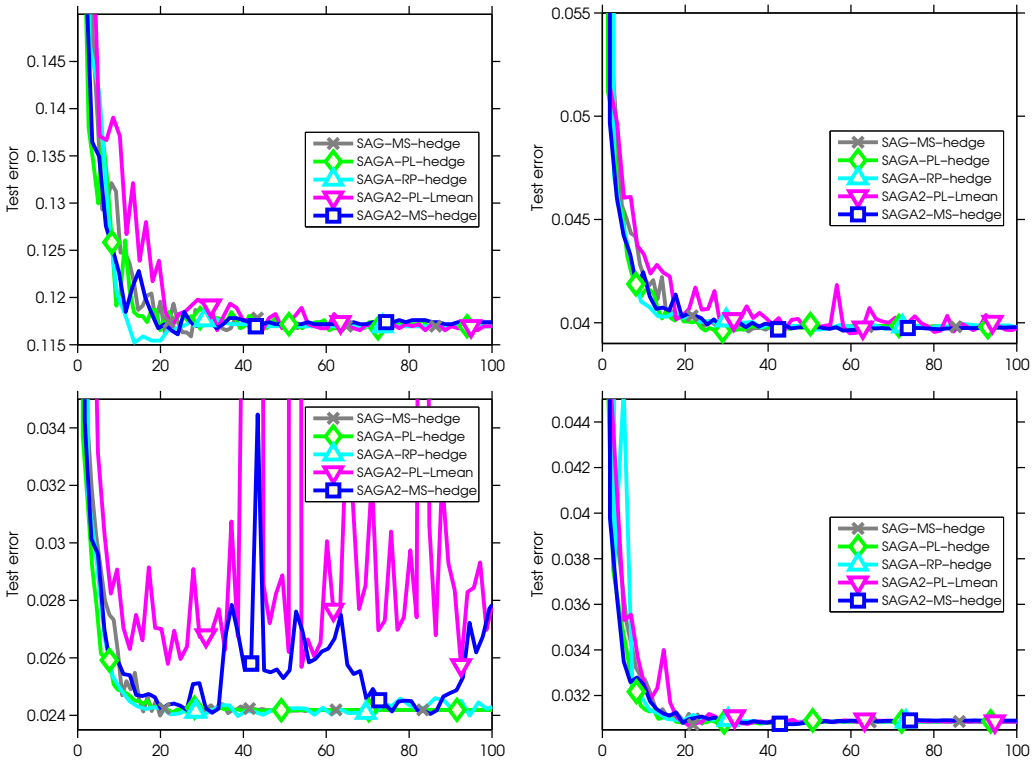


Figure 2.14: Test error against effective number of passes for the best SAG, SAGA, and SAGA2 optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

Chapter 3

Practical Stochastic Variance Reduced Gradient

This chapter still considers the same problem as Chapter 2. The problem is optimizing the average of a finite but large sum of smooth functions,

$$\min_{x \in \mathbb{R}^d} f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x). \tag{3.1}$$

A huge proportion of the model-fitting procedures in machine learning can be mapped to this problem. This includes classic models like least squares and logistic regression but also includes more advanced methods like conditional random fields and deep neural network models.

In Chapter 2 we focused on SAG, which only considers one training example on each iteration but still achieves a linear convergence rate. Other methods have subsequently been shown to have this property [Defazio et al., 2014a, Mairal, 2013, Shalev-Schwartz and Zhang, 2013], but these all require storing a previous evaluation of the gradient f_i' or the dual variables for each i. For many objectives this only requires $O(n)$ space, but for general problems this requires $O(nd)$ space making them impractical.

Recently, several methods have been proposed with similar convergence rates to SAG, but without the memory requirements such as *stochastic variance-reduced gradient* (SVRG) [Johnson and Zhang, 2013], *semi-stochastic gradient* [Konečnỳ and Richtárik, 2017], and *mixed gradient* [Mahdavi and Jin, 2013]. All of these algorithms represent small variations of the same idea. In this Chapter, we will consider SVRG. We give a canonical SVRG algorithm in the next section, but the salient features of these methods are that they evaluate two gradients on each iteration and occasionally must compute the gradient on all examples. SVRG methods often dramatically outperform classic full gradient (FG) and stochastic gradient (SG) methods, but these extra evaluations mean that SVRG is slower than SG methods in the important early iterations. They also mean that SVRG methods are typically slower than memory-based methods like SAG.

In this work we show that SVRG is robust to inexact calculation of the full gradients it requires, provided the accuracy increases over time. We use this to explore growing-batch strategies that require fewer gradient evaluations when far from the solution, and we propose a mixed SG/SVRG method that may improve performance in the early iterations. We next explore using support vectors to reduce the number of gradients required when close to the solution, and finally Section 3.4 presents the experimental results.

Algorithm 5 SVRG

Input: initial vector x^0, update frequency m, learning rate η.
for $s = 0, 1, 2, \ldots$ **do**
  $\mu^s = \frac{1}{n}\sum_{i=1}^{n} f_i'(x)$
  x_0=x^s
  for $t = 1, 2, \ldots, m$ **do**
    Randomly pick $i_t \in 1, \ldots, n$
    $x_t = x_{t-1} - \eta(f_{i_t}'(x_{t-1}) - f_{i_t}'(x^s) + \mu^s)$ $(*)$
  end for
  option I: set $x^{s+1} = x_m$
  option II: set $x^{s+1} = x_t$ for random $t \in \{1, \ldots, m\}$
end for

3.1 Notation and Stochastic Variance Reduced Gradient Algorithm

SVRG assumes f is μ-strongly convex, each f_i is convex, and each gradient f_i' is Lipschitz-continuous with constant L. The method begins with an initial estimate x^0, sets $x_0 = x^0$ and then generates a sequence of iterates x_t using

$$x_t = x_{t-1} - \eta(f_{i_t}'(x_{t-1}) - f_{i_t}'(x^s) + \mu^s), \tag{3.2}$$

where η is the positive step size, we set $\mu^s = f'(x^s)$, and i_t is chosen uniformly from $\{1, 2, \ldots, n\}$. After every m steps, we set $x^{s+1} = x_t$ for a random $t \in \{1, \ldots, m\}$, and we reset $t = 0$ with $x_0 = x^{s+1}$. SVRG is described in Algorithm 5.

Unfortunately, the SVRG algorithm requires $2m + n$ gradient evaluations for every m iterations of (3.2), since updating x_t requires two gradient evaluations and computing μ^s require n gradient evaluations. We can reduce this to $m + n$ if we store the gradients $f_i'(x^s)$, but this is not practical in most applications. Thus, SVRG requires many more gradient evaluations than the classic SG iterations of memory-based methods like SAG.

3.2 Stochastic Variance Reduced Gradient with Error

We first give a result for the SVRG method where we assume that μ^s is equal to $f'(x^s)$ up to some error e^s. This is in the spirit of the analysis of Schmidt et al. [2011], who analyze FG methods under similar assumptions. We assume that $\|x_t - x^*\| \leq Z$ for all t, which has been used in related work [Hu et al., 2009] and is reasonable because of the coercity implied by strong-convexity.

Proposition 1. *If $\mu^s = f'(x^s) + e^s$ and we set η and m so that $\rho < 1$, then the SVRG algorithm* (3.2) *with x^{s+1} chosen randomly from $\{x_1, x_2, \ldots, x_m\}$ satisfies*

$$\mathbb{E}[f(x^{s+1}) - f(x^*)] \leq \rho\mathbb{E}[f(x^s) - f(x^*)] + \frac{Z\mathbb{E}\|e^s\| + \eta\mathbb{E}\|e^s\|^2}{1 - 2\eta L}.$$

The proof is provided in Harikandeh et al. [2015]. This result implies that SVRG does not need a very accurate approximation of $f'(x^s)$ in the crucial early iterations since the first term in the bound will dominate. Further, this result implies that we can maintain the *exact* convergence rate of SVRG as long as the errors e^s decrease at an appropriate rate. For example, we obtain the same convergence rate provided that $\max\{\mathbb{E}\|e^s\|, \mathbb{E}\|e^s\|^2\} \leq \gamma\widetilde{\rho}^s$ for any $\gamma \geq 0$ and some $\widetilde{\rho} < \rho$. Further, we still obtain a linear convergence rate as long as $\|e^s\|$ converges to zero with a linear convergence rate.

3.2.1 Stochastic Variance Reduced Gradient with Batching

There are many ways we could allow an error in the calculation of μ^s to speed up the algorithm. For example:

- if evaluating each f_i' involves solving an optimization problem, then we could solve this optimization problem inexactly.
- if we are fitting a graphical model with an iterative approximate inference method, we can terminate the iterations early to save time.

When f_i is simple but n is large, a natural way to approximate μ^s is with a subset (or 'batch') of training examples $\mathcal{B}^s$ (chosen *without* replacement),

$$\mu^s = \frac{1}{|\mathcal{B}^s|}\sum_{i\in\mathcal{B}^s} f_i'(x^s).$$

The batch size $|\mathcal{B}^s|$ controls the error in the approximation, and we can drive the error to zero by increasing it to n. Existing SVRG methods correspond to the special case where $|\mathcal{B}^s| = n$ for all s. Algorithm 6 gives pseudo-code for an SVRG implementation that uses this sub-sampling strategy. If we assume that the sample variance of the norms of the gradients is bounded by S^2 for all x^s,

$$\frac{1}{n-1}\sum_{i=1}^{n}\left[\|f_i'(x^s)\|^2 - \|f'(x^s)\|^2\right] \leq S^2,$$

then we have that [Lohr, 2009, Chapter 2]

$$\mathbb{E}\|e^s\|^2 \leq \frac{n-|\mathcal{B}^s|}{n|\mathcal{B}^s|}S^2.$$

So if we want $\mathbb{E}\|e^s\|^2 \leq \gamma\widetilde{\rho}^{2s}$, where $\gamma \geq 0$ is a constant for some $\widetilde{\rho} < 1$, we need

$$|\mathcal{B}^s| \geq \frac{nS^2}{S^2 + n\gamma\widetilde{\rho}^{2s}}. \tag{3.3}$$

If the batch size satisfies the above condition then

Algorithm 6 Batching SVRG

Input: initial vector x^0, update frequency m, learning rate η.
for $s = 0, 1, 2, \ldots$ **do**
  Choose batch size $|\mathcal{B}^s|$
  $\mathcal{B}^s$ = $|\mathcal{B}^s|$ elements sampled without replacement from $\{1, 2, \ldots, n\}$.
  $\mu^s = \frac{1}{|\mathcal{B}^s|} \sum_{i \in \mathcal{B}^s} f_i'(x^s)$
  x_0=x^s
  for $t = 1, 2, \ldots, m$ **do**
    Randomly pick $i_t \in 1, \ldots, n$
    $x_t = x_{t-1} - \eta(f_{i_t}'(x_{t-1}) - f_{i_t}'(x^s) + \mu^s)$ $(*)$
  end for
  option I: set $x^{s+1} = x_m$
  option II: set $x^{s+1} = x_t$ for random $t \in \{1, \ldots, m\}$
end for

$$\begin{aligned} Z\mathbb{E}\|e^{s-1}\| + \eta\mathbb{E}\|e^{s-1}\|^2 &\leq Z\sqrt{\gamma}\widetilde{\rho}^s + \eta\gamma\widetilde{\rho}^{2s} \\ &\leq 2\max\{Z\sqrt{\gamma}, \eta\gamma\widetilde{\rho}\}\widetilde{\rho}^s, \end{aligned}$$

and *the convergence rate of SVRG is unchanged compared to using the full batch* on all iterations. The condition (3.3) guarantees a linear convergence rate under any exponentially-increasing sequence of batch sizes, the strategy suggested by Friedlander and Schmidt [2012] for classic SG methods. However, (3.3) has an inflection point at the following point $s = \log(S^2/\gamma n)/2\log(1/\widetilde{\rho})$, corresponding to $|\mathcal{B}^s| = \frac{n}{2}$. This was previously observed empirically [Aravkin et al., 2012, Figure 3], and occurs because we are sampling without replacement. This transition means we don't need to increase the batch size exponentially.

3.2.2 Mixed SG and SVRG Method

An approximate μ^s can drastically reduce the computational cost of the SVRG algorithm, but does not affect the factor 2 in the $2m + n$ gradients required for m SVRG iterations. This factor of 2 is significant in the early iterations, since this is when stochastic methods make the most progress and when we typically see the largest reduction in the *test* error.

To reduce this factor, we can consider a *mixed* strategy: if i_t is in the batch $\mathcal{B}^s$ then perform an SVRG iteration, but if i_t is not in the current batch then use a classic SG iteration. We illustrate this modification in Algorithm 7. This modification allows the algorithm to take advantage of the rapid initial progress of SG, since it predominantly uses SG iterations when far from the solution. Below we give a convergence rate for this mixed strategy.

Proposition 2. *Let $\mu^s = f'(x^s) + e^s$ and we set η and m so that $0 < \rho(L, \alpha L) < 1$ with $\alpha = |\mathcal{B}^s|/n$. If we assume $\mathbb{E}\|f_i'(x)\|^2 \leq \sigma^2$ then Algorithm 7 has*

$$\mathbb{E}[f(x^{s+1}) - f(x^*)] \leq \rho(L, \alpha L)\mathbb{E}[f(x^s) - f(x^*)] + \frac{Z\mathbb{E}\|e^s\| + \eta\mathbb{E}\|e^s\|^2 + \frac{\eta\sigma^2}{2}(1-\alpha)}{1 - 2\eta L}$$

Algorithm 7 Mixed SVRG and SG Method

Replace (*) in Algorithm 1 with the following lines:
if $f_{i_t} \in \mathcal{B}^s$ **then**
$x_t = x_{t-1} - \eta(f'_{i_t}(x_{t-1}) - f'_{i_t}(x^s) + \mu^s)$
else
$x_t = x_{t-1} - \eta f'_{i_t}(x_{t-1})$
end if

We give the proof in Harikandeh et al. [2015]. The extra term depending on the variance σ^2 is typically the bottleneck for SG methods. Classic SG methods require the step-size η to converge to zero because of this term. However, the mixed SG/SVRG method can keep the fast progress from using a constant η since the term depending on σ^2 converges to zero as α converges to one. Since $\alpha < 1$ implies that $\rho(L, \alpha L) < \rho$, this result implies that when $[f(x^s) - f(x^*)]$ is large compared to e^s and σ^2 that the mixed SG/SVRG method actually converges faster.

Sharing a single step size η between the SG and SVRG iterations in Proposition 2 is sub-optimal. For example, if x is close to x^* and $|\mathcal{B}^s| \approx n$, then the SG iteration might actually take us far away from the minimizer. Thus, we may want to use a decreasing sequence of step sizes for the SG iterations. In Harikandeh et al. [2015], we show that using $\eta = O^*(\sqrt{(n - |\mathcal{B}|)/n|\mathcal{B}|})$ for the SG iterations can improve the dependence on the error e^s and variance σ^2.

3.3 Using Support Vectors

Using a batch $\mathcal{B}^s$ decreases the number of gradient evaluations required when SVRG is far from the solution, but its benefit diminishes over time. However, for certain objectives we can further reduce the number of gradient evaluations by identifying *support vectors*. For example, consider minimizing the Huberized hinge loss (HSVM) with threshold ε [Rosset and Zhu, 2007],

$$\min_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f(b_i a_i^T x), \quad f(\tau) = \begin{cases} 0 & \text{if } \tau > 1 + \varepsilon, \\ 1 - \tau & \text{if } \tau < 1 - \varepsilon, \\ \frac{(1+\varepsilon-\tau)^2}{4\varepsilon} & \text{if } |1 - \tau| \leq \varepsilon, \end{cases}$$

In terms of (3.1), we have $f_i(x) = f(b_i a_i^T x)$. The performance of this loss function is similar to logistic regression and the hinge loss, but it has the appealing properties of both: it is *differentiable* like logistic regression meaning we can apply methods like SVRG, but it has *support vectors* like the hinge loss meaning that many examples will have $f_i(x^*) = 0$ and $f'_i(x^*) = 0$. We can also construct Huberized variants of many non-smooth losses for regression and multi-class classification.

If we knew the support vectors where $f_i(x^*) > 0$, we could solve the problem faster by ignoring the non-support vectors. For example, if there are 100000 training examples

Algorithm 8 Heuristic for skipping evaluations of f_i at x

if $sk_i = 0$ **then**
  compute $f_i'(x)$.
  if $f_i'(x) = 0$ **then**
    $ps_i = ps_i + 1$. {Update the number of consecutive times $f_i'(x)$ was zero.}
    $sk_i = 2^{\max\{0, ps_i - 2\}}$. {Skip exponential number of future evaluations if it remains zero.}
  else
    $ps_i = 0$. {This could be a support vector, do not skip it next time.}
  end if
  return $f_i'(x)$.
else
  $sk_i = sk_i - 1$. {In this case, we skip the evaluation.}
  return 0.
end if

but only 100 support vectors in the optimal solution, we could solve the problem 1000 times faster. While we typically don't know the support vectors, in this section we outline a heuristic that gives large practical improvements by trying to identify them as the algorithm runs.

Our heuristic has two components. The first component is maintaining the *list of non-support vectors at* x^s. Specifically, we maintain a list of examples i where $f_i'(x^s) = 0$. When SVRG picks an example i_t that is part of this list, we know that $f_{i_t}'(x^s) = 0$ and thus the iteration only needs one gradient evaluation. This modification is not a heuristic, in that it still applies the exact SVRG algorithm. However, at best it can only cut the runtime in half.

The heuristic part of our strategy is to skip $f_i'(x^s)$ or $f_i'(x_t)$ if our evaluation of f_i' has been zero more than two consecutive times (and skipping it an exponentially larger number of times each time it remains zero). Specifically, for each example i we maintain two variables, sk_i (for 'skip') and ps_i (for 'pass'). Whenever we need to evaluate f_i' for some x^s or x_t, we run Algorithm 8 which may skip the evaluation. This strategy can lead to huge computational savings in later iterations if there are few support vectors, since many iterations will require no gradient evaluations.

Identifying support vectors to speed up computation has long been an important part of SVM solvers, and is related to the classic shrinking heuristic [Joachims, 1999]. While it has previously been explored in the context of dual coordinate ascent methods [Usunier et al., 2010], this is the first work exploring it for linearly-convergent stochastic gradient methods.

3.4 Experimental Results

In this section, we present experimental results that evaluate our proposed variations on the SVRG method. We focus on logistic regression classification: given a set of training data $(a_1, b_1) \dots (a_n, b_n)$ where $a_i \in \mathbb{R}^d$ and $b_i \in \{-1, +1\}$, the goal is to find the $x \in \mathbb{R}^d$ solving

$$\arg\min x \in \mathbb{R}^d \ \frac{\lambda}{2}\|x\|^2 + \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-b_i a_i^T x)),$$

We consider the datasets used by Le Roux et al. [2012], whose properties are listed in Table 3.1. As in their work we add a bias variable, normalize dense features, and set the regularization parameter λ to $1/n$. We used a step-size of $\alpha = 1/L$ and we used $m = |\mathcal{B}^s|$ which gave good performance across methods and datasets. In our first experiment, we compared three variants of SVRG: the original strategy that uses all n examples to form μ^s (*Full*), a growing batch strategy that sets $|\mathcal{B}^s| = 2^s$ (*Grow*), and the mixed SG/SVRG described by Algorithm 7 under this same choice (*Mixed*). While a variety of practical batching methods have been proposed in the literature [Byrd et al., 2012, Friedlander and Schmidt, 2012, van den Doel and Ascher, 2012], we did not find that any of these strategies consistently outperformed the doubling used by the simple *Grow* strategy. Our second experiment focused on the ℓ_2-regularized HSVM on the same datasets, and we compared the original SVRG algorithm with variants that try to identify the support vectors (*SV*).

We plot the experimental results for one run of the algorithms on one dataset in Figure 3.1. In our results, the growing batch strategy (*Grow*) always had better test error performance than using the full batch, while for large datasets it also performed substantially better in terms of the training objective. In contrast, the *Mixed* strategy sometimes helped performance and sometimes hurt performance. Utilizing support vectors often improved the training objective, often by large margins, but its effect on the test objective was smaller.

In Figures [3.2-3.5], we plot the performance on the various datasets in terms of both the training objective and test error, showing the maximum/mean/minimum performance across 10 random trials. In these plots, we see a clear advantage for the *Grow* strategy on the largest datasets (bottom row), but less of an advantage or no advantage on the smaller datasets. The advantage of using support vectors seemed less dependent on the data size, as it helped in some small datasets as well as some large datasets, while in some small/large datasets it did not make a big difference.

3.5 Discussion

As SVRG is the only memory-free method among the new stochastic linearly-convergent methods, it represents the natural method to use for a huge variety of machine learning problems. In this work, we show that the performance of the SVRG algorithm can be preserved even under an inexact approximation to the full gradient. We also showed that using

Data set	Data Points	Variables	Reference
quantum	50 000	78	[Caruana et al., 2004]
protein	145 751	74	[Caruana et al., 2004]
sido	12 678	4 932	[Guyon, 2008]
rcv1	20 242	47 236	[Lewis et al., 2004]
covertype	581 012	54	[Frank and Asuncion, 2010]
news	19 996	1 355 191	[Keerthi and DeCoste, 2005]
spam	92 189	823 470	[Carbonetto, 2009, Cormack and Lynam, 2005]
rcv1Full	697 641	47 236	[Lewis et al., 2004]
alpha	500 000	500	Synthetic

Table 3.1: Binary data sets used in the experiments.

mini-batches to approximate μ^s gives a natural way to do this, explored the use of support vectors to further reduce the number of gradient evaluations, and considered several mini-batch strategies. Our experimental results indicate that many of these simple modifications should be considered in any practical implementation of SVRG. To facilitate reproducible research, the code is available at `http://www.cs.ubc.ca/~schmidtm/Software/practicalSVRG.zip`.

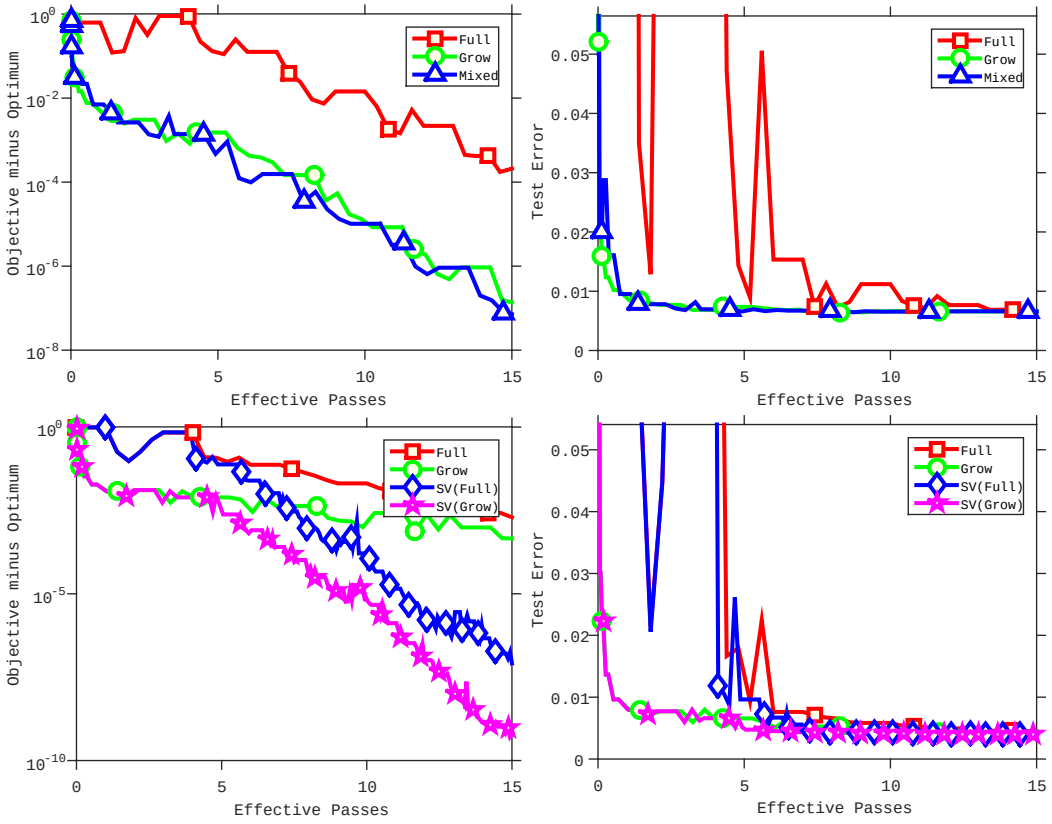


Figure 3.1: Comparison of training objective (left) and test error (right) on the *spam* dataset for the logistic regression (top) and the HSVM (bottom) losses under different batch strategies for choosing μ^s (*Full*, *Grow*, and *Mixed*) and whether we attempt to identify support vectors (*SV*).

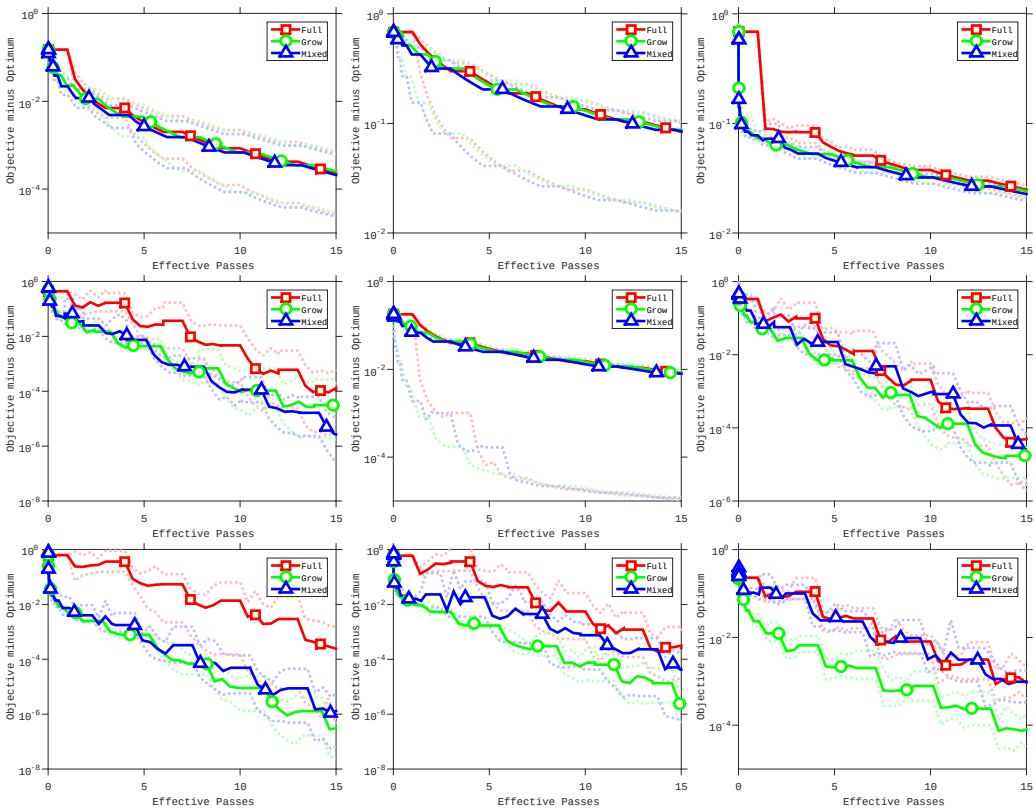


Figure 3.2: Comparison of training objective of logistic regression for different datasets. The top row gives results on the *quantum* (left), *protein* (center) and *sido* (right) datasets. The middle row gives results on the *rcv11* (left), *covertype* (center) and *news* (right) datasets. The bottom row gives results on the *spam* (left), *rcv1Full* (center), and *alpha* (right) datasets.

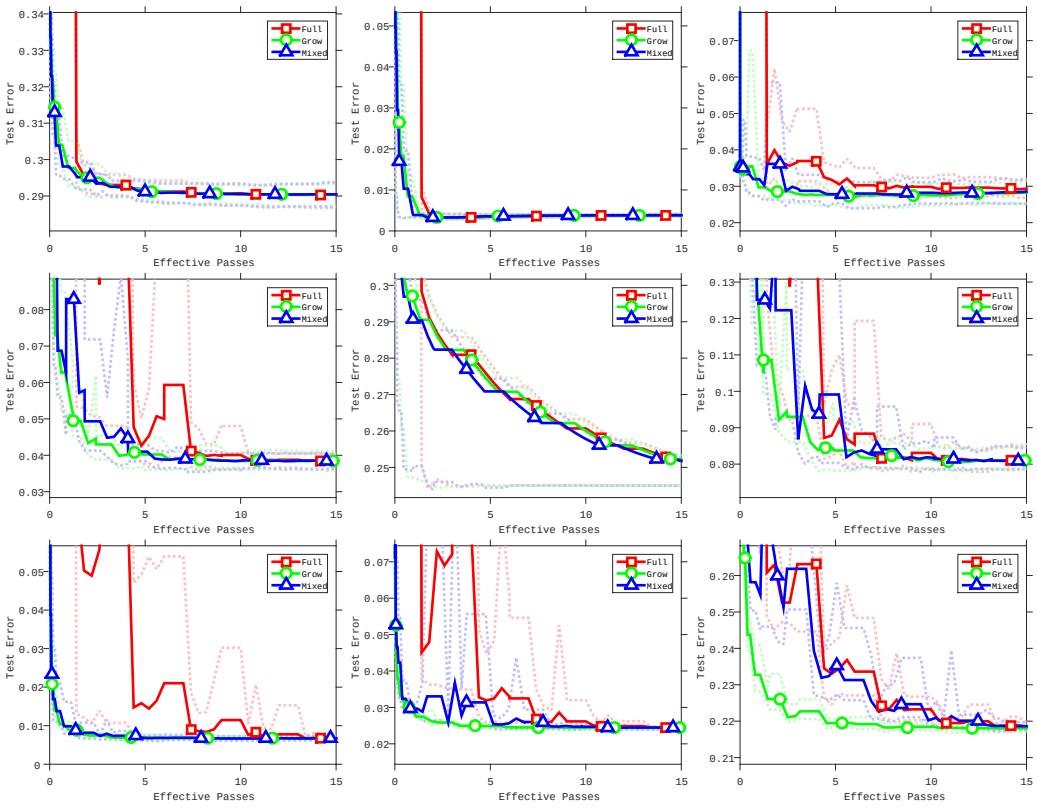


Figure 3.3: Comparison of test error of logistic regression for different datasets. The top row gives results on the *quantum* (left), *protein* (center) and *sido* (right) datasets. The middle row gives results on the *rcv11* (left), *covertype* (center) and *news* (right) datasets. The bottom row gives results on the *spam* (left), *rcv1Full* (center), and *alpha* (right) datasets.

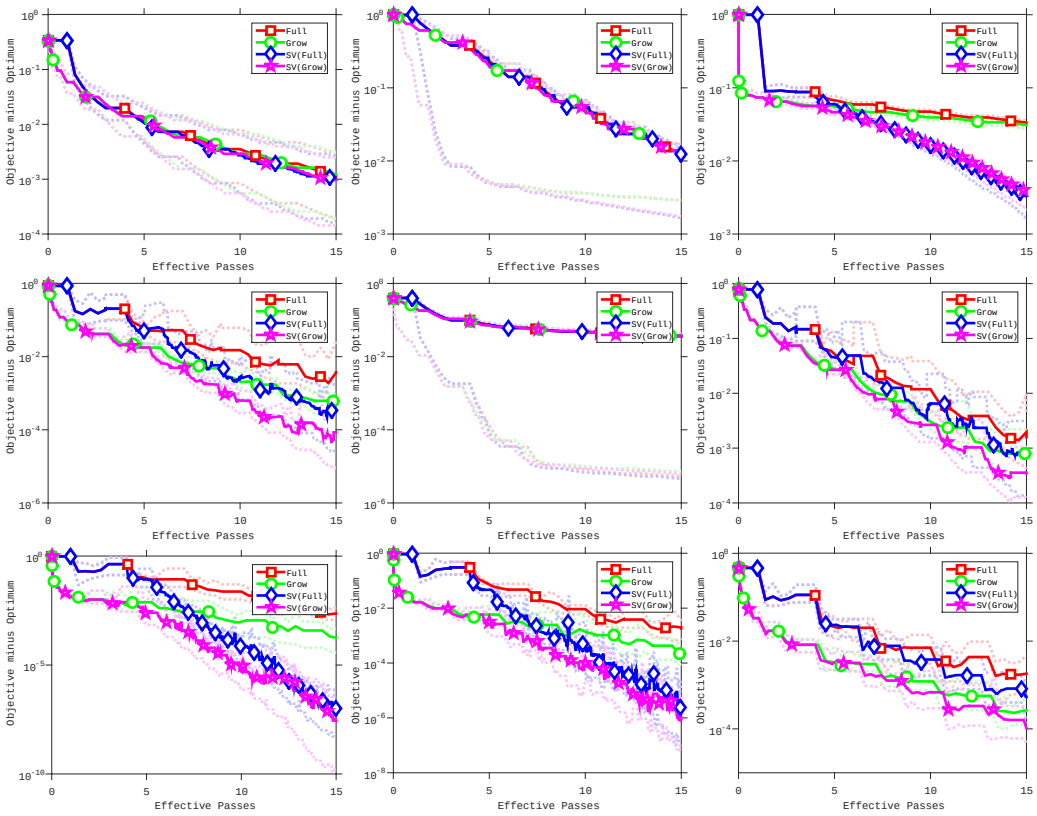


Figure 3.4: Comparison of training objective of SVM for different datasets. The top row gives results on the *quantum* (left), *protein* (center) and *sido* (right) datasets. The middle row gives results on the *rcv11* (left), *covertype* (center) and *news* (right) datasets. The bottom row gives results on the *spam* (left), *rcv1Full* (center), and *alpha* (right) datasets.

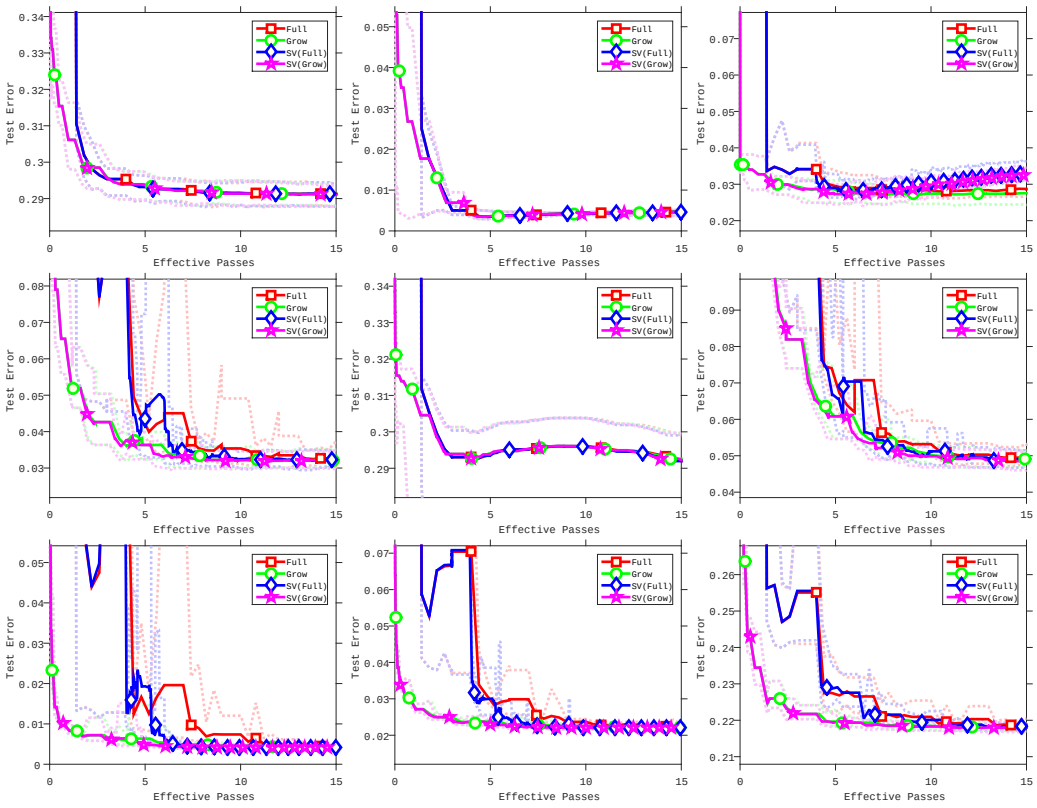


Figure 3.5: Comparison of test error of SVM for different datasets. The top row gives results on the *quantum* (left), *protein* (center) and *sido* (right) datasets. The middle row gives results on the *rcv11* (left), *covertype* (center) and *news* (right) datasets. The bottom row gives results on the *spam* (left), *rcv1Full* (center), and *alpha* (right) datasets.

Chapter 4

Harmless and First-Order Bayesian Optimization

Chapters 2 and 3 focused on training machine learning models which is usually a high-dimensional convex problem. In Chapters 4 and 5, we consider hyper-parameter tuning which usually leads to a low-dimensional non-convex problem. There are several approaches to solve this problem such as grid-search and random search. However, these methods works only in very low dimensions. One of the most popular methods in hyperparameter tuning is Bayesian optimization (BO).

BO has a long history and has been used in variety of fields (see Shahriari et al. [2016] for a recent review), with recent interest from the machine learning community in the context of automatic hyperparameter tuning [Snoek et al., 2012]. However, the empirical results of a recent work have called into question the usefulness of Bayesian optimization. Li et al. [2016] show on a large number of problems that Bayesian optimization often only gave a small gain over random search and in fact was typically outperformed by running random search for twice as many iterations. On the other hand, for some specific problems BO does in fact handily beat random (see our experiments) and we know that under certain smoothness assumptions BO can be exponentially faster than random [Bull, 2011, Theorem 5].

In the context of these conflicting results, in this chapter we propose two strategies for improving the performance of Bayesian optimization methods. First, in the next section we introduce the concept of a "harmless" Bayesian optimization algorithm. This is not specific to Bayesian optimization. In fact we introduce a more general concept known as "harmless global optimization". This is an algorithm that does no worse than random in the worst case, but that might still take advantage of certain structure of the function, for example, a high degree of smoothness in case of BO. Second, since smooth functions are one class of function where BO has a large advantage over random, we propose to re-explore the idea of "first-order" Bayesian optimization (FOBO) methods that use gradient information to improve performance. We show how gradient information can lead to improved performance, and also propose FOBO strategies that use directional derivatives. The advantage of using directional derivatives is that they reduce the cost and memory requirements of FOBO methods.

Directional derivatives can be obtained by:

1. automatic differentiation for any analytic function. In this case, directional derivative reduces the memory cost as we only need to do the forward mode so we do not need the memory requirements of reverse mode automatic differentiation.

2. numerical differentiation for "black box" situations where we do not have gradient code available. In this case directional derivative is cheaper than evaluating the full gradient as it requires only 2 function evaluations.

4.1 Bayesian Optimization

BO methods are typically based on Gaussian processes (GPs), since they have appealing universal consistency properties and admit a closed-form posterior distribution [Rasmussen and Williams, 2006]. In particular, BO methods assume a smooth GP prior on the unknown function, and use the observed function evaluations to compute a posterior distribution over the possible function values at any point x. At iteration t, given the previously selected points $\{x_1, x_2, \dots x_{t-1}\}$ and their corresponding observations $\mathbf{y}_t = [y_1, y_2, \dots, y_{t-1}]$, the algorithm uses an *acquisition function* (based on the GP posterior) to select the next point to evaluate. The value of the acquisition function at a point characterizes the importance of evaluating that point in order to maximize f. To determine x_t, we maximize this acquisition function over all x using an auxiliary optimization procedure (typically we can only approximately solve this maximization). Algorithm 9 outlines the generic Bayesian optimization framework.

We formalize the high-level procedure. We assume that $f \sim GP(0, k(x, x'))$. Here $k(x, x')$ is a kernel function which quantifies the similarity between points x and x'. We denote the maximum value of the function until iteration t as y_t^* and the set $\{1, 2, \dots, t\}$ as $[t]$. Let $\mathbf{k}_t(x) = [k(x, x_1), k(x, x_2), \dots, k(x, x_t)]$ and let us denote the $t \times t$ kernel matrix as K (so $K_{i,j} = k(x_i, x_j)$ for all $i, j \in [t]$). Given the function evaluations (observations), the posterior distribution at point x after t iterations is given as $\mathcal{P}[f_t(x)] \sim N(\mu_t(x), \sigma_t(x))$. Here, the mean and standard deviation of the function at x are given as:

$$\begin{aligned} \mu_t(x) &= \mathbf{k}_t(x)^T \left(K + \sigma^2 I_t\right)^{-1} \mathbf{y}_t, \\ \sigma_t(x) &= k(x, x) - \mathbf{k}_t(x)^T \left(K + \sigma^2 I_t\right)^{-1} \mathbf{k}_t(x). \end{aligned} \tag{4.1}$$

As alluded to earlier, an acquisition function uses the above posterior distribution in order to select the next point to evaluate the function at. A number of acquisition functions have been proposed in the literature, with the most popular ones being based on upper-confidence bounds (UCB) [Srinivas et al., 2010] or posterior sampling [Thompson, 1933], ones based on importance function values [Kushner, 1964, Močkus, 1975] or those based on entropy search [Hennig and Schuler, 2012, Hernández-Lobato et al., 2014, Villemonteix et al., 2009]. In this work, we focus on four simple widely-used acquisition functions: upper confidence bound (UCB) [Srinivas et al., 2010], Thompson sampling (TS) [Thompson, 1933], expected improvement (EI) [Močkus, 1975] and probability of improvement (PI) [Kushner, 1964]. However, we expect that our conclusions would apply to other acquisition functions. For brevity, when defining the acquisition functions we drop the $t-1$ subscripts from $\mu_{t-1}(x)$, $\sigma_{t-1}(x)$, and y_{t-1}^* .

UCB: The acquisition function $UCB(x)$ is defined as:

$$UCB(x) = \mu(x) + \beta_t^{1/2}\sigma(x) \tag{4.2}$$

Here, β_t is positive parameter that trades off exploration and exploitation.
TS: For TS, in each iteration we first sample a function $\widetilde{f}_t(x)$ from the GP posterior, $\widetilde{f}_t \sim GP(\mu_t(x), \sigma_t(x))$. TS then selects the point x_t which maximizes this deterministic function $\widetilde{f}_t$.
Probability of Improvement: We define the possible improvement (over the current maximum) at x as $I(x) = \max\{f(x) - y^*, 0\}$ and the indicator of improvement u(x) as

$$u(x)) = \begin{cases} 0, & \text{if } f(x) < y^* \\ 1, & \text{if } f(x) \geq y^* \end{cases}.$$

PI selects the point x which maximizes the probability of improving over y^*. If $\phi(\cdot)$ and $\Phi(\cdot)$ are the probability density function and the cumulative distribution function for the standard normal distribution, then the PI acquisition function is given as [Kushner, 1964]:

$$PI(x) = \int_{-\infty}^{\infty} u(x)\phi(f(x))df = \int_{y^*}^{\infty} \phi(f(x))df = \Phi\left(z(x, y^*)\right) \tag{4.3}$$

where we've defined the transformation $z(u, v) = \frac{\mu(u)-v}{\sigma(u)}$.
Expected Improvement: EI selects an x that maximizes $\mathbb{E}[I(x)]$, where the expectation is over the distribution $\mathcal{P}(f_t(x))$. If $\phi(\cdot)$ is the pdf of the standard normal distribution, the expected improvement acquisition function can be written as [Močkus, 1975]:

$$\begin{aligned} EI(x) &= \int_{-\infty}^{\infty} I(x)\phi(f(x))df \\ &= \int_{y^*}^{\infty} (f(x) - y^*)\phi(f(x))df \\ &= \sigma(x) \cdot [z(x, y^*) \cdot \Phi(z(x, y^*)) + \phi(z(x, y^*))] \end{aligned} \tag{4.4}$$

4.2 Harmless Bayesian Optimization

We consider the problem of maximizing a real-valued function f over a domain $\mathcal{X}$ consisting of finite lower and upper bounds on every variable,

$$\underset{x \in \mathcal{X}}{\text{argmax}}\, f(x). \tag{4.5}$$

We first consider a zero-order oracle where on iteration t the algorithm can learn about the function by choosing a parameter value x^t and receiving the corresponding function value $f(x^t)$. In this model, we consider the goal of trying to minimize the number of iterations t before we can guarantee that we find a parameter value $\widehat{x}$ such that $f(\widehat{x}) - f^* \leq \varepsilon$ (where

Algorithm 9 Bayesian optimization

Input: m data points $D_0 = (x_0, y_0), (x_1, y_1), ..., (x_m, y_m)$.
for iteration $t = 0, 1, 2, \ldots, T$ **do**
  select new x_{t+1} by optimizing acquisition function a

$$x_{t+1} = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, a(x; D_t) \tag{4.6}$$

  query objective function to obtain y_{n+1}.
  update model by $D_{n+1} = (x_{n+1}, y_{n+1})$
end for
return location of the maximum

f^* is the maximum of f and ε is some positive constant). In this chapter we will primarily be interested in BO methods based on Gaussian processes and will consider deterministic functions, but we expect that these observations also hold in other settings.

Firstly, we note that finding such an $\widehat{x}$ is *impossible in any finite number of oracle calls* unless we make assumptions about f. Because the real numbers are uncountable, for any deterministic algorithm we can construct a function that achieves an arbitrarily low function value at a parameter setting that the algorithm will never try (and similar arguments hold for stochastic algorithms). Thus, to say anything in this setting we need some sort of continuity assumptions about the function f.

One of the most natural assumption about f that we might make is that f is Lipschitz-continuous, meaning that the amount that f can change as we change x is bounded by a constant times the change in x. Under this assumption, it is known that *any* algorithm requires $\Omega(1/\varepsilon^d)$ iterations to reach an accuracy of ε [Nesterov, 2004, Theorem 1.1.2]. Further, an $O(1/\varepsilon^d)$ worst-case rate can be achieved by using a grid-based search [Nesterov, 2004, Corollary 1.1.1] or in expectation simply by trying random values for the x^t (because the probability that a random x is an ε-optimal solution under this asumption is $\Omega(\varepsilon^d)$). Thus, in some sense random search is an *optimal* algorithm. Further, this $O(1/\varepsilon^d)$ rate is faster than the best known rates for BO in many settings, in the sense that the exponent of ε for BO can be larger than d [Bull, 2011, Theorems 1-4].[10]

However, for functions with a sufficient amount of smoothness BO can beat random search. In particular, under additional assumptions and using ν as a measure of smoothness of f, Bull shows in a fascinating result that a variant of BO only requires $\widetilde{O}(1/\varepsilon^{d/\nu})$ iterations [Bull, 2011, Theorem 5]. Thus, for "nice" functions where ν is greater than 1 BO can be exponentially faster than random search. This gives support for the empirical observation that in some settings BO soundly beats random search.

Unfortunately, in the black-box setting we typically do not know ν and so we don't

[10] We state results in terms in terms of the number of needed iterations, but we could equivalently state results in terms of the error after a fixed number of iterations. For example, if we require $O(1/\varepsilon^d)$ iterations to reach an accuracy of ε then we can guarantee a sub-optimality of $O(1/t^{1/d})$ after t iterations.

know if it is greater or less than 1. This motivates us to consider "harmless" global optimization:

- A "harmless" global optimization algorithm is a method that requires at most $O(1/\varepsilon^d)$ iterations to achieve an accuracy of ε on a Lipschitz-continuous function, and thus up to a constant factor performs as well as random search in the worst case.

A harmless BO method would be a variant of BO that satisfies the harmless property above. It is quite simple to achieve a "harmless" BO method by combining an existing BO method with an existing "harmless" method like random search. We simply alternate between performing the iterations of the two algorithms. For example, if on odd iterations we evaluate a random x^t and on even iterations we apply a BO method, this would constitute a "harmless" BO method. This is actually quite similar to the "ε-greedy" approach that Bull requires just for convergence of BO (and which is a common way to address exploration-exploitation issues). Further, if our BO method is the method analyzed by Bull then this simple harmless BO method achieves the best known rate of $\widetilde{O}(1/\varepsilon^{\min\{d,d/\nu\}})$. This assumes that the BO method ignores the random search iterations, but in practice we would likely do better by giving the BO method access to these iterates. Another variation is, instead of random points, to choose the furthest point from all the previous x^t values. This is harmless and guarantees that we satisfy a measure of progress in terms of exploring the space (this is a known problem for BO).

Although the idea of a "harmless" GO method is quite simple (both to state and to achieve), it can be a useful guiding principle in designing GO methods. For example, in Chapter 5 we use it to design an adaptive approach to estimate the Lipschitz constant L within Lipschitz optimization that is provably "harmless". Previous methods use heuristics for estimating L, and these heuristics are not harmless if they underestimate it.

4.3 First-Order Bayesian Optimization

Making a BO method harmless protects it from obtaining poor performance on functions with a low degree of smoothness, but often we are faced with maximizing a function with a high degree of smoothness. In this differentiable case we should expect to improve performance by incorporating knowledge of the derivatives at the guesses x^t. We call methods based on this extra information first-order Bayesian optimization (FOBO) methods.

Incorporating derivative observations in Gaussian processes is not a new idea [Morris et al., 1993, Osborne, 2010, Rasmussen and Williams, 2006, Solak et al., 2003], and there has been some work on using derivative information specifically for the purpose of Bayesian optimization on low-dimensional problems [Lizotte, 2008, Section 5.2.4]. In this section we revisit this idea in higher dimensions and explore how directional derivatives can reduce the time/memory as well as the need to write gradient code when all that we have available is a zero-order black box.

The usual BO assumption is that the values of f are jointly Gaussian, being generated by a Gaussian process. We can incorporate derivative information by also assuming that

derivatives are being generated by a Gaussian process. In particular, we will assume that the function values and all first derivatives of f are jointly Gaussian and that the covariance kernel is twice differentiable. In this setting, the additional elements of the covariance kernel involving partial derivatives are given by [Adler, 2010, Papoulis and Pillai, 2002]

$$\mathrm{cov}(f(x^i), \partial_p f(x^j)) = \partial_p k(x^i, x^j), \text{ and} \tag{4.7}$$

$$\mathrm{cov}(\partial_p f(x^i), \partial_q f(x^j)) = \partial_p \partial_q k(x^i, x^j), \tag{4.8}$$

where $\partial_p f$ denotes the partial derivative of f with respect to direction p. Therefore, as long as the kernel is twice differentiable and the gradient can be evaluated, the GP can be extended to include gradient observations.

While the cost of computing gradients for analytic functions cannot be more than a constant factor more expensive than the cost of computing the function value, the memory and time requirement of the GP model increase if we use the full gradient of each x^t. In particular, the memory is increased from $O(t^2)$ to $O(t^2d^2)$ when we have d variables. Similarly, the cost of the GP is increased from $O(t^3)$ to $O(t^3d^3)$ (assuming we use an exact solver, for illustration purposes). If this extra memory requirement is too large, then we propose to instead of modelling the gradient as a GP to instead model a *directional derivative* $\partial_p f(x^t)$ for a particular direction p. The most logical direction p to consider is the gradient direction, $\nabla f(x^t)$ (So the directional derivative is the magnitude of the gradient). However, this direction is expensive to evaluate as it requires the full gradient information. Another alternative is use a random direction. This directional derivative still provides information about how the function changes, but since it is a scalar it only increases the time/memory by a constant factor.

In many scenarios where BO is applied it is possible to compute gradients. For example, Bengio as well as Maclaurin et al. have shown how to compute gradients with respect to hyper-parameters of machine learning models [Bengio, 2000, Maclaurin et al., 2015]. It is likely that these existing methods could be improved through the use of FOBO. On the other hand, sometimes we really have a "black box" and cannot get access to the gradient. In these settings, we can obtain directional derivatives $\partial_p f(x^t)$ for a given direction p for a cost of 2 function evaluations using numerical differentiation. A particularly nice way of doing this is the case of analytic functions with the complex-step trick [Martins et al., 2003] when using automatic differentiation. Our experiments indicate that simply setting p to a random direction can be more effective than standard BO methods for smooth functions.

4.4 Experiments

We performed a set of numerical experiments on a number of test functions to illustrate the two ideas explored in this chapter.

Datasets: We perform an extensive experimental evaluation and present results on twelve synthetic datasets. For the synthetic experiments, we use the standard global-optimization benchmarks. Table 4.1 summarizes the names and the properties of the used test functions.

Function name	Dimensions	x^*	Bounds
Branin	2D	[9.42478, 2.475]	[[-5, 10], [0, 15]]
Camel	2D	[0.0898, -0.7126]	[[-3, 3], [-2, 2]]
Goldstein-Price	2D	[0.0, -1.0]	[[-2.0, 2.0]]
Michalewicz	2D	[2.2, 1.57]	[[0, π]]
Michalewicz	5D		[[0, π]]
Michalewicz	10D		[[0, π]]
Rosenbrock	2D	[1, 1]	[[-2.048, 2.048]]
Rosenbrock	3D	[1, 1, 1]	[[-2.048, 2.048]]
Rosenbrock	4D	[1, 1, 1, 1]	[[-2.048, 2.048]]
Rosenbrock	5D	[1, 1, 1, 1, 1]	[[-2.048, 2.048]]
Hartmann	3D	[0.1146, 0.5556, 0.8525]	[[0.0, 1.0]]
Hartmann	6D	[0.2017, 0.15, 0.4769, 0.2753, 0.3117, 0.6573]	[[0.0, 1.0]]

Table 4.1: Test functions used.

The closed form formula and domain for each of these functions is given in Jamil and Yang [2013].

Experimental Setup: For Bayesian optimization, we use a Gaussian Process prior with the squared exponential kernel. We modified the publically available BO package *pybo* [Hoffman and Shahriari, 2014] to implement HBO and FOBO. All the prior hyper-parameters were set and updated across iterations according to the open-source Spearmint package[11]. In order to make the optimization invariant to the scale of the function values, similar to Spearmint, we standardize the function values; after each iteration, we center the observed function values by subtracting their mean and dividing by their standard deviation. We then fit a GP to these rescaled function values. We found that standardization significantly improves the performance over the original pybo code. We use DIRECT [Jones et al., 1993] in order to optimize the acquisition function in each iteration. We spent a long time optimizing the performance of our baseline BO methods and we verified that our version of BO performs better than or equal to Spearmint across benchmark problems. All our results are averaged over 10 independent runs, and each of our figures plots the mean and standard deviation of the absolute error (compared to the global optimum) versus the number of function evaluations. For functions evaluated on log scale, we show the mean and the 10^{th} and 90^{th} quantiles.

4.4.1 Harmless Bayesian Optimization Experiment

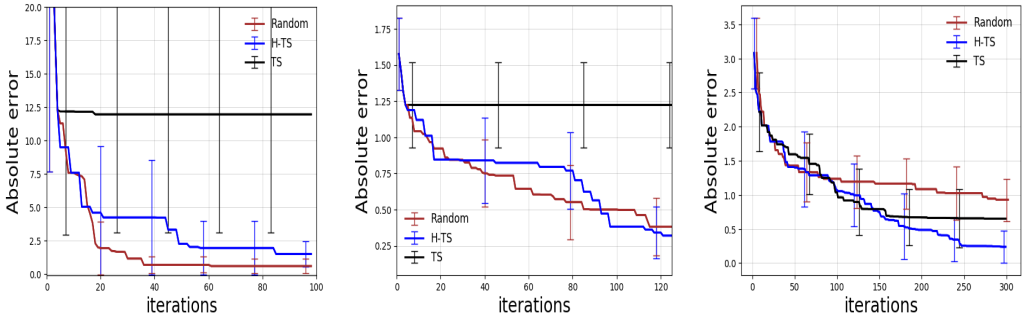
For this experiment, we compare the performance of the BO and HBO methods for the EI, PI, UCB and TS acquisition functions. For UCB, we set the trade-off parameter β according to Kandasamy et al. [2017]. In addition to these, we use random exploration as

[11] Available at https://github.com/JasperSnoek/spearmint

a baseline. HBO was implemented by switching from BO to evaluating a random x^t every fourth iteration. To make it easier to read, a sample of the results for the HBO experiments are presented in Figure 4.1, 4.1, 4.2, 4.3, and 4.4. In these figures 'H-*' stands for the harmless BO version with a certain acquisition function, for example, 'H-TS' represents Harmless BO with Thompson sampling acquisition function. In these figures, we present the results for Branin, Michalwicz 2D, and Hartmann 6D. All the experiments are presented in Appendix A.1.

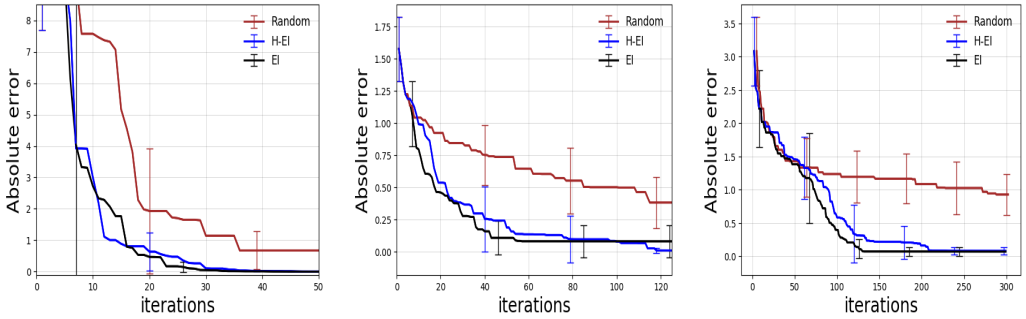
The results of the first experiment can be summarized as follows:

- **TS:** HBO does exactly what it is designed for, especially for Branin, Michalwicz 2D, and Michalwicz 5D. For these functions, Random search significantly outperforms BO. However, HBO improves the performance of BO to match Random search as shown in Figure 4.1. For the other functions, HBO performs no worse than random and close to the performance of BO.



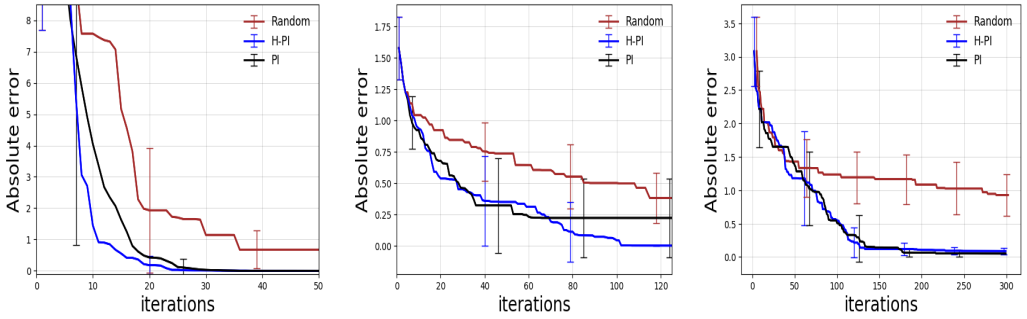
(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.1: Comparing conventional BO and HBO and random exploration for TS on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions.



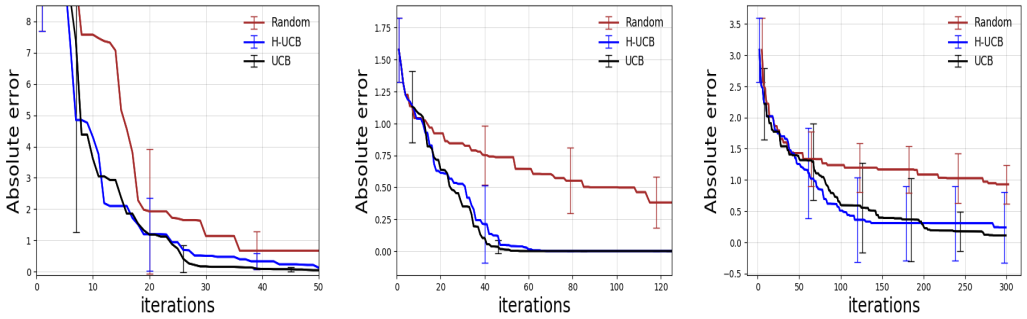
(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.2: Comparing conventional BO and HBO and random exploration for EI on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions.



(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.3: Comparing conventional BO and HBO and random exploration for PI on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions.



(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.4: Comparing conventional BO and HBO and random exploration for UCB on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions.

- **EI:** HBO is as good as BO as shown in Figure 4.2.
- **PI:** although BO is always better than random, HBO improves the performance for Branin as shown in Figure 4.3. However, for Goldstein-Price HBO is slightly worse than BO.
- **UCB:** HBO is as good as BO as shown in Figure 4.4.

4.4.2 First-Order Bayesian Optimization Experiment

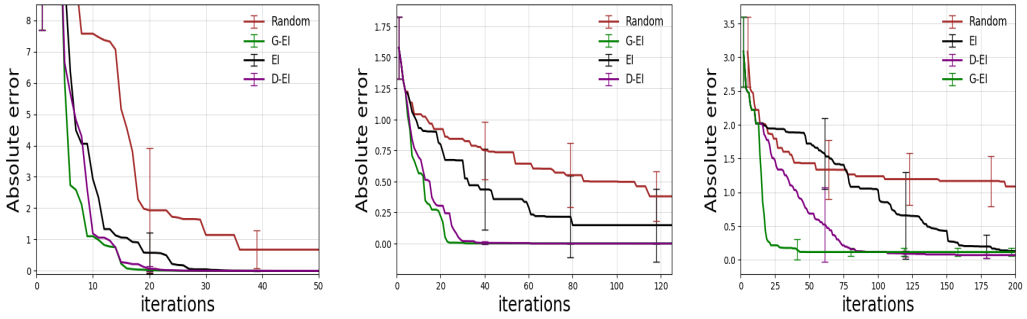
For the FOBO experiment, we compare BO and FOBO methods for the EI, PI, and UCB acquisition functions. The directional derivative was implemented by choosing a random direction at each iteration. This choice provided a good performance and there was no need to optimize the direction. We present some of the results in Figures 4.5, 4.6, and 4.7. In these figures 'G-*' and 'D-*' stands for BO with gradient and BO with directional derivatives,

respectively, with a certain acquisition function. These figures show that using gradient information provides a huge gain in the performance as expected. However, using directional derivatives provide a good trade-off between the computation and the performance. Furthermore, for some functions, using directional derivatives is almost as good as using the gradient information. It is important to note that using directional derivatives is only doubling the size of the covariance matrix instead of multiplying it by the dimensionality of the problem as in the case of using the full gradient information. Figures 4.8, 4.9, and 4.10 show the comparison in terms of the number of function evaluations. If finite difference is used, at each iteration BO, G-BO, and D-GBO requires 1, $d+1$, and 2 function evaluations, respectively. It can be noticed in these figures that the gain from using directional derivatives can compensate for its cost, especially in higher dimensions. This is not always the case when using gradients. In these figures, we present the results for Branin, Michalwicz 2D, and Hartmann 6D. All the experiments are presented in Appendix A.1.

4.5 Discussion

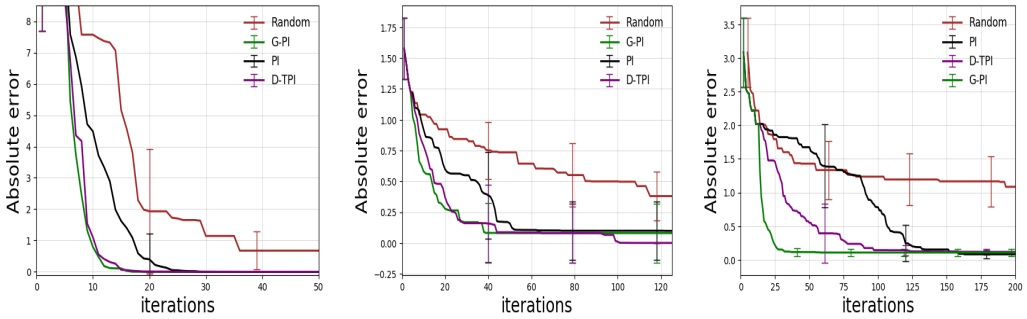
In this Chapter, we have proposed 2 ideas to improve the performance of Bayesian optimization. Our experiments show that using HBO makes BO "harmless", especially in the case of using the TS acquisition function. Moreover, we conjecture that better harmless methods are possible which *locally* try to exploit smoothness in certain regions by adapting to the properties of the black-box function. Similarly, it might be possible to improve the method by biasing the random samples away from the points sampled by the GP. As for FOBO, our experiments show that using gradient information provides huge performance gains. However, directional derivatives are cheaper to use so may be a more practical alternative in many settings. We conjecture that including gradient observations can further reduce the exponent in the iteration complexity of BO methods. After the publication of our work on this topic, several works have built on top of the proposed ideas in this Chapter, for example Frazier [2018], Wu et al. [2017a], Wu [2017], Wu and Frazier [2017], Wu et al. [2017b] for the gradient work and Dodge et al. [2017], Falkner et al. [2017] for the harmless BO work.

This chapter considered test problems without noise. We expect HBO to be more helpfull in the noisy case. As for FOBO, noisy gradients can be more harmful than noisy observations. For this reason, the noise parameter should be different between the function observations and the gradient observations.



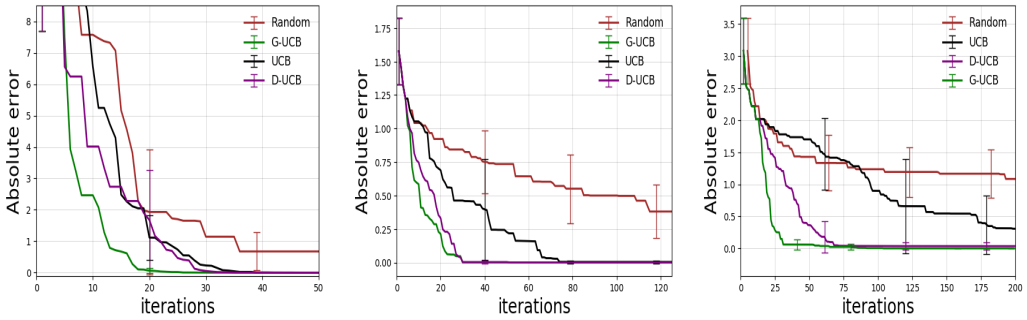
(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.5: Comparing conventional BO and FOBO and random exploration for EI on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions.



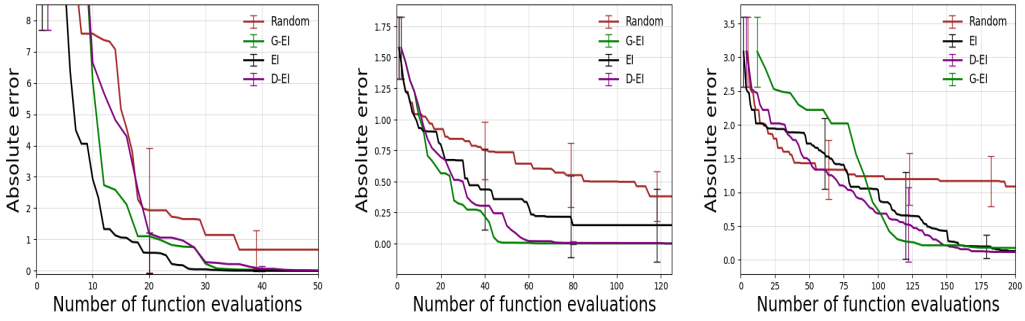
(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.6: Comparing conventional BO and FOBO and random exploration for PI on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions.



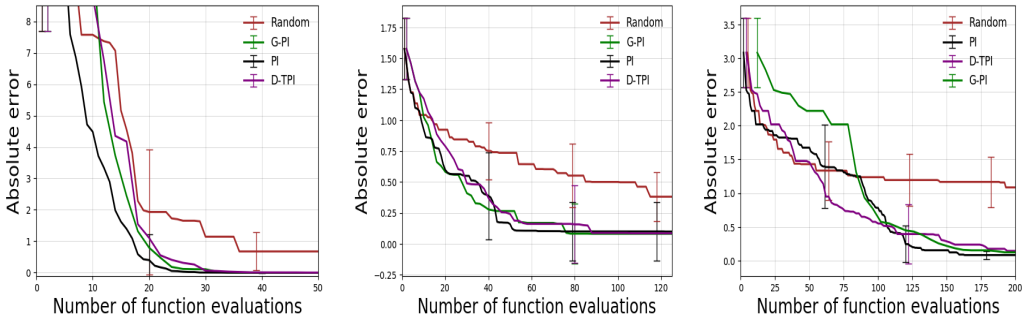
(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.7: Comparing conventional BO and FOBO and random exploration for UCB on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions.



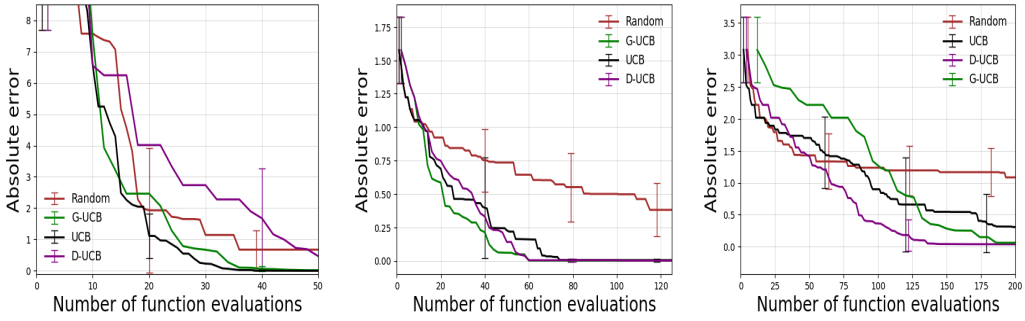
(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.8: Comparing conventional BO and FOBO and random exploration for EI on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions in terms of number function evaluations.



(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.9: Comparing conventional BO and FOBO and random exploration for PI on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions in terms of number function evaluations.



(a) Branin 2D (b) Michalwicz 2D (c) Hartmann 6D

Figure 4.10: Comparing conventional BO and FOBO and random exploration for UCB on Branin (a), Michalwicz 2D (b), and Hartmann 6D (c) test functions in terms of number function evaluations.

Chapter 5

Lipschitz Bayesian Optimization

In this chapter, we continue the work on BO. As discussed in Chapter 4, BO is an example of a global black-box optimization algorithm [Eligius and Boglárka, 2010, Jones et al., 1998, Pintér, 1991, Rios and Sahinidis, 2013] which optimizes an unknown function that may not have nice properties such as convexity. In the typical setting, we assume that we only have access to a black box that evaluates the function and that it is expensive to do these evaluations. The objective is to find a global optima of the unknown function with the minimum number of function evaluations.

The global optimization of a real-valued function is impossible unless we make assumptions about the structure of the unknown function. Lipschitz continuity (that the function can't change arbitrarily fast as we change the inputs) is one of the weakest assumptions under which optimizing an unknown function is still possible. Lipschitz optimization [Piyavskii, 1972, Shubert, 1972] (LO) exploits knowledge of the Lipschitz constant of the function (a bound on the amount that it can change) to prune the search space in order to locate the optimum. In contrast, Bayesian optimization makes a stronger assumption that the unknown function belongs to a known model class (typically a class of smooth functions), the most common being a Gaussian process (GP) generated using a Gaussian or Matérn kernel [Stein, 2012]. We review LO and BO in Section 5.1.

Under their own specific sets of additional assumptions, both BO [Bull, 2011, Theorem 5] and LO [Malherbe and Vayatis, 2017] can be shown to be exponentially faster than random search strategies. If the underlying function is close to satisfying the stronger BO assumptions, then typically BO is able to optimize functions faster than LO. However, when these assumptions are not reasonable, BO may converge slower than simply trying random values [Ahmed et al., 2016, Li et al., 2016]. On the other hand, LO makes minimal assumptions (not even requiring differentiability[12]) and simply prunes away values of the parameters that are not compatible with the Lipschitz condition and thus cannot be solutions. This is useful in speeding up simple algorithms like random search. Given a new function to optimizie, it is typically not clear which of these strategies will perform better.

In this chapter, we propose to combine BO and LO to expoit the advantages of both methods. We call this *Lipschitz Bayesian Optimization* (LBO). Specifically, in Section 5.2 we design *mixed acquisition functions* that use Lipschitz continuity in conjunction with existing BO algorithms. We also address the issue of providing a "harmless" estimate of the Lipschitz constant (see Section 5.1.3), which is an important practical issue for any

[12]The absolute value function $f(x) = |x|$ is an example of a simple non-differentiable but Lipschitz-continuous function.

LO method. Our experiments (Section 5.3) indicate that in some settings the addition of estimated Lipschitz information leads to a huge improvement over standard BO methods. This is particularly true for Thompson sampling, which often outperforms other standard acquisition functions when augmented with Lipschitz information. This seems to be because the estimated Lipschitz continuity seems to correct for the well-known problem of over-exploration [Shahriari et al., 2014]. Further, our experiments indicate that it does not hurt to use the Lipschitz information since even in the worst case it does not change the runtime or the performance of the method.

5.1 Background

We consider the problem of maximizing a real-valued function $f(x)$ over a bounded compact set $\mathcal{X}$. We assume that on iteration t ($t = 1{:}T$), an algorithm chooses a point $x_t \in \mathcal{X}$ and then receives the corresponding function value $f(x_t)$. Typically, our goal is to find the largest possible $f(x_t)$ across iterations. We describe two approaches for solving this problem, namely BO and LO, in detail below.

5.1.1 Bayesian Optimization

Following Section 4.1, we assume that $f \sim GP(0, k(x, x'))$. Here $k(x, x')$ is a kernel function which quantifies the similarity between points x and x'. Throughout this chapter, we use the Matern kernel for which $k(x, x') = \sigma^2 \exp\left(-\sqrt{5}r^2\right)\left(1 + \sqrt{5}r + \frac{5r^2}{3}\right)$ where $r = \frac{||x-x'||}{\ell} = \sum_{j=1}^{d} \frac{(x_j - x'_j)^2}{\ell}$. Here the hyper-parameter σ quantifies the amount of noise we expect in the function values. While the hyper-parameter ℓ is referred to as the length-scale and dictates the extent of smoothness we assume about the function f. We can have one length-scale per-dimension, so $r = \sum_{j=1}^{d} \frac{(x_j - x'_j)^2}{\ell_j}$.

As described in Section 4.1, the acquisition function uses the posterior distribution in order to select the next point to evaluate the function at. In the work presented in this chapter, we continue using four of the widely-used acquisition functions: upper confidence bound (UCB) [Srinivas et al., 2010], Thompson sampling (TS) [Thompson, 1933], expected improvement (EI) [Močkus, 1975] and probability of improvement (PI) [Kushner, 1964]

5.1.2 Lipschitz Optimization

As opposed to assuming that the function comes from a specific family of functions, in LO we simply assume that the function cannot change too quickly as we change x. In particular, we say that a function f is Lipschitz-continuous if for all x and y we have

$$|f(x) - f(y)| \leq L||x - y||_2, \tag{5.1}$$

for a constant L which is referred to as the Lipschitz constant. Note that unlike typical priors used in BO (like the Gaussian or Matérn kernel), a function can be non-smooth and still be Lipschitz continuous.

Lipschitz optimization uses this Lipschitz inequality in order to test possible locations for the maximum of the function. In particular, at iteration t the Lipschitz inequality implies that the function's value at any x can be upper and lower bounded for any $i \in [t-1]$ by

$$f(x_i) - L||x - x_i||_2 \leq f(x) \leq f(x_i) + L||x - x_i||_2.$$

Since the above inequality holds simultaneously for all $i \in [t-1]$, for any x the function value $f(x)$ can be bounded as:

$$f^l_{t-1}(x) \leq f(x) \leq f^u_{t-1}(x),$$

where,

$$\begin{aligned} f^l_{t-1}(x) &= \max_{i \in [t-1]} \{f(x_i) - L||x - x_i||_2\} \\ f^u_{t-1}(x) &= \min_{i \in [t-1]} \{f(x_i) + L||x - x_i||_2\} \end{aligned} \tag{5.2}$$

Notice that if $f^u_{t-1}(x) \leq y^*_{t-1}$, then x *cannot* achieve a higher function value than our current maximum y^*_{t-1}.

Malherbe and Vayatis [2017] exploits these bounds by sampling points x_p uniformly at random from $\mathcal{X}$ until it finds an x_p that satisfies $f^u_{t-1}(x_p) \geq y^*_{t-1}$. If we know the Lipschitz constant L (or use a valid upper bound on the minimum L value), this strategy may prune away large areas of the space while guaranteeing that we do not prune away any optimal solutions. This can substantially decrease the number of function values needed to come close to the global optimum compared to using random points without pruning.

A major drawback of Lipschitz optimization is that in most applications we do not know a valid L. We discuss this scenario in the next section, but first we note that there exist applications where we do have access to a valid L. For example, Bunin and François [2016] discuss cases where L can be dictated by the physical laws of the underlying process (e.g., in heat transfer, solid oxide fuel-cell system, and polymerization). Alternately, if we have a lower and an upper bound on the possible values that the function can take, then we can combine this with the size of $\mathcal{X}$ to obtain an over-estimate of the minimum L value.

5.1.3 Harmless Lipschitz Optimization

When our black-box functions arises from a real world process, a suitable value of L is typically dictated by physical limitations of the process. However, in practice we often do not know L and thus need to estimate it. A simple way to obtain an under-estimate L^{lb}_t of L at iteration t is to use the maximum value that satisfies the Lipschitz inequality across all pairs of points,

$$L^{lb}_t = \max_{i,j \in [t]; x_i \neq x_j} \left\{ \frac{|f(x_i) - f(x_j)|}{||x_i - x_j||_2} \right\}. \tag{5.3}$$

Note that this estimate monotonically increases as we see more examples, but that it may be far smaller than the true L value. A common variation is to sample several points on a grid (or randomly) to use in the estimate above. Unfortunately, without knowing the Lipschitz constant we do not know how fine this grid should be so in general this may still significantly under-estimate the true quantity.

A reasonable property of any estimate of L that we use is that it is "harmless" in the sense of Chapter 4. Specifically, the choice of L should not make the algorithm converge to the global optimum at a slower speed than random guessing (in the worst case). If we have an over-estimate for the minimum possible value of L, then the LO algorithm is harmless as it can only prune values that cannot improve the objective function (although if we over-estimate it by too much then it may not prune much of the space). However, the common under-estimates of L discussed in the previous paragraph are *not* harmless since they may prune the global optima.

We propose a simple solution to the problem that LO is not harmless if we don't have prior knowledge about L: we use a *growing estimate of* L. The danger in using a growing strategy is that if we grow L too slowly then the algorithm may not be harmless. However, in this work, we show that LO is "harmless" for most reasonable strategies for growing L. This result is not prescriptive in the sense that it does not suggest a practical strategy for growing L (since it depends on the true L), but this result shows that even for enormous values of L that an estimate would have to be growing exceedingly slowly in order for it to not be harmless (exponentially-slow in the minimum value of L, the dimensionality, and the desired accuracy). In our experiments we simply use $L_t^{ub} = \kappa t \cdot L_t^{lb}$, the under-estimator multiplied by the (growing) iteration number and a constant κ (a tunable hyper-parameter). In Section 5.3, we observe that this choice of L_t^{ub} with $\kappa = 10$ consistently works well across 14 datasets with 4 different acquisition functions.

5.2 Lipschitz Bayesian optimization

In this section, we show how simple changes to the standard acquisition functions used in BO allow us to incorporate the Lipschitz inequality bounds. We call this Lipschitz Bayesian Optimization (LBO). LBO prevents BO from considering values of x^t that cannot be global maxima (assuming we have over-estimated L) and also restricts the range of $f(x_t)$ values considered in the acquisition function to those that are consistent with the Lipschitz inequalities. Figure 5.1 illustrates the key features of BO, LO, and LBO. It is important to note that the Lipschitz constant L has a different interpretation than the length-scale ℓ of the GP. The constant L specifies an absolute maximum rate of change for the function, while ℓ specifies how quickly a parameterized distance between pairs of points changes the GP. We also note that the computational complexity of using the Lipschitz inequalities is $O(n^2)$ which is cheaper than the $O(n^3)$ cost of (exact) inference in the GP.

We can use the Lipschitz bounds to restrict the limits of the unknown function value for computing the improvement in Equations (4.3) and (4.4). The upper bound U will always be $f^u(x)$, while the lower bound L will depend on the relative value of y^*. In particular, we

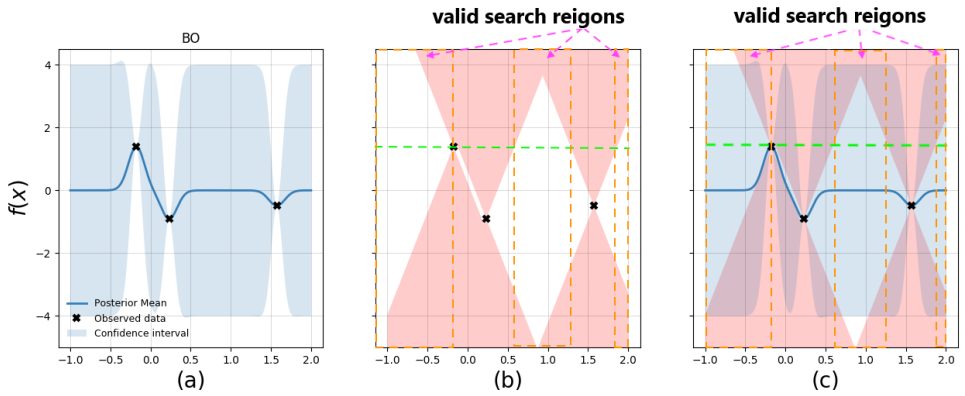


Figure 5.1: Visualization of the effect of incorporating the Lipschitz bounds to BO. a) Shows the posterior mean and confidence interval of the conventional BO. b) The red color represents the regions of the space that are excluded by the Lipschitz bounds. c) Shows the effect of LBO. The Grey color represents the uncertainty. Using LBO helps decrease the uncertainty which prevents over-exploration in unnecessary parts of the space.

have the following two cases:

$$L = \begin{cases} y^*, & \text{if } y^* \in \left(f^l(x), f^u(x)\right) \\ f^u(x), & \text{if } y^* \in \left(f^u(x), \infty\right) \end{cases}.$$

The second case represents points that cannot improve over the current best value (that are "rejected" by the Lipschitz inequalities).

Truncated-PI: We can define a similar variant for the PI acquisition function as:

$$TPI(x) = \Phi\left(z(x, L)\right) - \Phi\left(z(x, U)\right) \tag{5.4}$$

Truncated-EI: Using the above bounds, the truncated expected improvement for point x is given by:

$$\begin{aligned} TEI(x) = -\sigma(x) \cdot z(x, y^*) \left[\Phi(z(x, L)) - \Phi(z(x, U)\right] \\ + \sigma(x) \cdot \left[\phi(z(x, L) - \phi(z(x, U)\right]] \end{aligned} \tag{5.5}$$

Note that removing the Lipschitz bounds corresponds to using $f^l(x) = -\infty$ and $f^u(x) = \infty$, and in this case we recover the usual PI and EI methods in Equations (4.3) and (4.4) respectively.

Truncated-UCB: The same strategy can be applied to UCB as follows:

$$TUCB(x) = \min\left\{\mu(x) + \beta_t^{1/2}\sigma(x), f^u(x)\right\} \tag{5.6}$$

Accept-Reject: An alternative strategy to incorporate the Lipschitz bounds is to use an accept-reject based mixed acquisition function. This approach uses the Lipschitz bounds as a sanity-check to accept or reject the value provided by the original acquisition function, similar to LO methods. Formally, if $g(x)$ is the value of the original acquisition function (e.g. $g(x) = UCB(x)$ or $g(x) = \widetilde{f}(x)$ for TS), then the mixed acquisition function $\overline{g}(x)$ is given as follows:

$$\overline{g}(x) = \begin{cases} g(x), & \text{if } g(x) \in [f^l(x), f^u(x)] \text{ (Accept)} \\ -\infty, & \text{othewise (Reject)} \end{cases}.$$

We refer to the accept-reject based mixed acquisition functions as AR-UCB and AR-TS (respectively). Note that the accept-reject method is quite generic and can be used with any acquisition function that has values on the same scale as that of the function. When using an estimate of L it's possible that a good point could be rejected because the estimate of L is too small, but using a growing estimate ensures that such points can again be selected on later iterations.

5.3 Experiments

Datasets: We perform an extensive experimental evaluation and present results on twelve synthetic datasets and three real-world tasks. For the synthetic experiments, we use the standard global-optimization benchmarks namely the Branin, Camel, Goldstein Price, Hartmann (2 variants), Michalwicz (3 variants) and Rosenbrock (4 variants). The closed form and domain for each of these functions is given in Jamil and Yang [2013]. As examples of real-world tasks, we consider tuning the parameters for a *robot-pushing* simulation (2 variants) [Wang and Jegelka, 2017] and tuning the hyper-parameters for logistic regression [Wu et al., 2017b]. For the robot pushing example, our aim is to find a good pre-image [Kaelbling and Lozano-Pérez, 2017] in order for the robot to push the object to a pre-specified goal location. We follow the experimental protocol from Wang and Jegelka [2017] and use the negative of the distance to the goal location as the black-box function to maximize. We consider tuning the robot position r_x, r_y, and duration of the push t_r for the 3D case. We also tune the angle of the push θ_r to make it a 4 dimensional problem. For the hyper-parameter tuning task, we consider tuning the strength of the ℓ_2 regularization (in the range $[10^{-7}, 0.9]$) , the learning rate for stochastic gradient descent (in the range $[10^{-7}, 0.05]$) and the number of passes over the data (in the range $[2, 15]$). The black-box function is the negative loss on the test set (using a train/test split of $80\%/20\%$) for the MNIST dataset.
Experimental Setup: For Bayesian optimization, we use a Gaussian Process prior with the Matérn kernel (with a different length scale for each dimension). We modified the publically available BO package *pybo* of Hoffman and Shahriari [2014] to construct the mixed acquisition functions. Our modifications to pybo include but are not limited to:

- In order to make the optimization invariant to the scale of the function values, we standardize the function values; after each iteration, we center the observed function

values by subtracting their mean and dividing by their standard deviation. We then fit a GP to these rescaled function values and correct for our Lipschitz constant estimate by dividing it by the standard deviation.

- Detecting failures: if the point chosen by BO is within 10^{-12} from any previously chosen point then BO is stuck with the same pints so do random search. This made big improvement in the performance of pybo.
- Detecting and handling any numerical instabilities that resulted in non-positive posterior variance.
- Trying DIRECT with different computational time budgets and random search to optimize the acquisition function.
- Trying leave-one-out cross-validation to optimize the hyperparameters of the GP.
- Added the options to shift the bounds of the test problem to test the robustness of the algorithm.
- The option to explore various methods of combining LO and BO.

All the prior hyper-parameters were set and updated across iterations according to the open-source Spearmint package[13].A constant mean GP was used with a prior N (0,1). The length scale ℓ had a uniform prior between 10^{-11} and 10. The output variance σ^2 was set with a lognormal(0,1) prior. We use DIRECT [Jones et al., 1993] in order to optimize the acquisition function in each iteration. This is one of the standard choices in current works on BO [Eric et al., 2008, Mahendran et al., 2012, Martinez-Cantin et al., 2007], but we expect that Lipschitz information could improve the performance under other choices of the acquisition function optimization approach such as discretization [Snoek et al., 2012], adaptive grids [Bardenet and Kégl, 2010], and other gradient-based methods [Hutter et al., 2011, Lizotte et al., 2012]. In order to ensure that Bayesian optimization does not get stuck in sub-optimal maxima (either because of the auxiliary optimization or a "bad" set of hyper-parameters), on every fourth iteration of BO (or LBO) we choose a random point to evaluate rather than optimizing the acquisition function. This makes the optimization procedure "harmless" in the sense that BO (or LBO) will not perform worse than random search as described in Chapter 4. This has become common in recent implementations of BO methods such as Bull [2011], Hutter et al. [2011]; and Falkner et al. [2017], and to make the comparison fair we add this "exploration" step to all methods. Note that in the case of LBO we may need to reject random points until we find one satisfying the Lipschitz inequalities (this does not require evaluating the function). In practice, we found that both the standardization and iterations of random exploration are essential for good performance.[14] All our results are averaged over 10 independent runs, and each of our figures plots the mean

[13] https://github.com/hips/spearmint

[14] Note that we verified that our baseline version of BO performs better than or equal to Spearmint across benchmark problems.

and standard deviation of the absolute error (compared to the global optimum) versus the number of function evaluations. For functions evaluated on log scale, we show the mean and the 10^{th} and 90^{th} quantiles.

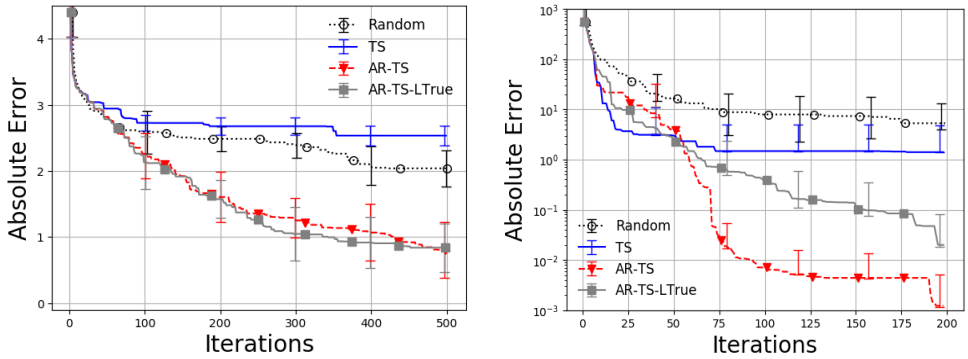
Algorithms compared: We compare the performance of the BO and LBO methods for the EI, PI, UCB and TS acquisition functions. For UCB, we set the trade-off parameter β according to Kandasamy et al. [2017]. For EI and PI, we use Lipschitz bounds to truncate the range of function values for calculating the improvement and use the LBO variants TEI and TPI respectively. For UCB and TS, we use the accept-reject strategy and evaluate the LBO variants AR-UCB and AR-TS respectively. In addition to these, we use random exploration as baseline. We chose the hyper-parameter κ (that controls the extent of over-estimating the Lipschitz constant) on the Rosenbrock-4D function and use the best value of κ for all the other datasets and acquisition functions for both BO and LBO. In particular, we set $\kappa = 10$.

Results: To make the results easier to read, we divide the results into the following groups:

1. Functions where LBO provides huge improvements over BO shown in Figure 5.2. Overall, these represent 21% of all the test cases.
2. Functions where LBO provides improvements over BO shown in Figure 5.3. Overall, these represent 9% of all the test cases.
3. Functions where LBO performs similar to BO shown in Figure 5.4. Overall, this represents 60% of all the test cases.
4. Functions where LBO performs slightly worse than BO shown in Figure 5.5. Overall, these represent 10% of all the test cases.

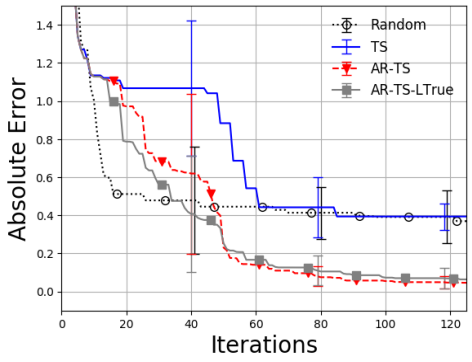
A comparison between the performance across different acquisition functions, (for both BO and LBO) on some of the benchmark functions, is presented Figure 5.6. Finally, we show the performance for UCB when β is misspecified in Figure 5.7. In Appendix B.1, we show the results for all the experiments on all the datasets for each of the four acquisition functions. From these experiments, we can observe the following:

- LBO can potentially lead to large gains in performance across acquisition functions and datasets, especially, for TS.
- Across datasets, we observe that the gains for EI are relatively small, they are occasionally large for PI and UCB and tend to be consistently large for TS. This can be explained as follows: using EI results in under-exploration of the search space, a fact that has been consistently observed and even theoretically proven in Qin et al. [2017]. As a result of this, there are less "bad" regions to prune using the Lipschitz inequality, resulting in small gains from LBO.
- TS suffers from exactly the opposite problem: it results in high variance leading to over-exploration of the search space and poor performance. This can be observed in

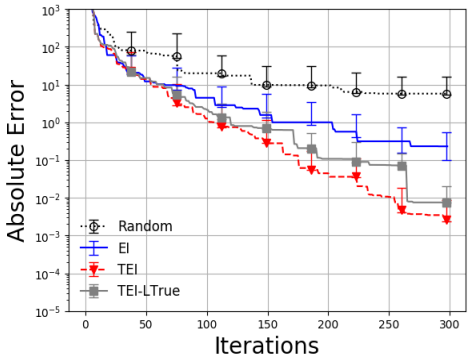


(a) Michalwicz 5D with TS

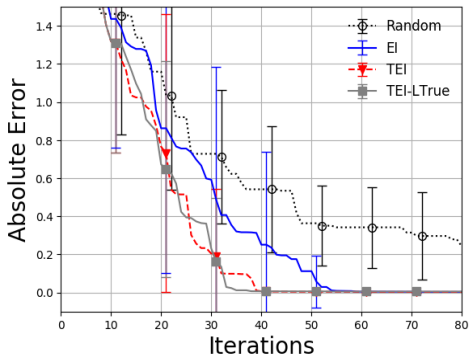
(b) Rosenbrock 3D with TS



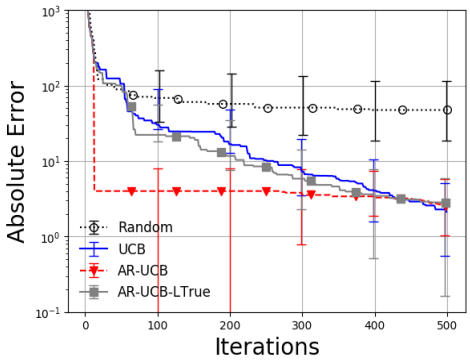
(c) Robot pushing 3D with TS



(d) Goldstein 2D with EI

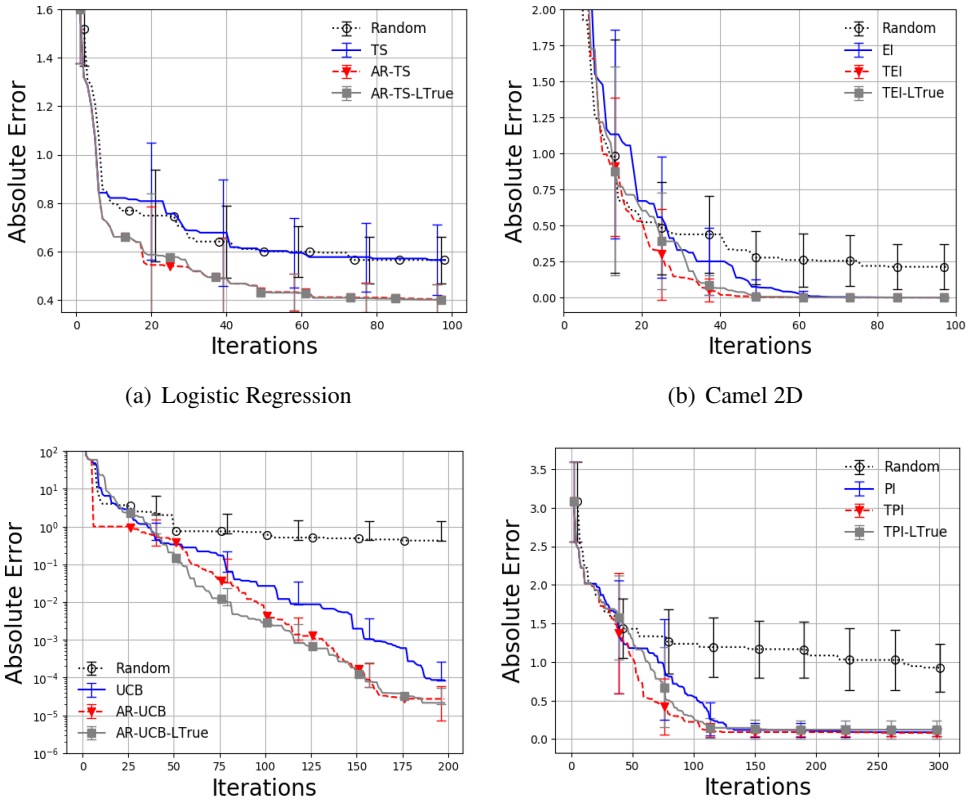


(e) Hartmann 3D with EI



(f) Rosenbrock 5D with UCB

Figure 5.2: Examples of functions where LBO provides huge improvements over BO for the different acquisition functions. The figure also shows the performance of random search and LBO using the *True* Lipschitz constant.



(a) Logistic Regression

(b) Camel 2D

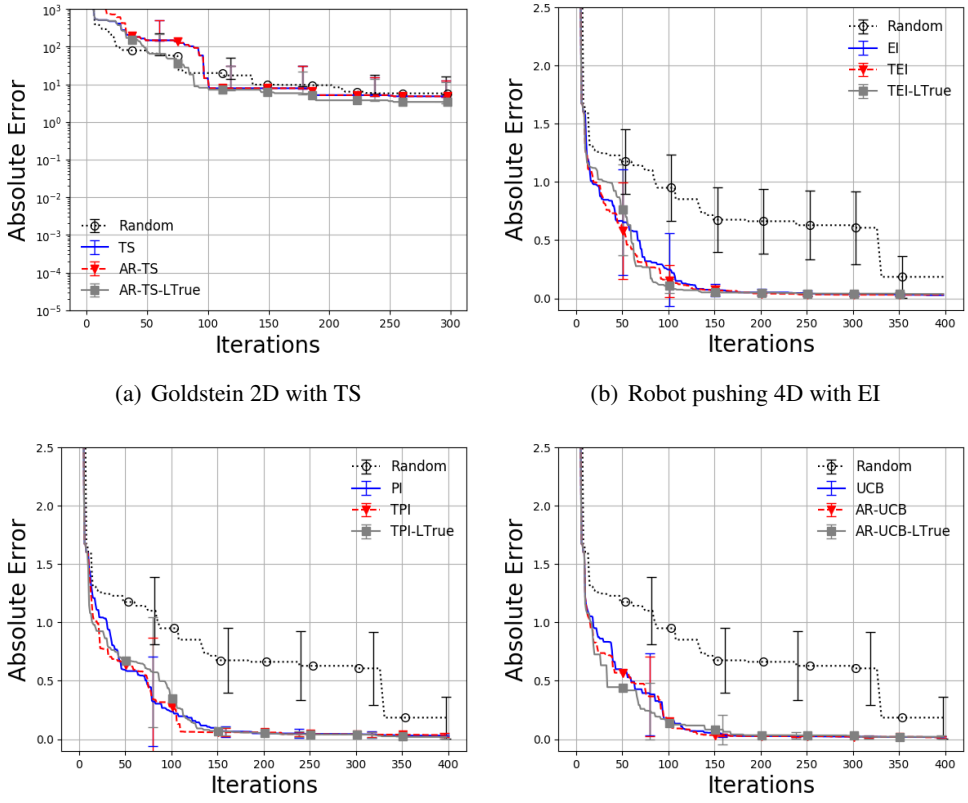
(c) Rosenbrock 2D

(d) Hartmann 6D

Figure 5.3: Examples of functions where LBO provides some improvements over BO for the different acquisition functions. The figure also shows the performance of random search and LBO using the *True* Lipschitz constant.

Figures 5.2(a), 5.2(b) and 5.2(c) where the performance of TS is near random. This has also been observed and noted in Shahriari et al. [2016]. For the discrete multi-armed bandit case, Chapelle and Li [2011] multiply the obtained variance estimate by a small number to discourage over-exploration and show that it leads to better results. LBO offers a more principled way of obtaining this same effect and consequently results in making TS more competitive with the other acquisition functions.

- The only functions where it slightly hurts are Rosenbrock-4D and Goldstein-2D with UCB and PI.
- For Michalwicz-5D (Figure 5.6(a)), we see that there is no gain for EI, PI, or UCB. However, the gain is huge for TS functions. In fact, even though TS is the worst performing acquisition function on this dataset, it's LBO variant, AR-TS gives the best performance across all methods. This demonstrates the possible gain that can be



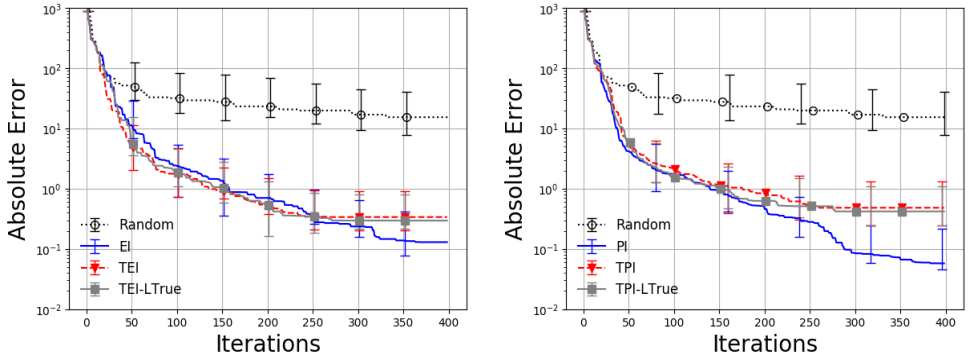
(a) Goldstein 2D with TS

(b) Robot pushing 4D with EI

(c) Robot pushing 4D with PI

(d) Robot pushing 4D with UCB

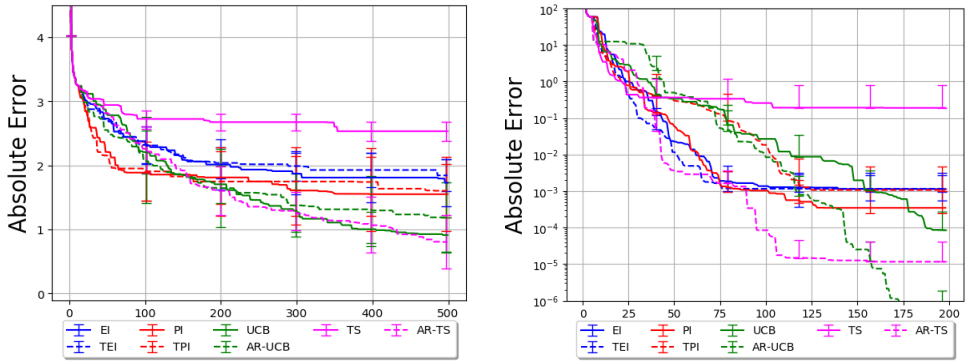
Figure 5.4: Examples of functions where LBO performs similar to BO for the different acquisition functions.



(a) Rosenbrock 4D with EI

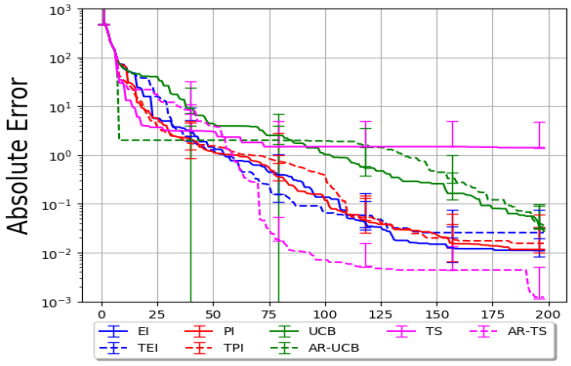
(b) Rosenbrock 4D with PI

Figure 5.5: Examples of functions where BO slightly performs better than LBO.



(a) Michalwicz 5D

(b) Rosenbrock 2D



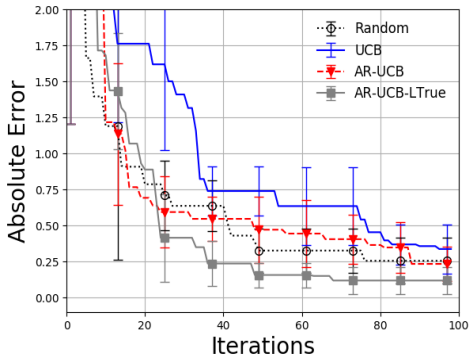
(c) Rosenbrock 3D

Figure 5.6: Examples of functions where LBO boosts the performance of BO with TS. (All figures are better seen in color)

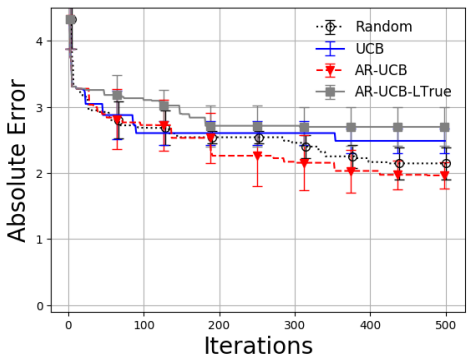
obtained from using our mixed acquisition functions.

- We observe a similar trend in Figures 5.6(b) and 5.6(c) where LBO improves TS from near-random performance to being competitive with the best performing methods and does not adversely affect the methods performing well.
- For the cases where BO performs slightly better than LBO, we notice that the True estimate of L provides compararble performance to BO, so the problem can be narrowed down to finding a good estimate of L.
- Figure 5.7 shows examples where LBO saves BO with UCB when the parameter β is chosen too large. In this case BO performs near random, but using LBO leads to better performance than random search.

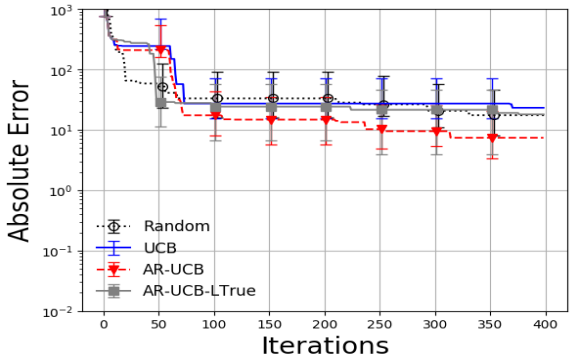
In any case, our experiments indicate that LBO methods rarely hurt the performance of the original acquisition function. Since they have minimal computational or memory



(a) Camel 2D with UCB



(b) Michalwicz 5D with UCB



(c) Rosenbrock 5D with UCB

Figure 5.7: Examples of functions where LBO outperforms BO with UCB when the β parameter is too large ($\beta = 10^{16}$).

requirements and are simple to implement, these experiments support using the Lipschitz bounds.

5.4 Related work

The Lipschitz condition has been used with BO under different contexts in two previous works [González et al., 2016, Sui et al., 2015]. The aim of Sui et al. [2015] is to design a "safe" BO algorithm. They assume knowledge of the true Lipschitz constant and exploit Lipschitz continuity to construct a safety threshold in order to construct a "safe" region of the parameter space. This is different than our goal of improving the performance of existing BO methods, and also different in that we estimate the Lipschitz constant as we run the algorithm. On the other hand, González et al. [2016] used Lipschitz continuity to model interactions between a batch of points chosen simultaneously in every iteration of BO (referred to as "Batch" Bayesian optimization). This contrasts with our work where we

are aiming to improve the performance of existing sequential algorithms (it's possible that our ideas could be used in their framework).

5.5 Discussion

In this chapter, we have proposed simple ways to combine Lipschitz inequalities with some of the most common BO methods. Our experiments show that this often gives a performance gain, and in the worst case it performs similar to a standard BO method. Although we have focused on four of the simplest acquisition functions, it seems that these inequalities could be used within other acquisition functions. From the experimental results, we should always use LBO, especially, with TS. Moreover, with better methods to estimate L, we expect more performance gain for all the acquisition functions.

Chapter 6

Conclusions and Future Work

This thesis presents our work on practical optimization methods for structured machine learning problems. Our work covered problems for both training and hyperparameter search of ML models.

6.1 Stochastic Average Gradient

We presented the first work that applies SAG to CRFs. Due to its memory requirements, it may be difficult to apply the SAG algorithm for natural language applications involving complex features that depend on a large number of labels. However, grouping training examples into mini-batches can also reduce the memory requirement (since only the gradients with respect to the mini-batches would be needed).

We believe linearly-convergent stochastic gradient algorithms with non-uniform sampling could give a substantial performance improvement in a large variety of CRF training problems, and we emphasize that the method likely has extensions beyond what we have examined. For example, we have focused on the case of ℓ_2-regularization but for large-scale problems there is substantial interest in using ℓ_1-regularized CRFs [Lavergne et al., 2010, Tsuruoka et al., 2009, Zhou et al., 2011]. Fortunately, such non-smooth regularizers can be handled with a proximal-gradient variant of the method, see Defazio et al. [2014a]. While we have considered chain-structured data the algorithm applies to general graph structures, and any method for computing/approximating the marginals could be adopted. Moreover, the SAG algorithm could be modified to use multi-threaded computation as in the algorithm of Lavergne et al. [2010], and indeed might be well-suited to massively distributed parallel implementations.

6.2 Stochastic Variance Reduced Gradient

As for SVRG, it is the only memory-free method among the new stochastic methods with linear convergence rates. It represents the natural method to use for a huge variety of machine learning problems. Future research directions include using SVRG for nonconvex problems. This can be useful in several problems such as training deep learning models [Allen-Zhu and Hazan, 2016, Reddi et al., 2016] or in reinforcement learning [Xu et al., 2017].

6.3 Bayesian Optimization

We have proposed three ideas to improve BO. Our experiments show that our proposed methods provides performance gain over conventional BO methods. Future research ideas include designing more powerful harmless BO methods. One possible research direction is to use a cost function that measures the exploration part of the algorithm. The idea here is to make sure that the chosen points cover different regions of the space. At the same time, the chosen points should not be too close. How far and how close the points should be, these are questions that require further investigation.

As for LBO, better methods for estimating L can improve the performance of this algorithm. Although we have focused on four of the simplest acquisition functions, it seems that the Lipschitz inequalities could be used within other acquisition functions. Moreover, we expect that these inequalities could also be used in other settings like BO with constraints [Gardner et al., 2014, Gelbart et al., 2014, Hernández-Lobato et al., 2016], BO methods based on other model classes like neural networks [Snoek et al., 2015] or random forests [Hutter et al., 2011], methods that evaluate more than one x_t at a time [Ginsbourger et al., 2010, Wang et al., 2016]. Finally, if the gradient is Lipschitz continuous then it is possible to use the descent lemma Bertsekas [2016] to obtain Lipschitz bounds that depend on both function values and gradients.

6.3.1 Final Comment on Implementing BO Algorithms

For the BO work, we have spent a long time optimizing the baseline. Reading a lot of research papers, we find that some papers do not tune the baseline. Thus, it is easy to develop methods that seems to work. We recommend that researchers compare their implementations to the packages available online. For example, using the Rosenbrock-2D function, if the baseline can not be within 0.001 from f^* after 200 iterations, then the baseline needs improvement. Furthermore, we found that a lot of the computational tricks can make a huge difference in performance. These tricks are not reported in papers. For example, standardizing the function values. All of these points can facilitate reproducible research and will provide a better test environment for new research ideas.

To guarantee that our implementation is reproducible, in the following, we summarize our BO implementation choices:

- The GP mean function was chosen as a constant with a prior $N\ (0, 1)$.
- Matern kernel with $\nu = 5/2$.
- length scale ℓ with a uniform prior between 10^{-11} and 10.
- output variance σ^2 with a lognormal(0,1) prior.
- GP hyperparameters are represented with 10 MCMC samples that are updated at every iteration.

We have chosen these parameters after doing a lot of experiments. These are the most robust choices for the GP implementation that are not affected by the test function which is unknown to BO. From our BO experiments, we found that:

- Matern kernel works better than SE.
- Standardizing the function values is crucial to the performance.
- Using MCMC is better than using Type II Maximum Likelihood for GP hyperparameter optimization.
- Constant mean is better than zero mean or using a mean function.
- Using the furthest point instead of random points for HBO is still harmless. However, there was no gain than using random points.

6.4 Future Work

This thesis presented our work on designing better optimization algorithms for machine learning problems. This involves both the training phase and the hyperparameter tuning phase. For the training problem, future work includes investigating the use of the proposed algorithms for deep learning. Several works have attempted to apply SVRG and its variants to train deep learning models. However, SGD and its variants are still the dominant approaches. The contributions in this thesis can still be used to speed up training the last layer and may eventually lead to better deep learning methods. On the other hand, because of the large memory requirements of SAG, it may not be suitable for deep learning. However, we have demonstrated its efficiency in training CRFs.

As for BO, due to the simplicity of the proposed methods, many existing implementations could easily incorporate them. Future directions include better incorporation of the Lipschitz optimization with BO such as using a different Lipschitz constant for each direction, and designing better harmless BO methods such as using furthest points and automatic detection of BO failures (for example, if BO picks points that are very close to each other).

Bibliography

R. J. Adler. *The geometry of random fields*, volume 62. Siam, 2010.

M. O. Ahmed, B. Shahriari, and M. Schmidt. Do we need "harmless" bayesian optimization and "first-order" bayesian optimization? *NIPS Workshop on Bayesian Optimization*, 2016.

Z. Allen-Zhu and E. Hazan. Variance reduction for faster non-convex optimization. In *International Conference on Machine Learning (ICML)*, pages 699–707, 2016.

A. Aravkin, M. P. Friedlander, F. J. Herrmann, and T. Van Leeuwen. Robust inversion, dimensionality reduction, and randomized sampling. *Mathematical Programming*, 134 (1):101–125, 2012.

F. Bach and E. Moulines. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. *Advances in Neural Information Processing Systems (NIPS)*, pages 451–459, 2011.

P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

R. Bardenet and B. Kégl. Surrogating the surrogate: accelerating gaussian-process-based global optimization with a mixture cross-entropy algorithm. In *International Conference on Machine Learning (ICML)*, pages 55–62. Omnipress, 2010.

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

Y. Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8): 1889–1900, 2000.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

D. P. Bertsekas. *Nonlinear Programming*. MIT, 3rd edition, 2016.

N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

A. D. Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12(Oct):2879–2904, 2011.

G. A. Bunin and G. François. Lipschitz constants in experimental optimization. *arXiv preprint arXiv:1603.07847*, 2016.

R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu. Sample size selection in optimization methods for machine learning. *Mathematical programming*, 134(1):127–155, 2012.

P. Carbonetto. *New probabilistic inference algorithms that harness the strengths of variational and Monte Carlo methods*. PhD thesis, University of British Columbia, May 2009.

R. Caruana, T. Joachims, and L. Backstrom. KDD-cup 2004: results and analysis. *ACM SIGKDD Newsletter*, 6(2):95–108, 2004.

O. Chapelle and L. Li. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2249–2257, 2011.

T. Cohn and P. Blunsom. Semantic role labelling with tree conditional random fields. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 169–172. Association for Computational Linguistics, 2005.

M. Collins, A. Globerson, T. Koo, X. Carreras, and P. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *The Journal of Machine Learning Research*, 9:1775–1822, 2008.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12 (Aug):2493–2537, 2011.

G. V. Cormack and T. R. Lynam. Spam corpus creation for TREC. In *Proc. 2nd Conference on Email and Anti-Spam*, 2005. `http://plg.uwaterloo.ca/~gvcormac/treccorpus/`.

A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in Neural Information Processing Systems (NIPS)*, 2014a.

A. J. Defazio, A. AU, T. S. Caetano, N. C. AU, and J. Domke. Finito: A faster, permutable incremental gradient method for big data problems. *International Conference on Machine Learning (ICML)*, 2014b.

J. Dodge, K. Jamieson, and N. A. Smith. Open loop hyperparameter optimization and determinantal point processes. *arXiv preprint arXiv:1706.01566*, 2017.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

M. Eligius and G.-T. Boglárka. *Introduction to nonlinear and global optimization*. Springer, 2010.

B. Eric, N. D. Freitas, and A. Ghosh. Active preference learning with discrete choice data. In *Advances in Neural Information Processing Systems (NIPS)*, pages 409–416, 2008.

S. Falkner, A. Klein, and F. Hutter. Combining hyperband and bayesian optimization. In *NIPS Workshop on Bayesian Optimization*, 2017.

J. R. Finkel, A. Kleeman, and C. D. Manning. Efficient, feature-based, conditional random field parsing. *Annual Meeting of the Association for Comptuational Linguistics: Human Language Technologies*, 2008.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

P. I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

M. P. Friedlander and M. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal of Scientific Computing*, 34(3):A1351–A1379, 2012.

J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. P. Cunningham. Bayesian optimization with inequality constraints. In *International Conference on Machine Learning (ICML)*, pages 937–945, 2014.

M. A. Gelbart, J. Snoek, and R. P. Adams. Bayesian optimization with unknown constraints. In *Uncertainty in Artificial Intelligence (UAI)*, pages 250–259, Arlington, Virginia, United States, 2014.

D. Ginsbourger, R. Le Riche, and L. Carraro. Kriging is well-suited to parallelize optimization. In *Computational intelligence in expensive optimization problems*, pages 131–162. Springer, 2010.

J. González, Z. Dai, P. Hennig, and N. Lawrence. Batch bayesian optimization via local penalization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 648–657, 2016.

I. Guyon. Sido: A phamacology dataset, 2008. URL `http://www.causality.inf.ethz.ch/data/SIDO.html`.

R. Harikandeh, M. O. Ahmed, A. Virani, M. Schmidt, J. Konečnỳ, and S. Sallinen. Stopwasting my gradients: Practical svrg. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2251–2259, 2015.

P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837, 2012.

J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 918–926, 2014.

J. M. Hernández-Lobato, M. A. Gelbart, R. P. Adams, M. W. Hoffman, and Z. Ghahramani. A general framework for constrained bayesian optimization using information-based search. *Journal of Machine Learning Research*, 2016.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

M. W. Hoffman and B. Shahriari. Modular mechanisms for bayesian optimization. *NIPS Workshop on Bayesian Optimization*, 2014.

C. Hu, J. Kwok, and W. Pan. Accelerated gradient methosd for stochastic optimization and online learning. *Advances in Neural Information Processing Systems (NIPS)*, 2009.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

M. Jamil and X.-S. Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in Neural Information Processing Systems (NIPS)*, 2013.

D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

L. P. Kaelbling and T. Lozano-Pérez. Pre-image backchaining in belief space for mobile manipulation. In *Robotics Research*, pages 383–400. Springer, 2017.

K. Kandasamy, A. Krishnamurthy, J. Schneider, and B. Poczos. Asynchronous parallel bayesian optimisation via thompson sampling. *arXiv preprint arXiv:1705.09236*, 2017.

S. Keerthi and D. DeCoste. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6:341–361, 2005.

J. Konečnỳ and P. Richtárik. Semi-stochastic gradient descent methods. *Frontiers in Applied Mathematics and Statistics*, 2017.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate frank-wolfe optimization for structural svms. *International Conference on Machine Learning (ICML)*, 2013.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *International Conference on Machine Learning (ICML)*, 2001.

M. Längkvist, L. Karlsson, and A. Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

T. Lavergne, O. Cappé, and F. Yvon. Practical very large scale CRFs. In *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 504–513, 2010.

N. Le Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for strongly-convex optimization with finite training sets. *Advances in Neural Information Processing Systems (NIPS)*, 2012.

D. Lewis, Y. Yang, T. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Efficient hyperparameter optimization and infinitely many armed bandits. *arXiv preprint arXiv:1603.06560*, 2016.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

D. Lizotte. *Practical Bayesian optimization*. PhD thesis, University of Alberta, 2008.

D. J. Lizotte, R. Greiner, and D. Schuurmans. An experimental methodology for response surface optimization methods. *Journal of Global Optimization*, 53(4):699–736, 2012.

S. Lohr. *Sampling: design and analysis*. Cengage Learning, 2009.

D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. *International Conference on Machine Learning (ICML)*, 2015.

M. Mahdavi and R. Jin. Mixedgrad: An $o(1/t)$ convergence rate algorithm for stochastic smooth optimization. *Advances in Neural Information Processing Systems (NIPS)*, 2013.

N. Mahendran, Z. Wang, F. Hamze, and N. De Freitas. Adaptive mcmc with bayesian optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 751–760, 2012.

J. Mairal. Optimization with first-order surrogate functions. *International Conference on Machine Learning (ICML)*, 2013.

C. Malherbe and N. Vayatis. Global optimization of lipschitz functions. In *International Conference on Machine Learning (ICML)*, pages 2314–2323, 2017. URL `http://proceedings.mlr.press/v70/malherbe17a.html`.

R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems*, volume 3, pages 321–328, 2007.

J. R. Martins, P. Sturdza, and J. J. Alonso. The complex-step derivative approximation. *ACM Transactions on Mathematical Software (TOMS)*, 29(3):245–262, 2003.

A. McCallum, K. Rohanimanesh, and C. Sutton. Dynamic conditional random fields for jointly labeling multiple sequences. In *NIPS Workshop on Syntax, Semantics, Statistics*, 2003.

J. Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

M. D. Morris, T. J. Mitchell, and D. Ylvisaker. Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics*, 35(3):243–255, 1993.

A. Nedic and D. Bertsekas. Convergence rate of incremental subgradient algorithms. In *Stochastic Optimization: Algorithms and Applications*, pages 263–304. Kluwer Academic, 2000.

D. Needell, N. Srebro, and R. Ward. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Advances in Neural Information Processing Systems (NIPS)*, 2014.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

Y. Nesterov. *Introductory lectures on convex optimization: A basic course*. Springer, 2004.

Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM J. Optim.*, 22(2):341–362, 2012.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, second edition, 2006.

S. Nowozin and C. H. Lampert. Structured learning and prediction in computer vision. *Foundation and Trends in Computer Vision*, 6, 2011.

M. Osborne. *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature*. PhD thesis, University of Oxford, 2010.

A. Papoulis and S. U. Pillai. *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 2002.

F. Peng and A. McCallum. Information extraction from research papers using conditional random fields. *Information Processing & Management*, 42(4):963–979, 2006.

J. D. Pintér. Global optimization in action. *Scientific American*, 264:54–63, 1991.

S. Piyavskii. An algorithm for finding the absolute extremum of a function. *USSR Computational Mathematics and Mathematical Physics*, 12(4):57–67, 1972.

B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

C. Qin, D. Klabjan, and D. Russo. Improving the expected improvement algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5387–5397, 2017.

C. E. Rasmussen and C. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.

B. Recht and C. Ré. Beneath the valley of the noncommutative arithmetic-geometric mean inequality: conjectures, case-studies, and consequences. *arXiv preprint arXiv:1202.4184*, 2012.

S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola. Stochastic variance reduction for nonconvex optimization. In *International Conference on Machine Learning (ICML)*, pages 314–323, 2016.

L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

S. Rosset and J. Zhu. Piecewise linear regularized solution paths. *The Annals of Statistics*, 35(3):1012–1030, 2007.

M. Schmidt, N. Le Roux, and F. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. *Advances in Neural Information Processing Systems (NIPS)*, 2011.

M. Schmidt, R. Babanezhad, M. O. Ahmed, A. Defazio, and A. Sarkar. Non-uniform stochastic average gradient method for training conditional random fields. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

B. Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, 2004.

F. Sha and F. Pereira. Shallow parsing with conditional random fields. *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technology*, 2003.

B. Shahriari, Z. Wang, M. W. Hoffman, A. Bouchard-Côté, and N. de Freitas. An entropy search portfolio. In *NIPS Workshop on Bayesian Optimization*, 2014.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2016.

S. Shalev-Schwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599, 2013.

S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: primal estimated subgradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

B. O. Shubert. A sequential method seeking the global maximum of a function. *SIAM Journal on Numerical Analysis*, 9(3):379–388, 1972.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems (NIPS)*, 2012.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning (ICML)*, pages 2171–2180, 2015.

E. Solak, R. Murray-Smith, W. E. Leithead, D. J. Leith, and C. E. Rasmussen. Derivative observations in gaussian process models of dynamic systems. *Advances in Neural Information Processing Systems (NIPS)*, 2003.

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, pages 1015–1022, 2010.

M. L. Stein. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 2012.

T. Strohmer and R. Vershynin. A randomized Kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262–278, 2009.

Y. Sui, A. Gotovos, J. Burdick, and A. Krause. Safe exploration for optimization with gaussian processes. In *International Conference on Machine Learning (ICML)*, pages 997–1005, 2015.

B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. *Advances in Neural Information Processing Systems (NIPS)*, 2003.

M. Teichmann, M. Weber, J. M. Zöllner, R. Cipolla, and R. Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. *CoRR*, abs/1612.07695, 2016.

P. Thodoroff, J. Pineau, and A. Lim. Learning robust features using deep learning for automatic seizure detection. In *Machine learning for healthcare conference*, pages 178–190, 2016.

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

Y. Tsuruoka, J. Tsujii, and S. Ananiadou. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. *Annual Meeting of the Association for Computational Linguisitics*, pages 477–485, 2009.

N. Usunier, A. Bordes, and L. Bottou. Guarantees for approximate incremental svms. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 884–891, 2010.

K. van den Doel and U. Ascher. Adaptive and stochastic algorithms for EIT and DC resistivity problems with piecewise constant solutions and many measurements. *SIAM J. Scient. Comput*, 34, 2012.

J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509, 2009.

S. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. *International Conference on Machine Learning (ICML)*, 2006.

H. Wallach. Efficient training of conditional random fields. Master's thesis, University of Edinburgh, 2002.

J. Wang, S. C. Clark, E. Liu, and P. I. Frazier. Parallel bayesian global optimization of expensive functions. *arXiv preprint arXiv:1602.05149*, 2016.

Z. Wang and S. Jegelka. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning (ICML)*, 2017.

A. Wu, M. C. Aoi, and J. W. Pillow. Exploiting gradients and hessians in bayesian optimization and bayesian quadrature. *arXiv preprint arXiv:1704.00060*, 2017a.

J. Wu. *Knowledge Gradient Methods for Bayesian Optimization*. PhD thesis, Cornell University, 2017.

J. Wu and P. I. Frazier. Discretization-free knowledge gradient methods for bayesian optimization. *arXiv preprint arXiv:1707.06541*, 2017.

J. Wu, M. Poloczek, A. G. Wilson, and P. I. Frazier. Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems (NIPS)*, 2017b.

T. Xu, Q. Liu, and J. Peng. Stochastic variance reduction for policy gradient estimation. *arXiv preprint arXiv:1710.06034*, 2017.

W. Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint*, 2010.

J. Zhou, X. Qiu, and X. Huang. A fast accurate two-stage training algorithm for L1-regularized CRFs with heuristic line search strategy. *International Joint Conference on Natural Language Processing*, 2011.

J. Zhu and E. Xing. Conditional Topic Random Fields. In *International Conference on Machine Learning (ICML)*, 2010.

Appendix A

Chapter 4 Supplementary Material

A.1 Additional Experimental Results

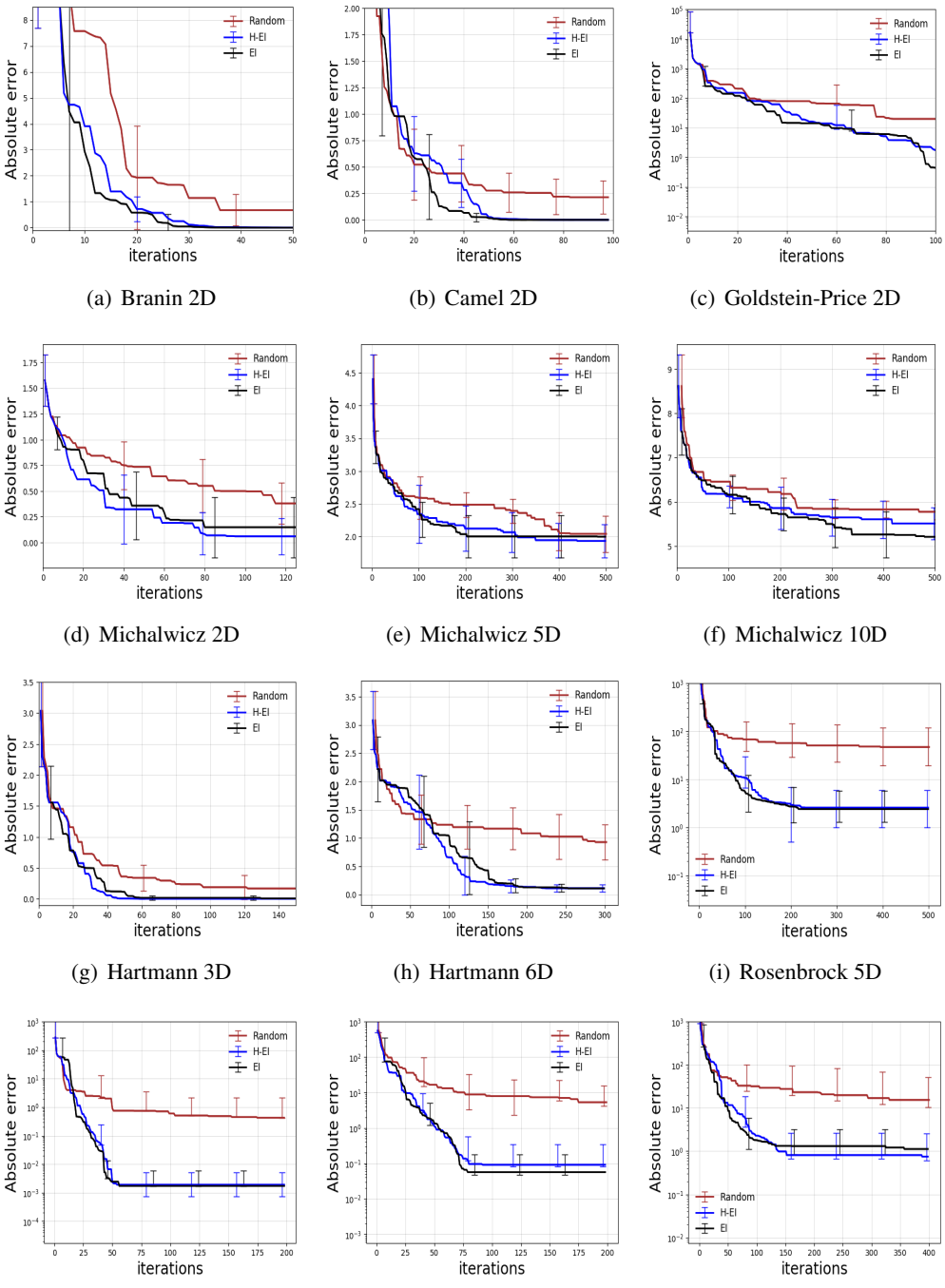
Below we show the results for all the datasets as follows:

- Figure A.1 shows the performance of Random search, BO, and HBO for the EI acquisition function.
- Figure A.2 shows the performance of Random search, BO, HBO for the PI acquisition function.
- Figure A.3 shows the performance of Random search, BO, and HBO for the UCB acquisition function.
- Figure A.4 shows the performance of Random search, BO, and HBO for the TS acquisition function.

In these figures 'H-*' stands for the harmless BO version with a certain acquisition function, for example, 'H-TS' represents Harmless BO with Thompson sampling acquisition function.
All the results of the FOBO experiments are shown as follows:

- Figure A.5 shows the performance of Random search, BO, and FOBO for the EI acquisition function.
- Figure A.6 shows the performance of Random search, BO, and FOBO for the PI acquisition function.
- Figure A.7 shows the performance of Random search, BO, and FOBO for the UCB acquisition function.
- Figure A.8 shows the performance of Random search, BO, and FOBO for the EI acquisition function in terms of the number of function evaluations.
- Figure A.9 shows the performance of Random search, BO, and FOBO for the PI acquisition functionin terms of the number of function evaluations.
- Figure A.10 shows the performance of Random search, BO, and FOBO for the UCB acquisition functionin terms of the number of function evaluations.

In these figures 'G-*' and 'D-*' stands for BO with gradient and BO with directional derivatives, respectively, with a certain acquisition function.



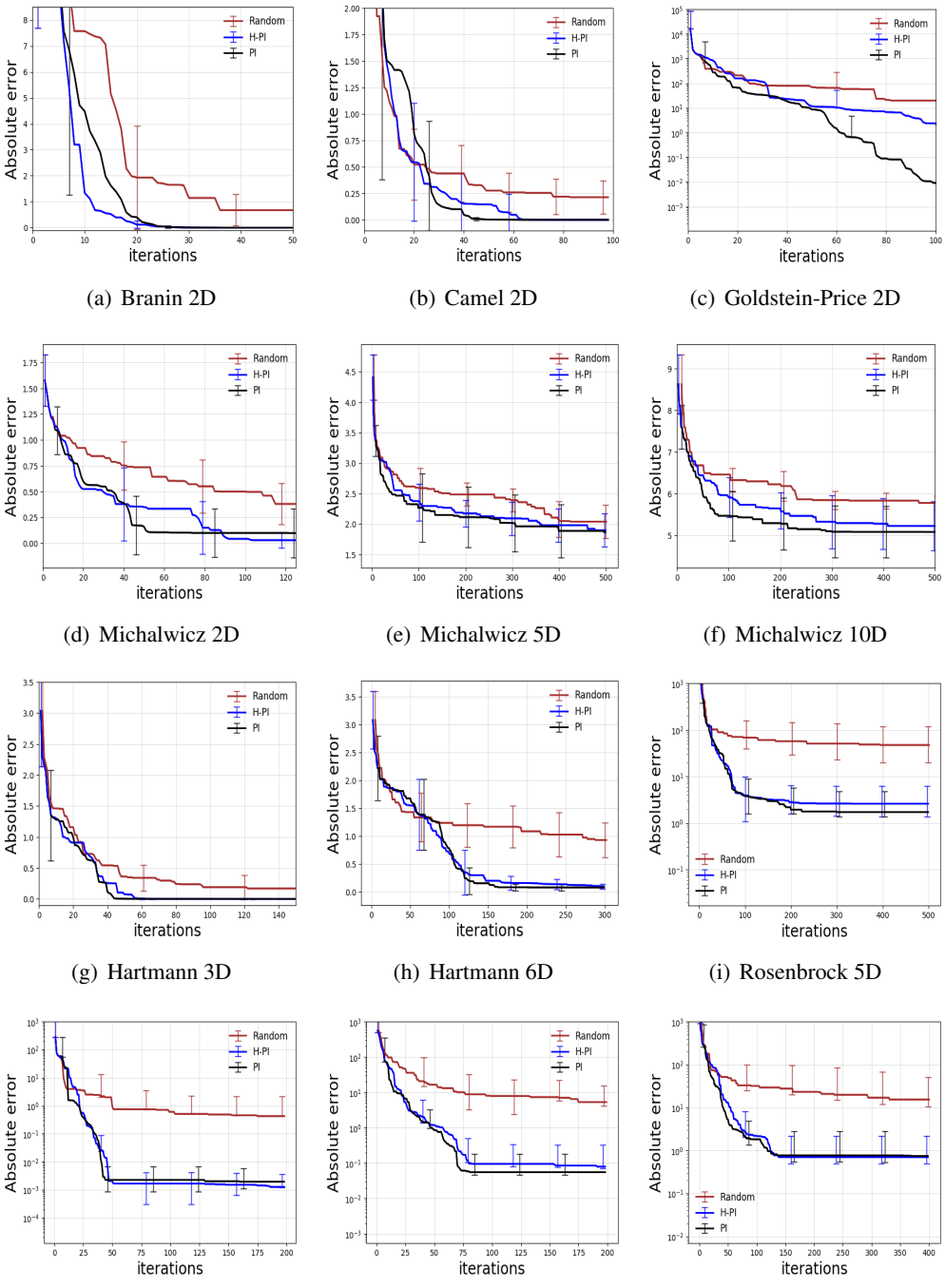
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D

(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D

(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

Figure A.1: Comparing conventional BO and HBO and random exploration for EI on the test functions.



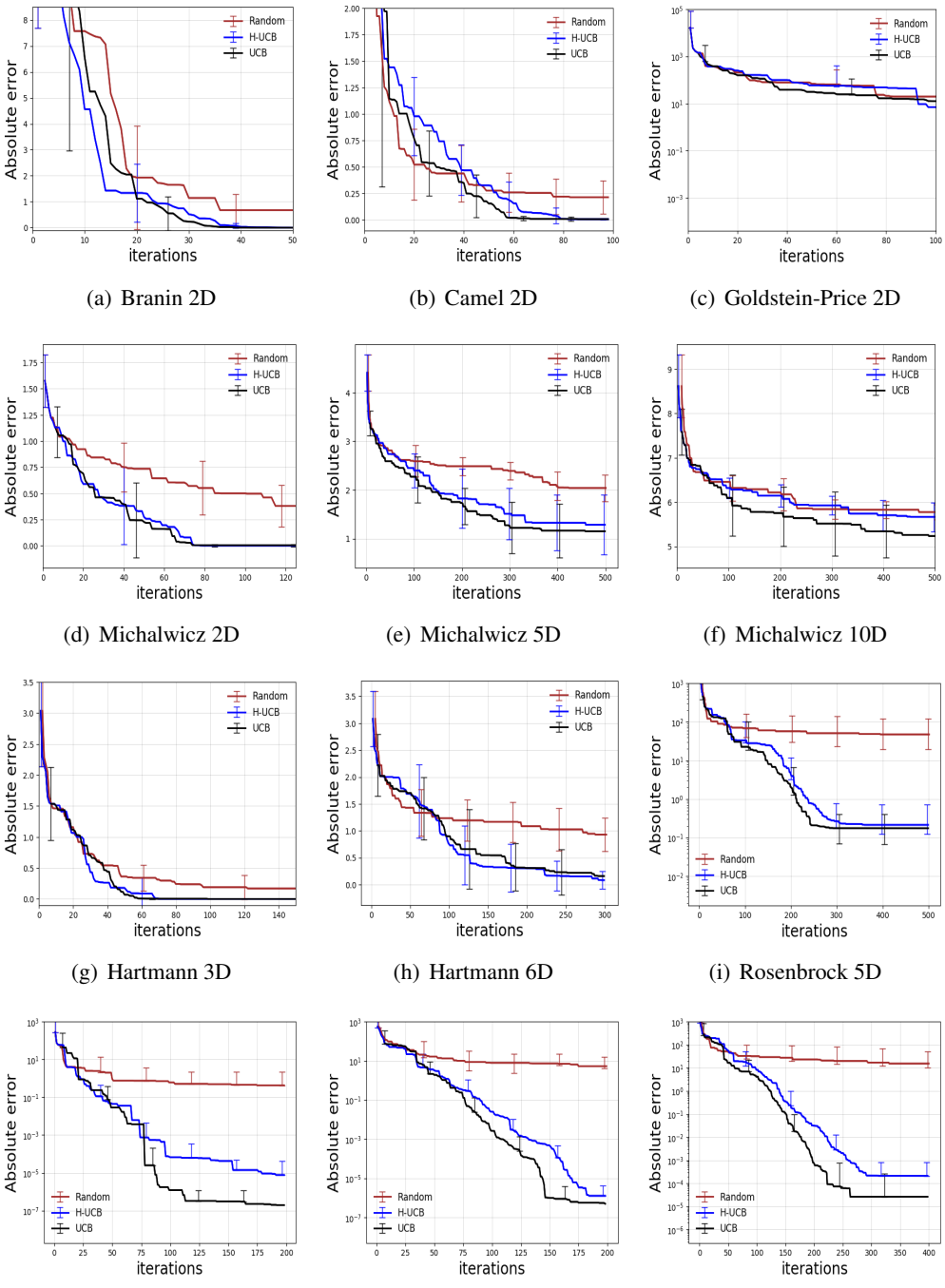
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D

(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D

(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

Figure A.2: Comparing conventional BO and HBO and random exploration for PI on the test functions.



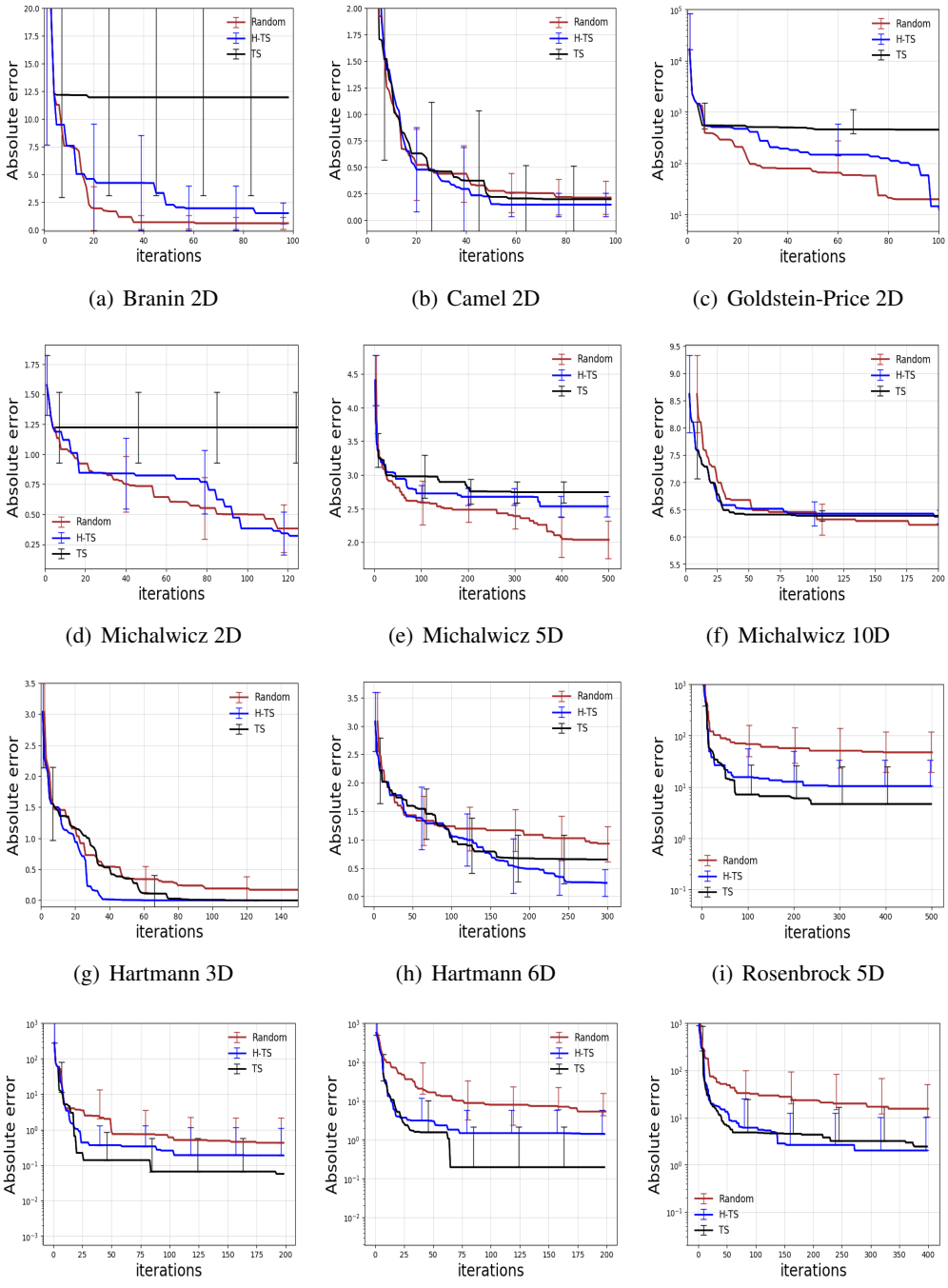
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D

(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D

(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

Figure A.3: Comparing conventional BO and HBO and random exploration for UCB on the test functions.



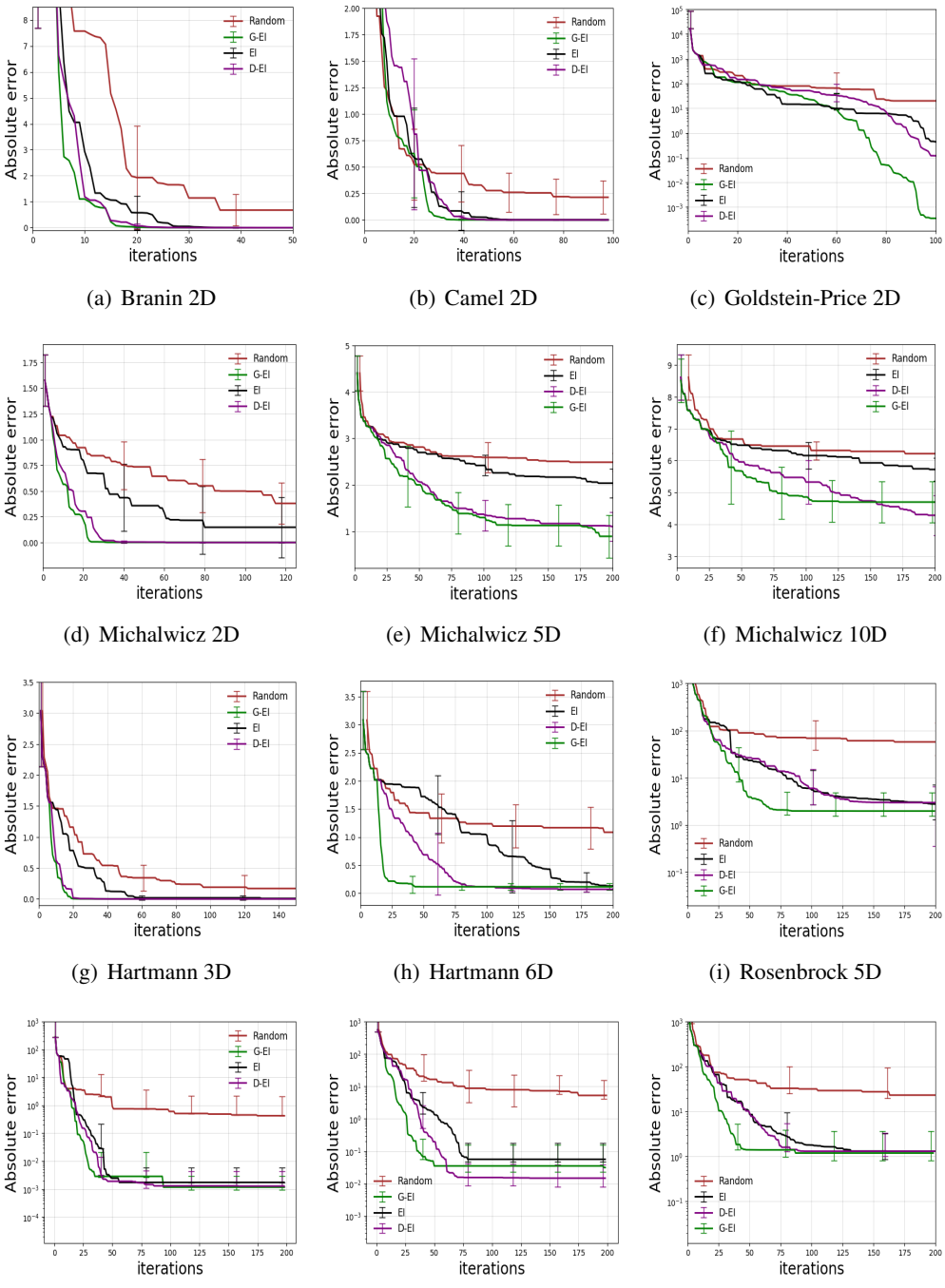
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D

(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D

(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

Figure A.4: Comparing conventional BO and HBO and random exploration for TS on the test functions.



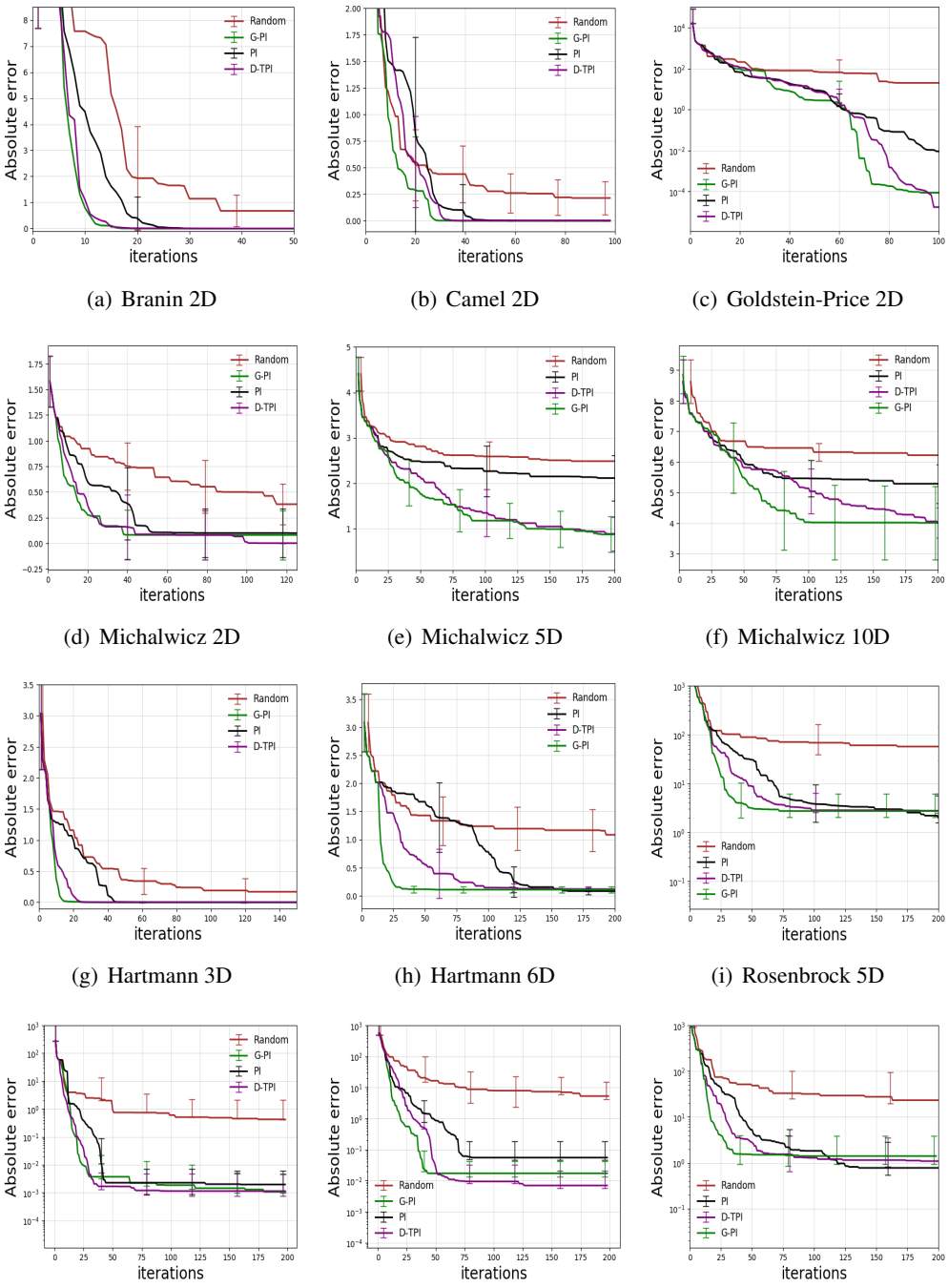
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D

(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D

(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

Figure A.5: Comparing conventional BO and FOBO and random exploration for EI on the test functions.



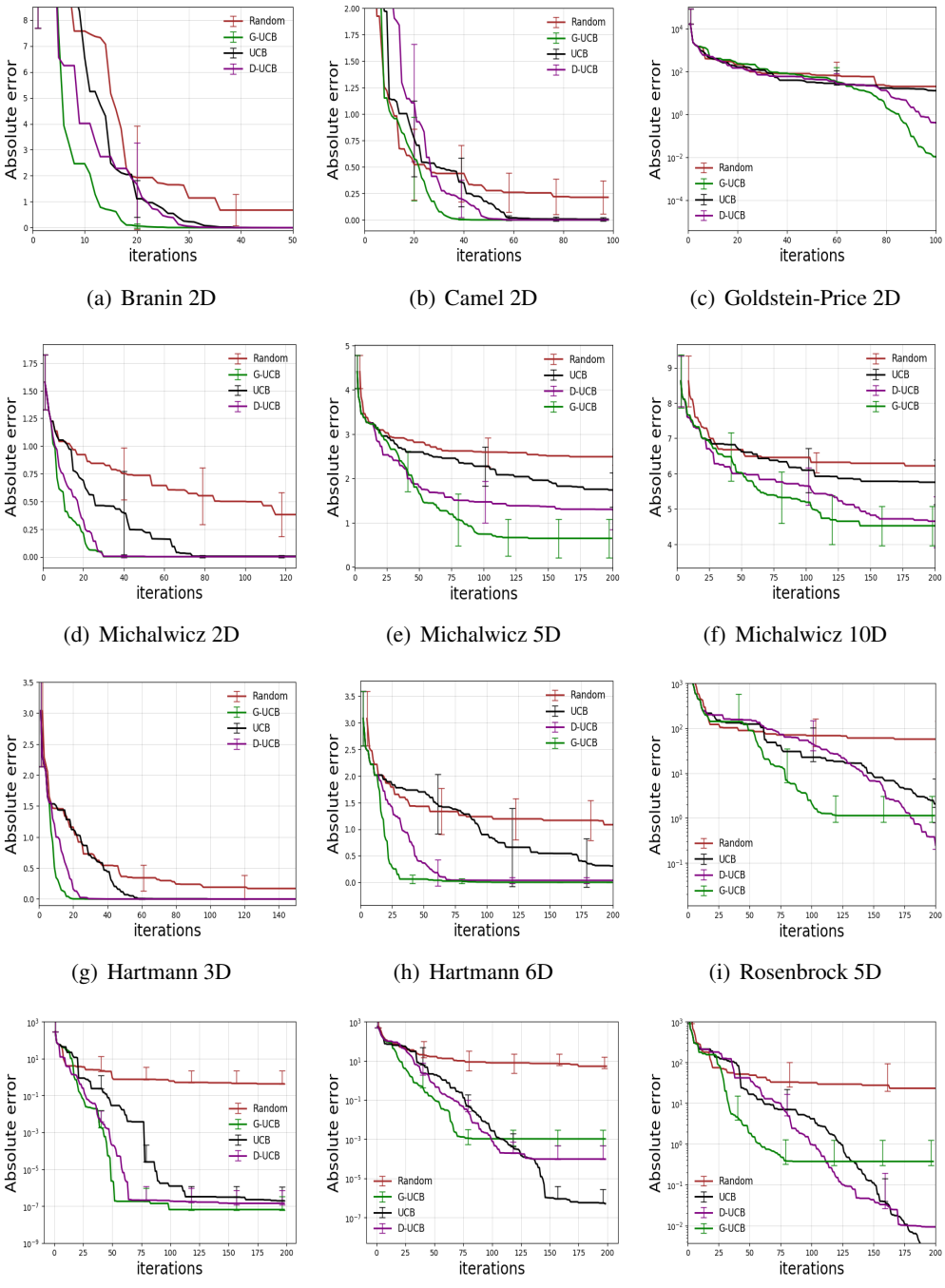
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D

(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D

(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

Figure A.6: Comparing conventional BO and FOBO and random exploration for PI on the test functions.



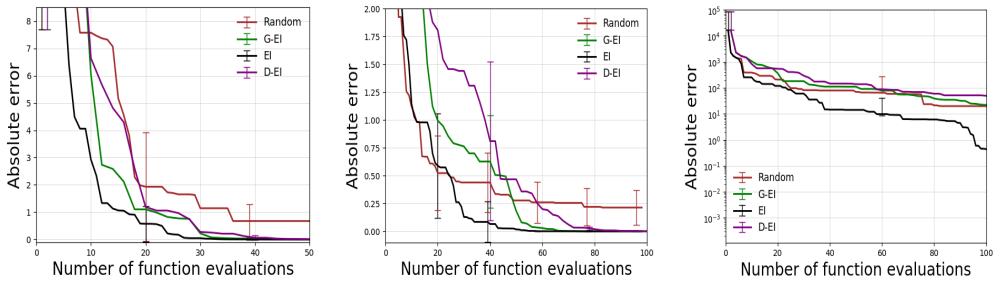
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D

(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

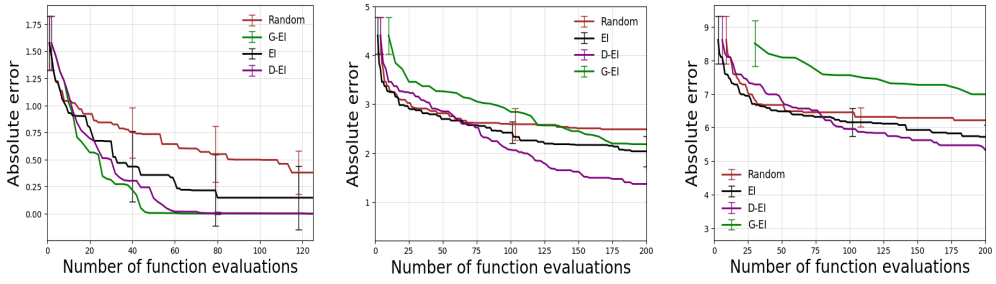
(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D

(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

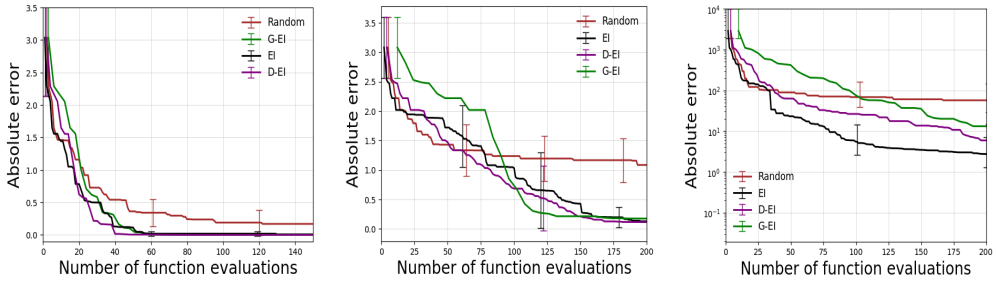
Figure A.7: Comparing conventional BO and FOBO and random exploration for UCB on the test functions.



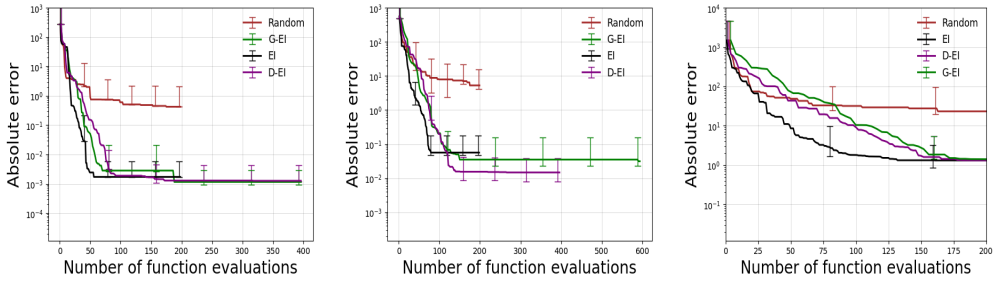
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D



(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

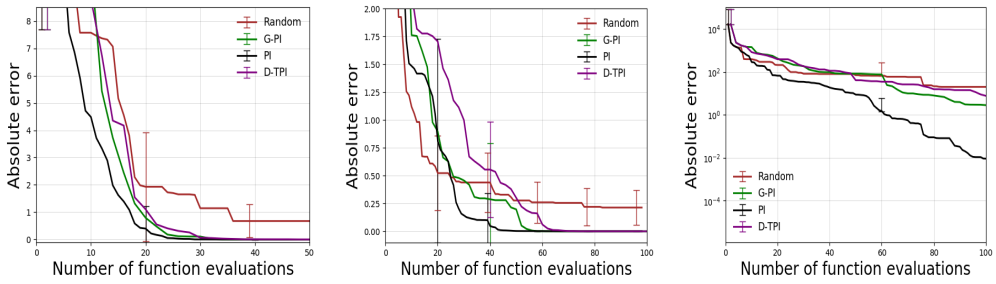


(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D

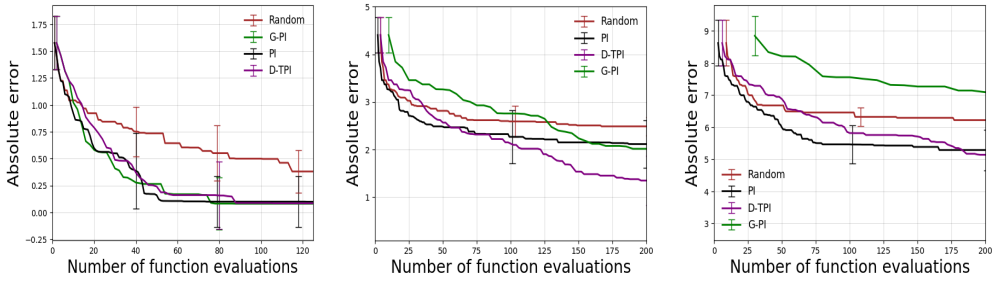


(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

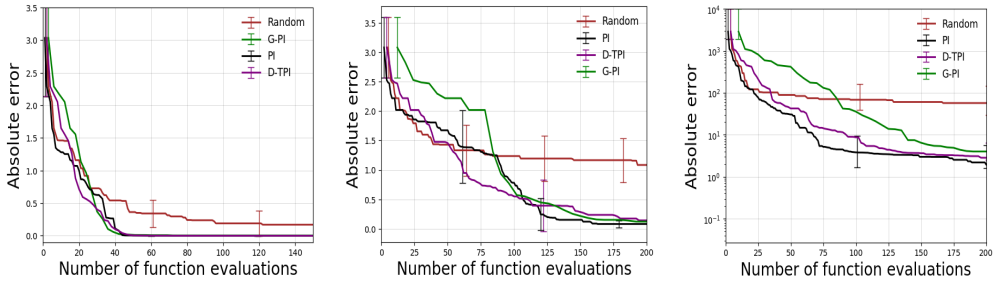
Figure A.8: Comparing conventional BO and FOBO and random exploration for EI on the test functions in terms of number function evaluations.



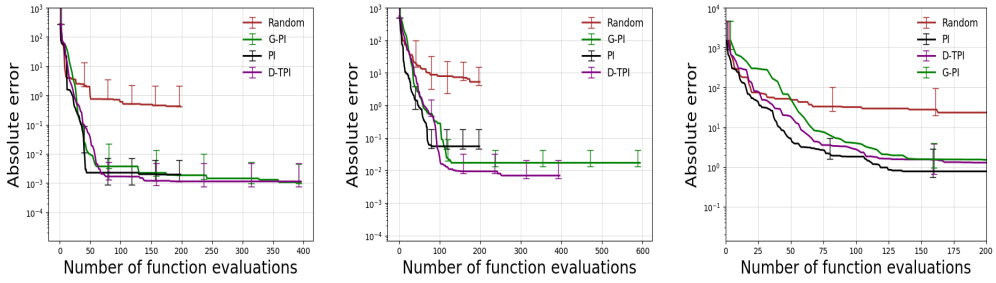
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D



(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

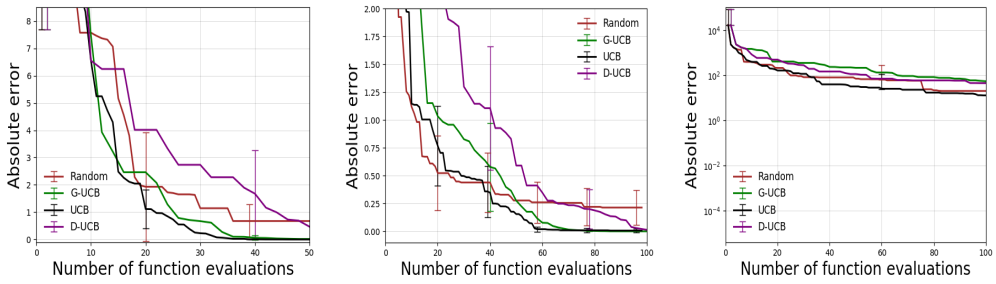


(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D

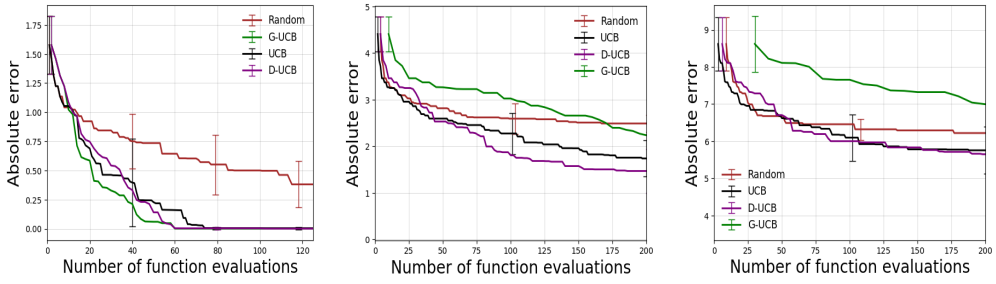


(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

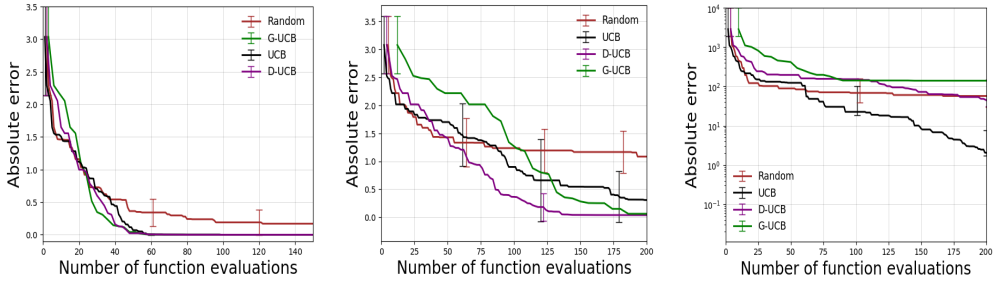
Figure A.9: Comparing conventional BO and FOBO and random exploration for PI on the test functions in terms of number function evaluations.



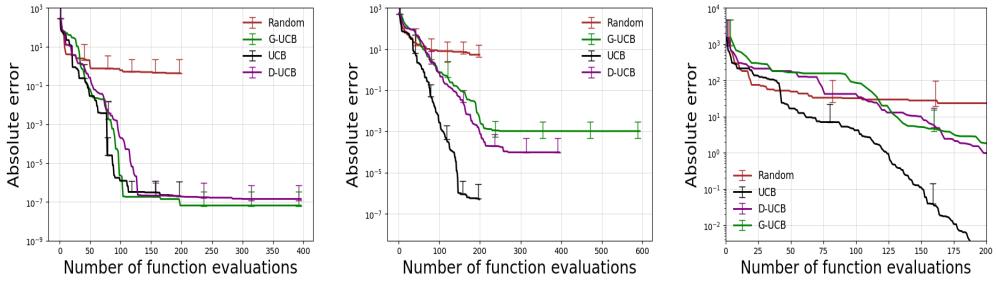
(a) Branin 2D (b) Camel 2D (c) Goldstein-Price 2D



(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D



(g) Hartmann 3D (h) Hartmann 6D (i) Rosenbrock 5D



(j) Rosenbrock 2D (k) Rosenbrock 3D (l) Rosenbrock 4D

Figure A.10: Comparing conventional BO and FOBO and random exploration for UCB on the test functions in terms of number function evaluations.

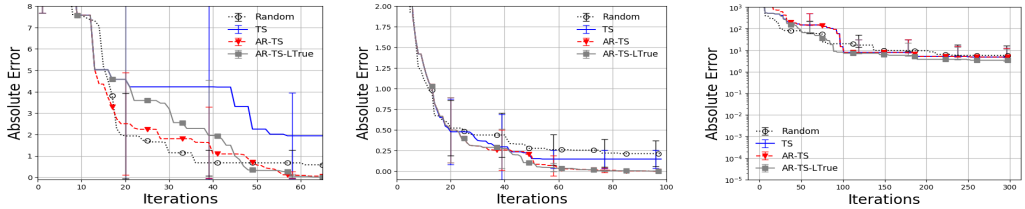
Appendix B

Chapter 5 Supplementary Material

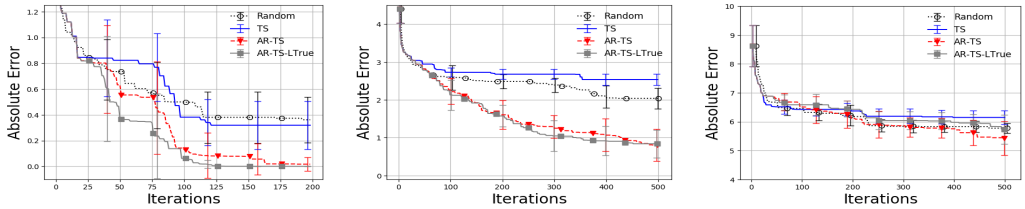
B.1 Additional Experimental Results

Below we show the results for all the datasets as follows:

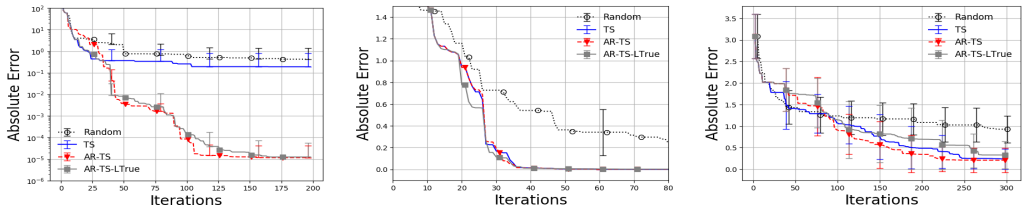
- Figure B.1 shows the performance of Random search, BO, and LBO (using both the estimated and *True* L) for the TS acquisition function.
- Figure B.2 shows the performance of Random search, BO, and LBO (using both the estimated and *True* L) for the UCB acquisition function.
- Figure B.3 shows the performance of Random search, BO, and LBO (using both the estimated and *True* L) for the EI acquisition function.
- Figure B.4 shows the performance of Random search, BO, and LBO (using both the estimated and *True* L) for the PI acquisition function.
- Figure B.5 shows the performance of BO and LBO using the estimated L for the all acquisition function.
- Figure B.6 shows the performance of Random search, BO, and LBO (using both the estimated and *True* L) for the UCB acquisition function with misspecified $\beta = 10^{16}$.



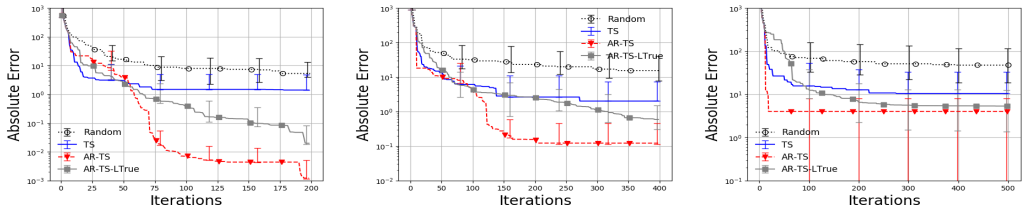
(a) Branin 2D (b) Camel 2D (c) Goldstein Price 2D



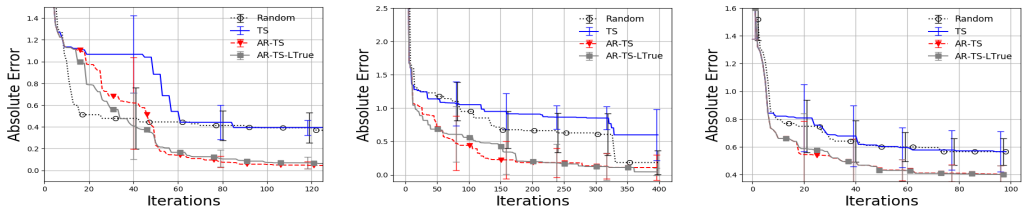
(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D



(g) Rosenbrock 2D (h) Hartmann 3D (i) Hartmann 6D

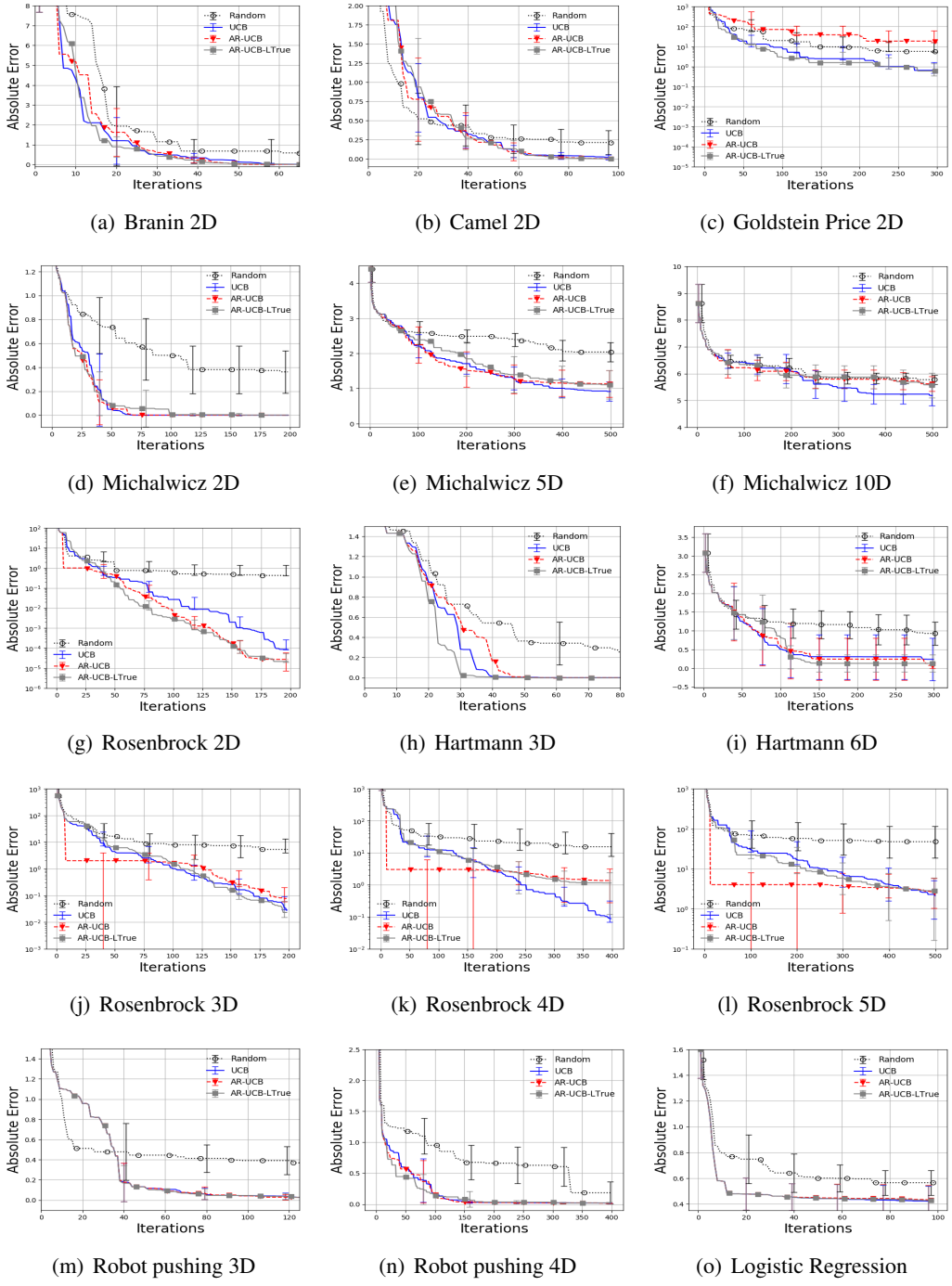


(j) Rosenbrock 3D (k) Rosenbrock 4D (l) Rosenbrock 5D



(m) Robot pushing 3D (n) Robot pushing 4D (o) Logistic Regression

Figure B.1: Comparing the performance of the conventional BO acquisition function, corresponding LBO mixed acquisition function, Lipschitz optimization and random exploration for the TS acquisition functions.



(a) Branin 2D (b) Camel 2D (c) Goldstein Price 2D

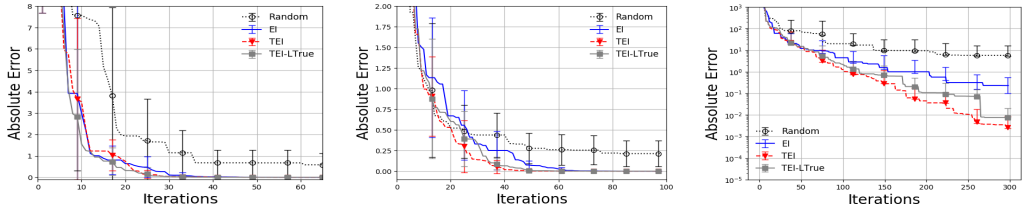
(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

(g) Rosenbrock 2D (h) Hartmann 3D (i) Hartmann 6D

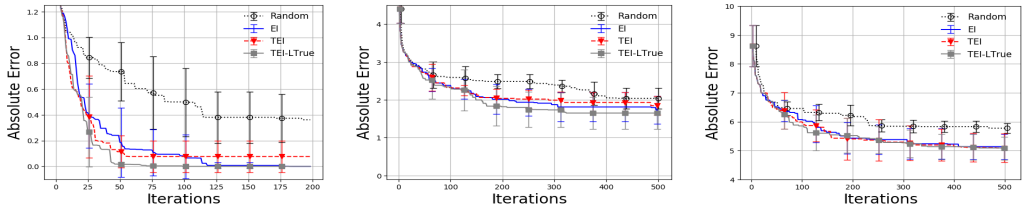
(j) Rosenbrock 3D (k) Rosenbrock 4D (l) Rosenbrock 5D

(m) Robot pushing 3D (n) Robot pushing 4D (o) Logistic Regression

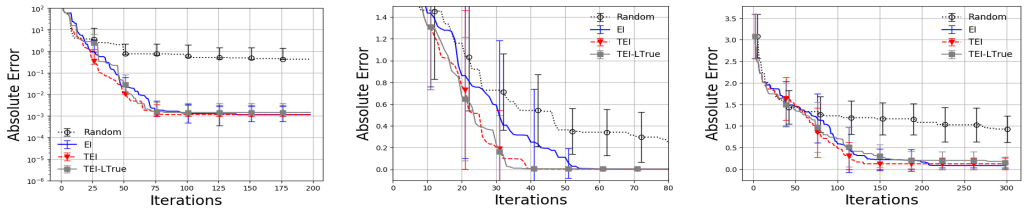
Figure B.2: Comparing the performance of the conventional BO acquisition function, corresponding LBO mixed acquisition function, Lipschitz optimization and random exploration for the UCB acquisition functions.



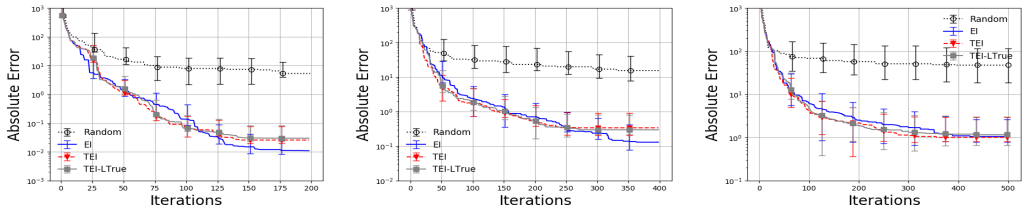
(a) Branin 2D (b) Camel 2D (c) Goldstein Price 2D



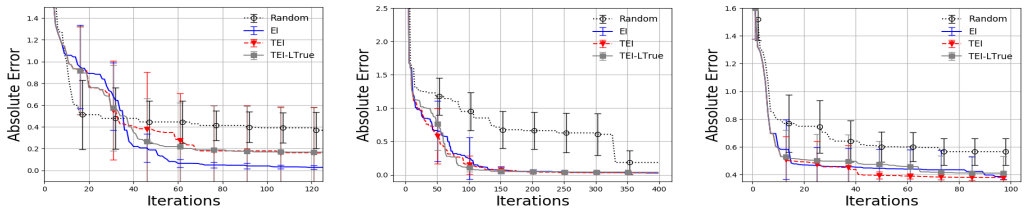
(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D



(g) Rosenbrock 2D (h) Hartmann 3D (i) Hartmann 6D

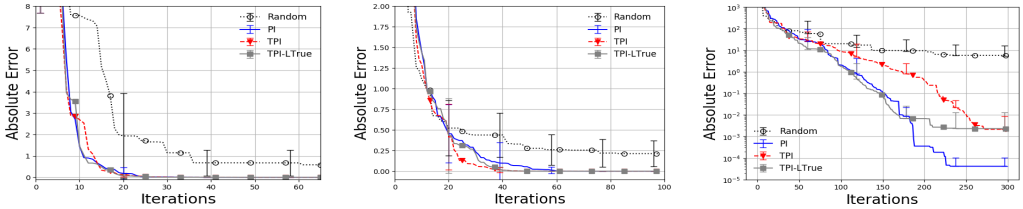


(j) Rosenbrock 3D (k) Rosenbrock 4D (l) Rosenbrock 5D

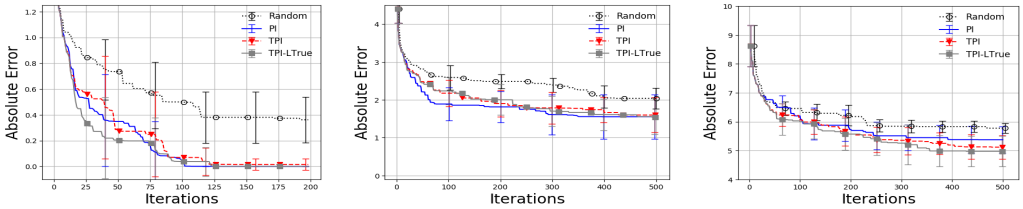


(m) Robot pushing 3D (n) Robot pushing 4D (o) Logistic Regression

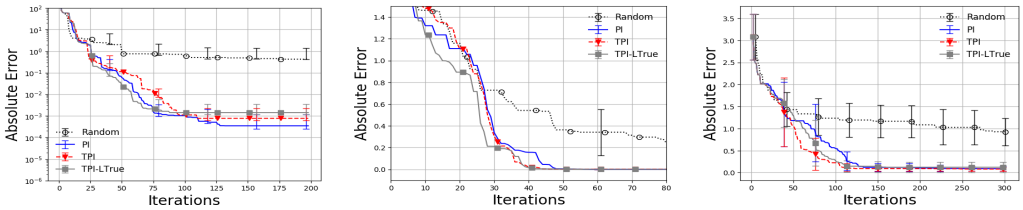
Figure B.3: Comparing the performance of the conventional BO acquisition function, corresponding LBO mixed acquisition function, Lipschitz optimization and random exploration for the EI acquisition functions.



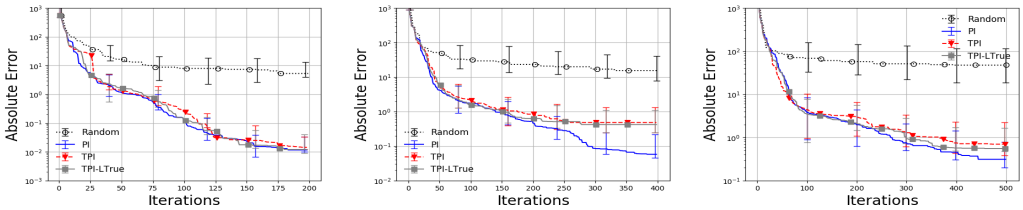
(a) Branin 2D (b) Camel 2D (c) Goldstein Price 2D



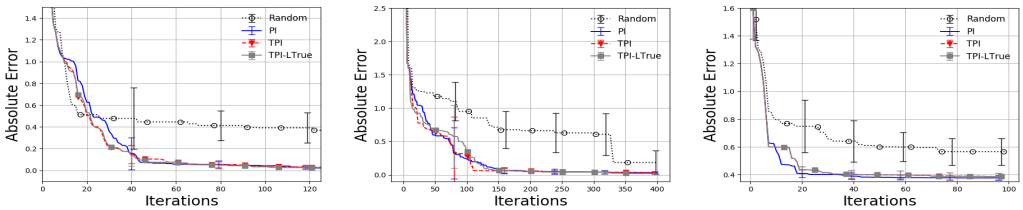
(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D



(g) Rosenbrock 2D (h) Hartmann 3D (i) Hartmann 6D

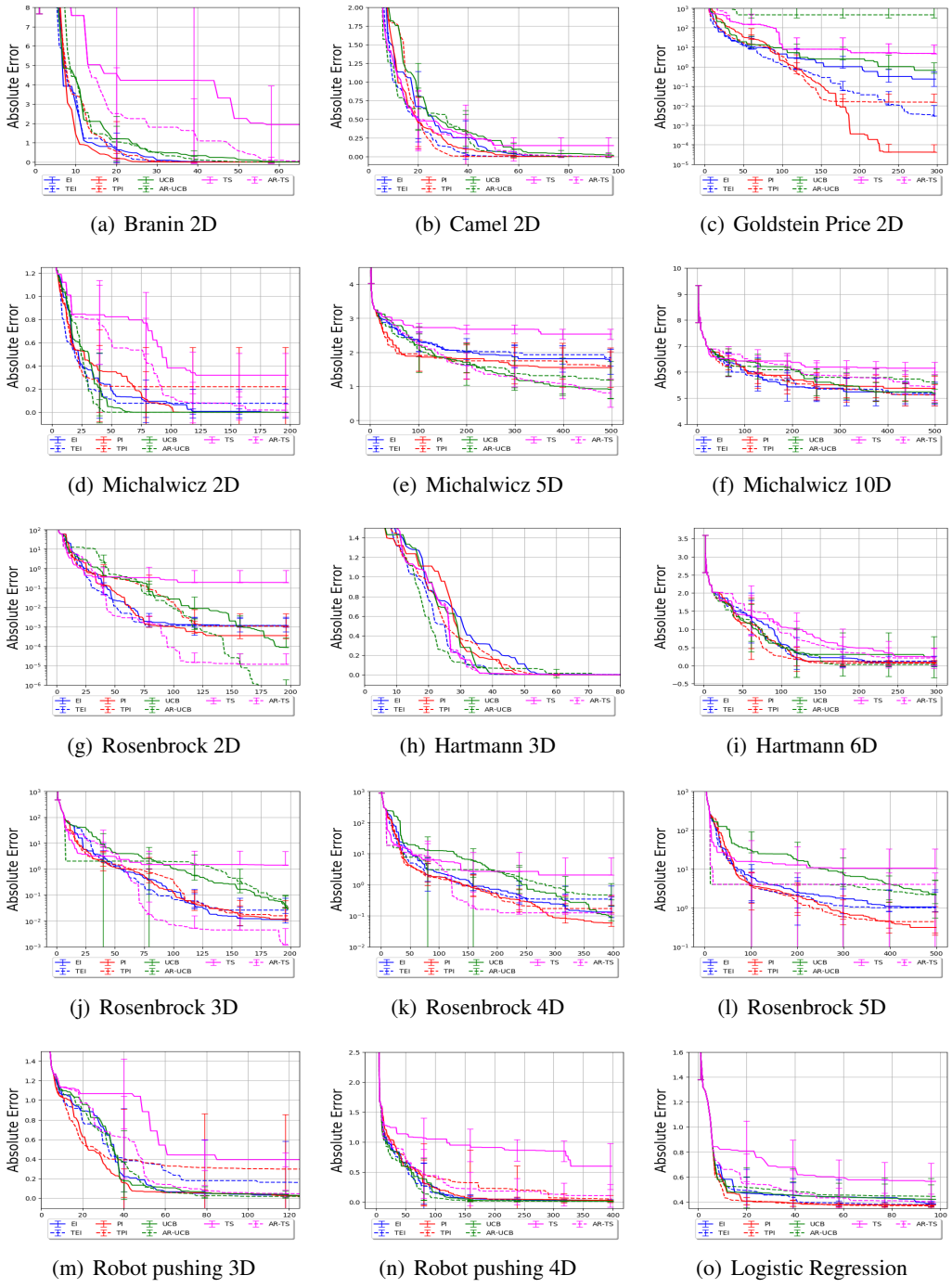


(j) Rosenbrock 3D (k) Rosenbrock 4D (l) Rosenbrock 5D



(m) Robot pushing 3D (n) Robot pushing 4D (o) Logistic Regression

Figure B.4: Comparing the performance of the conventional BO acquisition function, corresponding LBO mixed acquisition function, Lipschitz optimization and random exploration for the PI acquisition functions.



(a) Branin 2D (b) Camel 2D (c) Goldstein Price 2D

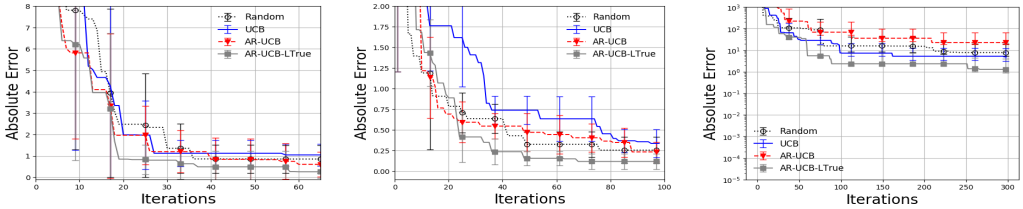
(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D

(g) Rosenbrock 2D (h) Hartmann 3D (i) Hartmann 6D

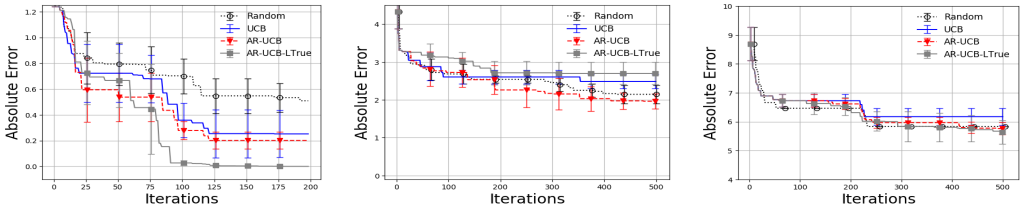
(j) Rosenbrock 3D (k) Rosenbrock 4D (l) Rosenbrock 5D

(m) Robot pushing 3D (n) Robot pushing 4D (o) Logistic Regression

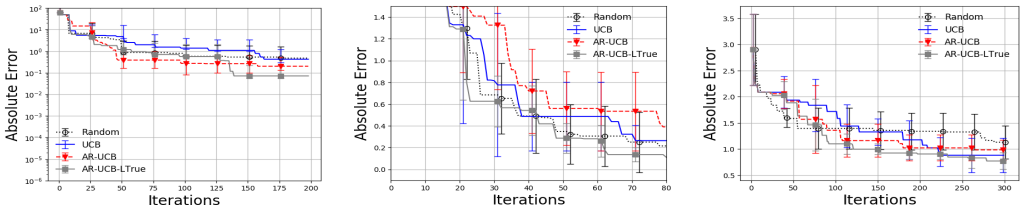
Figure B.5: Comparing the performance across the four BO and the corresponding LBO acquisition functions against Lipschitz optimization and random exploration on all the test functions (Better seen in color).



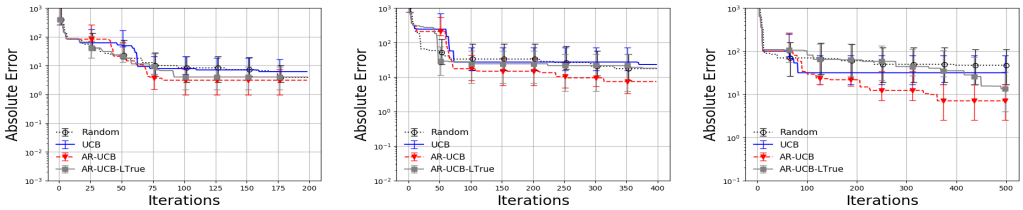
(a) Branin 2D (b) Camel 2D (c) Goldstein Price 2D



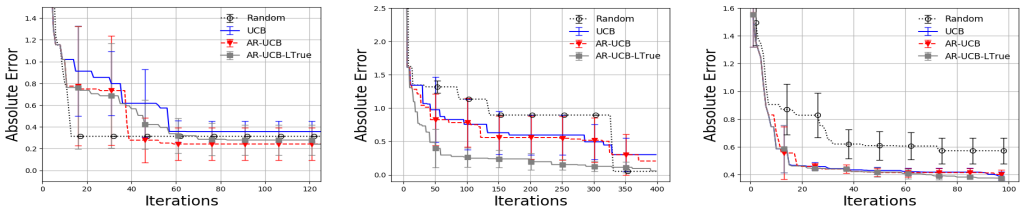
(d) Michalwicz 2D (e) Michalwicz 5D (f) Michalwicz 10D



(g) Rosenbrock 2D (h) Hartmann 3D (i) Hartmann 6D



(j) Rosenbrock 3D (k) Rosenbrock 4D (l) Rosenbrock 5D



(m) Robot pushing 3D (n) Robot pushing 4D (o) Logistic Regression

Figure B.6: Comparing the performance of the conventional BO acquisition function, corresponding LBO mixed acquisition function, Lipschitz optimization and random exploration for the UCB acquisition functions when using very large $\beta = 10^{16}$