Integrated Predictive Entropy Search for Bayesian Optimization

James Ryan Requeima

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of

Master of Philosophy

Darwin College August 2016

Declaration

I, James Ryan Requeima of Darwin College, being a candidate for the Master of Philosophy in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described within is my own, unaided except where specified, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. This thesis contains 10253 words.

[illegible]

James Ryan Requeima

August 2016

Acknowledgements

First I would like to thank my supervisor, Zoubin Gharamini, for the help and encouragement. Thank you for making me feel like my ideas, even the half-baked ones, have value. I would also like to thank my co-supervisor for this thesis project, Matt Hoffman. Working with you has been a lot of fun and I look forward to continuing to do so.

Thanks to the faculty on the MLSALT programme for making it an interesting year - I definitely learned more this year than any other. Thanks to all of the students in the programme - the atmosphere was always one of collaboration and sharing. Thanks to Rich Turner for making the CBL a welcoming place to work on interesting and exciting problem. Thanks to Will, Alex, Ambrish, Wessel and Will for keeping me company during the late night work sessions at the department.

Thanks to all of my friends at Invenia for the help and support and for being an amazing place to work on machine learning applications.

Thanks to David Duvenaud, who got me interested in Machine Learning in the first place and showed me that Cambridge is an amazing place to study it.

Thanks to my parents and family for getting me to where I am today. Lastly, I want to thank my wife, Antonia - I honestly couldn't have done this without you. You have given me so much love and support.

Abstract

Predictive Entropy Search (PES) is an information-theoretic based acquisition function that has been demonstrated to perform well on several applications. PES harnesses our estimate of the uncertainty in our objective to recommend query points that maximize the amount of information gained about the local maximizer. It cannot, however, harness the potential information gained in our objective model hyperparameters for better recommendations. This dissertation introduces a modification to the Predictive Entropy Search acquisition function called Integrated Predictive Entropy Search (IPES) that uses a fully Bayesian treatment of our objective model hyperparameters. The IPES aquisition function is the same as the original PES aquision function except that the hyperparameters have been marginalized out of the predictive distribution and so it is able to recommend points taking into account the uncertainty and reduction in uncertainty in the hyperparameters. It can recommend queries that yield more information about the local maximizer through information gained about hyperparameters values.

Currently, approximations used in the computation of the PES acquisition function require stationary kernels - a condition that greatly restricts the types of kernel functions that can be used in the Gaussian process objective model. More flexible kernel functions, in particular the Spectral Mixture kernel and kernels using input space warpings, are examined. Alternative approximations for use in PES and IPES are introduced, in particular a procedure for obtaining analytic approximations to Gaussian process samples. These alternate approximations eliminate the stationarity requirement in PES and extend the applicability of PES and IPES to a larger class of problems.

Performance of IPES and other methods including PES are examined both with and without these alternative approximations.

Contents

List of Figures

Chapter 1

Introduction

1.1 Optimization

Optimization problems are widespread in science, engineering, economics and finance. For example, regional electricity grid system operators (ISOs) optimise the production of electricity (solar, wind and fossil fuels) and its allocation to produce and provide energy as efficiently and cheaply as possible; The Harvard Clean Energy Project identifies molecules for organic photovoltaics that have optimal power conversion efficiency; software controlling robots can be optimised so that the agent fulfils user specified goals in the most energy efficient, timely or safest way. Generally speaking, optimization problems come in two broad categories [17]. First, there are optimization problems where it is very cheap to query the function to be optimized. Often function evaluation takes place on a computer and its derivative observations are available. This type of optimization is not the main application of the types of methods that will be explored in this thesis.

The second type of optimization, and the one that we will be primarily concerned with, is the so-called *black-box* optimization. In these the types of problems we seek to find the global maximizer of a nonlinear and generally nonconvex function f whose derivatives are unavailable. In applications resulting in such objectives, data collection is often expensive, difficult and potentially dangerous. For this reason it is crucial to gather data in an efficient way and to extract the maximum possible value from it. Active learning - using an algorithm to recommend at which points to query the function - will help us to optimize our objective more quickly and with fewer data-points. This thesis will examine the application of a Bayesian approach to optimization called *Bayesian optimization* as well as an information theoretic technique for active learning recommendations for data collection.

Bayesian optimization has been successfully applied to optimization problems in science and engineering. For example; to adjust the parameters of a robot's controller [6]; to find the optimal pump-and-treat wells for water decontamination [15]; to automatically find optimal parameters for statistical machine translation systems [3] and other machine learning systems [42]. Many organizations use the OBM ILOG CPLEX[1] mixed integer programming solver which has 76 free parameters, often too many to tune by hand [39]. Bayesian optimization has also emerged as a powerful tool for automating design choices [39].

1.2 Contribution of Thesis

Bayesian optimization is a powerful optimization technique and methods currently implemented in the python Bayesian optimization library `Pybo`, in particular the method called Predictive Entropy Search (PES), have been shown to perform well on several applications [18]. However, the current implentation restricts the class of problems on which these methods can be applied.

In this this thesis, I investigate an modification to PES called Integrated Predictive Entropy Search (IPES) that has a better theoretical foundation than predictive entropy search. I examine its performance when compared with other popular Bayesian optimization and, in particular, Predictive Entropy Search.

I also investigate methods to extend the applicability of IPES as well as PES to more realistic applications In particular input space warping, the application of more expressive kernel functions (in particular the Spectral Mixture Kernel) and a GP sampling approach called bootstrap sampling.

These methods were implemented in and the `pybo`[2] package and will become freely available for public use.

[1]https://www.ibm.com/software/commerce/optimization/cplex-optimizer/
[2]Hoffman and Shahriari, https://github.com/mwhoffman/pybo

Chapter 2

Bayesian Optimization

2.1 The Bayesian Optimization Algorithm

The goal of Bayesian optimization is to solve optimization problems of the form

$$\mathbf{x}_\star = \arg\max_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$$

where f is often a nonlinear and non-convex function over a domain $\mathcal{X}$. In global optimization, $\mathcal{X}$ is often a bounded subset of $\mathbb{R}^d$. We also assume that f is a *black-box* function; f has no simple closed form but can be evaluated at any query point in the domain and its derivatives are usually unavailable. We also assume that we are only able to observe our function f through unbiased noisy observations y, meaning $\mathbb{E}[y|f] = f(x)$.

To address the problems associated with black-box optimization, Bayesian optimization models the unknown function and its behaviour in order to minimize the number of evaluations required to find the global optima. Crucial to Bayesian optimization is the use of a model that maintains a measure of uncertainty in its function values. This is important because often the observations used to train our model is corrupted by noise, and we harness this measure of uncertainty for form recommendations for data collection.

Bayesian optimization is a sequential model based approach to problem solving [39]. We incorporate our prior beliefs about our objective function in our model and iteratively update these believes in the presences of newly queried datapoints. The generic Bayesian optimization algorithm is as follows [18]:

Algorithm 1 Bayesian optimization

```
Input: a black-box function $f$
1: for $n = 1, \dots, N$ do
2:     select $\mathbf{x}_n = \arg\max_{\mathbf{x} \in \mathcal{X}} \alpha_{n-1}(\mathbf{x})$
3:     query $f$ at $\mathbf{x}_n$ to obtain $y_n$
4:     augment data $\mathcal{D}_n = \mathcal{D}_{n-1} \cup \{(\mathbf{x}_n, y_n)\}$
return $\widetilde{\mathbf{x}}_N = \arg\max_{\mathbf{x} \in \mathcal{X}} \mu_N(\mathbf{x})$
```

Two main decisions have to be made in order to use Bayesian optimization: we need to select the model that we will use to model our objective, and we need to select an acquisition function that we will use to recommend data points to sample.

2.2 Choice of Model

It is necessary for our choice of objective function model to have two main attributes: it needs to be able to incorporate our prior beliefs about our objective and for these to be updatable, and it needs to maintain a measure of uncertainty over its predictions. What sets Bayesion optimization apart from most other optimization procedures is that it uses a probabilistic model for our objective and exploits this model to produce recommendations for where to next query the objective.

Many candidates for objective models have been proposed. Some of the earliest work in Bayesian optimization used the Wiener process [26] [**?**]. Shahriari et al. [39] outline some of the parametric models that have been used. Snoek et al. [42] have proposed the use of deep neural networks. Hutter et al. proposed a method using random forests in sequential model-based algorithm configuration (SMAC) [23].

The most common class of nonparametric models for Bayesian optimization, and the one that we will be focusing on in this thesis, are the Gaussian process models.

2.2.1 The Gaussian Process

A Gaussian process (GP) is an elegant way of defining a distribution over functions. Let $\mathbf{X}$ be a collection of n points in $\mathcal{X}$, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, stacked into a matrix. A GP is any distribution over functions such that the function values $f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)$ are jointly Gaussian distributed [37] [9]. Before conditioning on data, a Gaussian process, denoted $\mathcal{GP}(\mu, k)$, is completely specified by its prior mean function $\mu : \mathcal{X} \to \mathbb{R}$ and covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, also referred to as the kernel function. These functions specify

$$\mathbb{E}[f(\mathbf{x})] = \mu(\mathbf{x}) \quad \text{and} \tag{2.1}$$

$$\operatorname{cov}[f(\mathbf{x}), f(\mathbf{x}')] = k(f(\mathbf{x}), f(\mathbf{x}')) \tag{2.2}$$

If we further assume that the observations $y_1, y_2, \ldots, y_n$ associated with $\mathbf{X}$ are independently and normally distributed given $f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)$ we have the following generative model [39]:

$$\mathbf{f} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}) \tag{2.3}$$

$$\mathbf{y} \sim \mathcal{N}(\mathbf{f}, \sigma^2\mathbf{I}) \tag{2.4}$$

where $\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)]^\top$, $\mathbf{y} = [y_1, y_2, \ldots, y_n]^\top$ are column vectors, the mean vector $\mathbf{m}$ is defined by $m_i = \mu(\mathbf{x}_i)$ and the covariance matrix, or kernel matrix, $\mathbf{K}$ by $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. Given the set of observations $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}$ and an arbitrary point we can compute the Gaussian process posterior $\mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$:

$$\mu_{\text{post}}(\mathbf{x}) = \mu(\mathbf{x}) + k(\mathbf{x}, \mathbf{X})^\top(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(\mathbf{y} - \mathbf{m}) \quad \text{and} \tag{2.5}$$

$$k_{\text{post}}(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{X})^\top[\mathbf{K} + \sigma^2\mathbf{I}]^{-1}k(\mathbf{X}, \mathbf{x}') \tag{2.6}$$

where $k(\mathbf{x}, \mathbf{X}) = [k(\mathbf{x}, \mathbf{x}_1), k(\mathbf{x}, \mathbf{x}_2), \ldots, k(\mathbf{x}, \mathbf{x}_n)]^\top$. And so the predictive distribution of an observation $y_\star$ at a test point $\mathbf{x}_\star$ has the form

$$p(y_\star|\mathbf{x}_\star, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mu(\mathbf{x}) + k(\mathbf{x}, \mathbf{X})^\top(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(\mathbf{y} - \mathbf{m}), \tag{2.7}$$

$$k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{X})^\top[\mathbf{K} + \sigma^2\mathbf{I}]^{-1}k(\mathbf{X}, \mathbf{x}')). \tag{2.8}$$

Here we can see one of the main limitations of the Gaussian process model - inference requires computing matrix inverses, and so is $\mathcal{O}(n^3)$ in the number of observations. As a nonparametric model, it is in fact parametrized by the our dataset. The amount of information that our model can capture about the data can grow as we gather more data but so do the computation costs [13]. However, for problems where Bayesian optimization is appropriate, we are often data scarce.

Mean Functions

The prior mean function is a way of incorporating prior and expert knowledge of the objective function in a principled way. In practice, however, this function is often set to a constant $\mu(\mathbf{x}) = \mu_0$. In fact, it is often set to zero since the kernel function can account for uncertainty about the mean [9]. In what follows, unless otherwise specified, it is assumed that a constant prior mean function is used.

Covariance Functions (Kernels)

The kernel function determines the kind of structure that can be captured by a GP model and how it generalizes. The kernel must be a positive definite function of two inputs $k(\mathbf{x}, \mathbf{x}')$. Kernels are often used to specify the similarity between two objects and in the context of Gaussian processes specify the covariance between function values.

The main kernel function that will be used in our experimentation will be the squared exponential kernel with automatic relevance determination determination (SEARD). The SEARD kernel [12] is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{2{l_d}^2}\right) \tag{2.9}$$

where D is the number of dimensions of the data. The feature that distinguishes the SEARD kernel from the standard squared exponential kernel is the individual length-scales in each dimension. In this way, through hyperparameter learning, our kernel can scale the data independently in each dimension. The lengthscale hyperparameter l_d determines the length of the wiggles in dimension d and generally determining the distance away from datapoints that our GP regressor can extrapolate. The output variance hyperparamter determines the average the average distance of the function away from the mean [10] This parameter corresponds to the normalizing constant in the kernel spectral density, discussed in chapter 3.

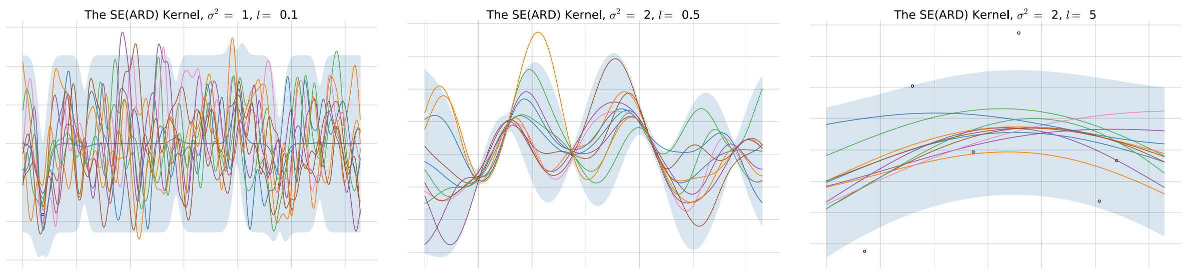


Figure 2.1: Samples from a GP with a SEARD kernel using various hyperparameter settings.

2.3 Acquisition Functions

Most acquisition functions are designed to recommend candidates for the objective maximizer. In this case, we would want $\alpha_n(\mathbf{x})$ to be high in areas where the maximum is likely. This would suggest that $\alpha_n(\mathbf{x})$ should closely follow our posterior mean, a behaviour known as exploitation. However, we would also like recommendations that explore our search space and not just the maximum of the posterior mean at step N of our algorithm in order to ensure that $\tilde{\mathbf{x}}_N$ yields the global maximum. For this, we need to take into account the uncertainty in our model predictions and utilize this uncertainty in our recommendations. A good balance of exploitation and exploration is required for a well performing acquisition function and the right balance is objective dependent.

The optimal policy would be to plan for multiple samples, especially in cases where the number of samples in the budget for our problem is known. At each iteration, we would want to select the best point given that we will be sampling N points according to a given strategy [36] [16]. This is often too computationally expensive in practice and so most current acquisition functions are myopic heuristics, and attempt to approximate the total value of querying any given point.

The acquisition functions used in Bayesian optimization are often multi-modal and difficult to optimize with respect to querying efficiency when compared to the black-box function [39]. Since the acquisition function, α_n, is to be maximized at every iteration, it is necessary for it to be cheap to evaluate or approximate, especially when compared to the evaluation of our black-box objective. Many have analytic forms or can be approximated analytically and are easier to optimize using numerical optimization techniques [21] such as LBFGS [35] [28] and DIRECT [24].

We would like our data collected to be as valuable as possible to our task of maximizing our objective function. Shahriari et al. [39] discuss framing this value in terms of a utility function mapping points $\mathbf{x} \in \mathcal{X}$, along with its function value $v = f(x)$ and model hyperparameters θ to the quality of the query,

$$U : \mathbb{R}^d \times \mathbb{R} \times \Theta \to \mathbb{R}, \tag{2.10}$$

where U is larger at points that are more valuable for our optimization task.

Using whichever utility function we define and the data observed so far, we can define an acquisition function by marginalizing out the hyperparameters θ and function values v. We mostly ignore the dependence on θ but will return the discussion of integrating over the hyperparameters in Chapter 5.

$$\alpha(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_\theta \mathbb{E}_v [U(\mathbf{x}, v, \theta)] \tag{2.11}$$

Note that this function is often called the expected utility in the experimental design and decision theory literature.

2.3.1 Improvement-based Acquisition Functions

Probability of Improvement

An early proposal for an acquisition function was the probability of improvement (PI) acquisition function [26]. This acquisition function models the probability that a particular data point will yield a larger objective value over the best observation so far, defined to be the observation with the highest posterior mean. Denote the previously sampled point with the highest posterior mean by

$$\mathbf{x}_n^+ = \operatorname*{argmax}_{\mathbf{x}_i \in \mathcal{D}_n} \mu_n(\mathbf{x}_i). \tag{2.12}$$

We will call this point the current incumbent. Kushner originally proposed selecting the point with the largest possibility of improvement over this incumbent [26] but this approach is biased toward exploitation, exhibits very myopic behaviour and tends to not explore the domain very much. To remedy this, Kushner suggests incorporating a tradeoff parameter $\xi \geq 0$ to consider the probability of improving over a target $\tau = \mu(\mathbf{x}_n^+) + \xi$. Formally, we can define $\alpha(\mathbf{x}; \mathcal{D}_n) = P(v > \tau)$. To connect this idea with our earlier discussion of utility functions, this definition is equivalent to that of setting the utility to the improvement indicator utility $U(\mathbf{x}, v, \theta) = \mathbb{I}[v > \tau]$. Since the predictive posterior is Gaussian, we can compute the expected utility analytically:

$$\alpha_{\mathrm{PI}}(\mathbf{x}; \mathcal{D}_n) := \mathbb{E}_v \left[\mathbb{I}[v > \tau]\right] \tag{2.13}$$

$$= \int_{v>\tau} \mathcal{N}(v; \mu_n(\mathbf{x}), \sigma^2(\mathbf{x})) dv \tag{2.14}$$

$$= \Phi\left(\frac{\mu_n(\mathbf{x}) - \tau}{\sigma_n(\mathbf{x})}\right) \tag{2.15}$$

where Φ is the standard normal cumulative distribution function. Notice that since Φ is a monotonic increasing function, we can equivalently maximize

$$\alpha(\mathbf{x}; \mathcal{D}_n) = \frac{\mu_n(\mathbf{x}) - \tau}{\sigma_n(\mathbf{x})} \tag{2.16}$$

Kushner recommends using a decreasing schedule to set ξ [21], however, empirical results on a suite of test functions showed that this approach did not yield improvements over using a constant ξ value [29].

Expected Improvement

The PI considers only the probability of improvement but weights all potential improvements equally. The previous utility gave a value of 1 to any improvement over the target and 0 to no improvement. The Expected Improvement acquisition function, a concept proposed by Mockus [32], defines the utility as the amount of improvement over our target. This utility is called the improvement function:

$$U(\mathbf{x}, v, \theta) = (v - \tau)\mathbb{I}[v - \tau]. \tag{2.17}$$

Again, since our predictive posterior is Gaussian, we can compute the expectation of this utility analytically using integration by parts [25] [5]:

$$\alpha_{\text{EI}} := \mathbb{E}_v[(v-\tau)\mathbb{I}(v-\tau)] \tag{2.18}$$

$$= (\mu_n(\mathbf{x}) - \tau)\Phi\left(\frac{\mu_n(\mathbf{x}) - \tau}{\sigma_n(\mathbf{x})}\right) + \sigma_n(\mathbf{x})\phi\left(\frac{\mu_n(\mathbf{x}) - \tau}{\sigma_n(\mathbf{x})}\right) \tag{2.19}$$

where ϕ is the standard normal probability density function.

Using the incumbent point as the target ($\xi = 0$) works reasonably well for EI [40] [39].

2.3.2 Optimistic Acquisition Functions

Upper Confidence Bounds

Using upper confidence bounds has long been a popular was of balancing exploration and exploitation in the context of the multi-arm bandit problem [27] [39]. The idea moved over to the Bayesian optimization literature with the introduction of an algorithm called Sequential Design for Optimization [8] and a similar method called UCB1 was proposed by Auer et. al [2] [21]. Both ideas correspond to maximizing the sum of the posterior mean and weighted variance, $\mu_n(\mathbf{x}) + \beta\sigma_n(\mathbf{x})$. Shahriari et al. refer to this class of acquisition function as optimistic because they are "optimistic in the face of uncertainty" [39], indeed, maximizing a fixed probability best case scenario.

Using the acquisition function defined by

$$\alpha_{\text{UCB}}(\mathbf{x}; \mathcal{D}_n) := \mu_n(\mathbf{x}) + \beta_n\sigma_n(\mathbf{x}) \tag{2.20}$$

Srinivas et al present theoretically motivated guidelines, with respect to minimizing regret, for setting and scheduling the parameter β_n [44]. It can be shown that, for most kernels, this method has cumulative regret bounded by $\mathcal{O}(N\gamma_N)$ where γ_N is sublinear in N and guaranteed convergence [21].

2.3.3 Distribution of the Maximizer Based Procedures

A class of acquisition procedures considers the posterior distribution over the unknown latent variable $\mathbf{x}_\star$ denoted $p(\mathbf{x}_\star|\mathcal{D}_n)$. The two main approaches explored so far is inducing exploration by sampling from this distribution and examining its entropy.

Thompson Sampling

Thompson sampling (TS) was introduced in 1933 [32] [38] and re-emerged in the multi-armed bandit literature [7].

The TS procedure samples the next query point from the posterior over the latent maximizer $\mathbf{x}_\star$:

$$\mathbf{x}_{n+1} \sim p(\mathbf{x}_\star | \mathcal{D}_n). \tag{2.21}$$

This procedure is naturally applied to the bandit setting - TS samples a reward function from our posterior distribution of reward functions and selects the arm that maximizes this function. However, in order to apply this procedure in black-box optimization of functions over continuous spaces, a function must be sampled from our GP posterior, $f \sim \mathcal{GP}(\mu, k | \mathcal{D}_n)$ and then optimized. The sample f must be fixed an we must be able to query it at arbitrary points in order for it to be optimized. Shahriari et al. [38] use spectral sampling techniques to produce approximate samples from the posterior that can be maximized. In this way we can define the TS acquisition function according to the following procedure [39].

$$\alpha_{\mathrm{TS}}(\mathbf{x}; \mathcal{D}_n) := f_n(\mathbf{x}) \quad \text{where} \quad f_n \sim \mathcal{GP}(\mu, k | \mathcal{D}_n) \tag{2.22}$$

Notice that although this procedure is generated from random sampling rather than optimization, it does not avoid the optimization altogether since the GP posterior sample still has to be optimized. Experiments have shown good performance form TS in low dimensions but excessive exploration results in degrading performance in higher dimensions.

Thompson sampling is a sub-procedure used in Predictive Entropy Search and will be discussed in more detail in Chapter 3.

Entropy Search and Predictive Entropy Search

Rather than sampling from the posterior distribution of the global maximizer $\mathbf{x}_\star$, Hennig and Schuler describe an information theoretic method called Entropy Search which seeks to maximize the information about this distribution resulting from the next sample point[17]. This idea is expressed in the corresponding acquisition function - the expected reduction in entropy on the distribution over $\mathbf{x}_\star$ resulting from observing the point $\mathbf{x}$.

$$\alpha_{\mathrm{ES}}(\mathbf{x}) := \mathrm{H}\Big[p(\mathbf{x}_\star|\mathcal{D}\Big)\big] - \mathbb{E}_{p(y_\mathbf{x}|\mathcal{D},\mathbf{x})}\Big[\mathrm{H}\Big[p(\mathbf{x}_\star|\mathcal{D} \cup \{\mathbf{x}, y_\mathbf{x}\})\big)\Big]\Big] \tag{2.23}$$

where $\mathrm{H} = -\int p(\mathbf{x}) \log p(\mathbf{x}) d\mathbf{x}$ is the differential entropy of $p(\mathbf{x})$. However, exact evaluation of (3.1) is feasible only after many approximations [17].

In [18] Hernández-Lobato et al. base their predictive entropy search method on an equivalent but easier to approximate acquisition function.

$$\alpha_{\mathrm{PES}}(\mathbf{x}) := \mathrm{H}\Big[p(y_\mathbf{x}|\mathcal{D})\Big] - \mathbb{E}_{p(\mathbf{x}_\star|\mathcal{D})}\Big[\mathrm{H}\Big[p(y_\mathbf{x}|\mathcal{D}, \mathbf{x}_\star)\Big]\Big] \tag{2.24}$$

We will discuss the derivation and computation of the PES acquisition function in Chapter 3.

Hernández-Lobato et al. show that PES outperforms ES in terms of immediate regret on synthetic and real-world functions and produces better result than the expected improvement (EI) acquisition function. They also state that PES can easily marginalize its approximation with respect to the posterior distribution of its Gaussian process hyperparameters, while ES cannot. We will return to the discussion of marginalizing over the hyperparameters in Chapter 5.

2.4 Evaluation Metric

In order to compare various methods other it is necessary to specify an evaluation metric. A popular metric in the literature consistent with our statement of the Bayesian optimization problem is Immediate regret (IR) which we will simply refer to as regret $r = |f(\tilde{\mathbf{x}}_N) - f(\mathbf{x}_\star)|$ where $\tilde{\mathbf{x}}_N$ is the recommended point at iteration N. In the statement of the Bayesian optimization algorithm (algorithm 1) we recommended the **Latent**

Maximizer:

$$\widetilde{\mathbf{x}}_N = \operatorname*{argmax}_{\mathbf{x}\in\mathcal{X}} \mu_N(\mathbf{x}) \tag{2.25}$$

where μ_N is the predictive posterior mean function at iteration N. From a Bayesian modelling perspective, this is the correct recommendation to use. If we believe our modelling assumptions, including that the likelihood noise is Gaussian, the latent maximizer recommendation will minimize regret in expectation. In practice, however, it may be difficult for Bayesian optimization users to justify recommending a point that, potentially, has not been seen before. In this case, we have the **Best Observed** recommendation.

$$\widetilde{\mathbf{x}}_N = \operatorname*{argmax}_{\mathbf{x}\in\{\mathbf{x}_i\}_{i=1}^N} y_i \tag{2.26}$$

This recommender recommends the $\mathbf{x}$ value corresponding to the best y value observed so far. This recommender restricts the set of possible recommendations to locations that have already been seen. For objectives where the observation noise is small this recommender works reasonably well. In cases where observation noise is large, however, it can mistake high value regions of the objective with locations where a sample $\epsilon_i \sim p(y|f(\mathbf{x}))$ happens to be large. The **Incumbent** recommender is a hybrid of these two approaches:

$$\widetilde{\mathbf{x}}_N = \operatorname*{argmax}_{\mathbf{x}\in\{\mathbf{x}_i\}_{i=1}^N} \mu_N(\mathbf{x}) \tag{2.27}$$

As with the best observed recommender, we restrict our potential recommendations to the previously queried points. With this recommender, however, we leverage our model at these points to select our recommendation, making use of our estimates in our uncertainty.

All three of these recommendations are possible using the `pybo` package. For the experiments in this dissertation, the latent maximizer recommender was used.

I will note here that although the evaluation metric used to evaluate the performance of

PES and IPES, neither optimize that objective directly and we hope that it will perform well indirectly. Although not investigated in this dissertation, it would be interesting to design and experiment to evaluate how well PES and IPES maximize their designed objective, the reduction in entropy over $\mathbf{x}_\star$.

2.5 Modular Bayesian Optimization

In this dissertation, the majority of the implementation was within and experimentation was performed with the Bayesian optimization python package `pybo`, except for the experiments in chapter 4. The `pybo` package was written and is maintained by Matthew W. Hoffman and Bobak Shahriari[1]. In `pybo`, each component of Bayesian optimization is encapsulated within a python object so that the choices necessary for Bayesian optimization can be easily varied to facilitate experimentation. Because there is not "best" setting for these choices that work well on every application [38], providing these choices is not only important for the development of Bayesian optimization but to make it more widely applicable for its users. The authors also view their modular approach as a step toward automating the process of Bayesian optimization. Hoffman and Shariari discuss their design choices and argue for a modular approach to Bayesian optimization in [20].

The particular aspect of Bayesian optimization that is examined in this dissertation is that of the acquisition function (chapter 5.1) and but I also examine some approaches to extend the applicability of the methods in `pybo` (chapter 6).

[1]https://github.com/mwhoffman/pybo

Chapter 3

Predictive Entropy Search

3.1 Entropy Search and Predictive Entropy Search

MacKay [30] proposed using an information theoretic approach for active data collection. Following this approach, Hennig and Schuler proposed a method called Entropy Search [17] which seeks to maximize the information about the location of the maximizer $\mathbf{x}_\star$ at iteration n of Bayesian optimization.

As a measure of the current information about the location of the maximizer, they use the negative differential entropy of the posterior $p(\mathbf{x}_x|\mathcal{D}_n)$. Correspondingly the aim of ES is to maximize the decrease in entropy resulting from querying the function. Hennig and Schuler define the ES acquisition function as:

$$\alpha_{\mathrm{ES}}(\mathbf{x};\mathcal{D}_n) := \mathrm{H}\Big[p(\mathbf{x}_\star|\mathcal{D})\Big] - \mathbb{E}_{p(y_\mathbf{x}|\mathcal{D},\mathbf{x})}\Big[\mathrm{H}\Big[p(\mathbf{x}_\star|\mathcal{D}\cup\{\mathbf{x},y_\mathbf{x}\})\Big]\Big] \tag{3.1}$$

Notice that this function computes the difference in the entropy of the current posterior $p(\mathbf{x}|\mathcal{D}_n)$ and the expected entropy of the posterior after conditioning on an observation at $\mathbf{x}$.

Rather than attempting to balance a tradeoff between exploration and exploitation, the entropy search acquisition function aims to recommend queries that yield the most of information about the location of the maximizer of our objective $\mathbf{x}_\star$. In this way, the function values at each the recommendation $\mathbf{x}_n$ may be low in the hopes that the resulting posterior $p(\mathbf{x}_\star|\mathcal{D}_{n+1})$ has low entropy and we are more certain about the location of the maximizer.

Unfortunately in practice, Hennig and Schuler were only able to compute 3.1 after performing many approximations.

In [18] Hernández-Lobato et al. note that the ES acquisition function is equivalent to the mutual information between $x_\star$ and y which is symmetric.

$$\begin{aligned}
\alpha_{\mathrm{ES}}(\mathbf{x}|\mathcal{D}_n) &= \mathrm{H}[p(x_\star|\mathcal{D}_n)] - \int_y p(y|\mathcal{D}_n,\mathbf{x})\mathrm{H}\Big[p(\mathbf{x}_\star|\mathcal{D}\cup\{(\mathbf{x},y)\})\Big]dy && (3.2)\\
&= \mathrm{H}[\mathbf{x}_\star] - \mathrm{H}[\mathbf{x}_\star|y] && (3.3)\\
&= \mathrm{I}(\mathbf{x}_\star;y) = \mathrm{I}(y;\mathbf{x}_\star) && (3.4)\\
&= \mathrm{H}[p(y|\mathcal{D}_n\mathbf{x})] - \mathbb{E}_{p(x_\star|\mathcal{D}_n)}\Big[\mathrm{H}[p(y|\mathcal{D}_n,\mathbf{x},\mathbf{x}_\star)]\Big] && (3.5)
\end{aligned}$$

and so they define an equivalent acquisition function:

$$\alpha_{\mathrm{PES}}(\mathbf{x}_\star;\mathcal{D}_n) := \mathrm{H}[p(y|\mathcal{D}_n,\mathbf{x})] - \mathbb{E}_{p(x_\star|\mathcal{D}_n)}\Big[\mathrm{H}[p(y|\mathcal{D}_n,\mathbf{x},\mathbf{x}_\star)]\Big] \qquad (3.6)$$

These entropies are computed over predictive posteriors and are much easier to approximate. In fact, the entropy on the left is computed over our GP predictive posterior and can be computed analytically as the entropy of a Gaussian: $\mathrm{H}[p(y|\mathcal{D})] = \log\sqrt{2\pi e\, v_n(\mathbf{x})}$ where $v_n(\mathbf{x})$ is the variance of the predictive distribution (incorporating the likelihood variance).

Three hurdles have to be overcome in order to compute the entropy on the right side of equation 3.6: i) we need to be able to compute or approximate the predictive distribution conditioning on a given point being the location of the maximizer, ii) we need to be able to compute the entropy of such a distribution and iii) we need to be able to compute the expectation of this entropy over the posterior distribution of $\mathbf{x}_\star$.

3.2 PES Approximations

Hernández-Lobato et al. take the following approach to evaluating the right side of equation 3.6: the expectation over $\mathbf{x}_\star$ is computed using a Monte Carlo approximation which will require sampling from $p(\mathbf{x}_\star|\mathcal{D}_n)$. Given a sample $\mathbf{x}_\star \sim p(\mathbf{x}_\star|\mathcal{D}_n)$, moment matching is used to approximate $p(y|\mathcal{D}_n,\{\mathbf{x}_\star,y\})$. The details of these approximations are presented in the following sections.

3.2.1 Sampling from $p(\mathbf{x}_\star|\mathcal{D}_n)$

As discussed in Chapter 2, Thompson sampling is used to approximate sampling from $p(\mathbf{x}_\star|\mathcal{D}_n)$. For motivation, let us consider the finite domain setting, say $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d\}$. Note that the latent function takes the form of a d-dimensional vector $\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_d)]$. Write $p(\mathbf{x}_\star = \mathbf{x}_i|\mathcal{D}_n)$ to denote the probability that the maximum of $\mathbf{f}$ is in position $\mathbf{x}_i$. Then

$$p(\mathbf{x}_\star = \mathbf{x}_i|\mathcal{D}_n) = \int p(\mathbf{f}|\mathcal{D}_n) \prod_{j\leq m} \mathbb{I}[f(\mathbf{x}_i) \geq f(\mathbf{x}_j)]d\mathbf{f}. \tag{3.7}$$

In the finite domain case, this suggests that samples can be generated by i) sampling from posterior $p(\mathbf{f}|\mathcal{D}_n)$ and then ii) maximizing the function $\mathbf{f}$, taking the index of the maximum as $\mathbf{x}_\star$.

We can apply this generative procedure to produce samples from $p(\mathbf{x}_\star = \mathbf{x}_i|\mathcal{D}_n)$ but this requires optimizing samples drawn from our GP posterior. Optimizing such a sample directly would cost $\mathcal{O}(m^3)$ where m is the number of function evaluations required to find the optimum. Instead, Hernández-Lobato et al. optimize an analytic approximation to samples from our GP posterior $f \sim \mathcal{GP}(\mu, k|\mathcal{D}_n)$.

The analytic approximation is the same as the one used for Thomspon sampling by Shahriari et al. in [38]. A description of the procedure is given in [18] and a summary is given below.

Analytic Approximations of GP samples

A kernel function k is called *stationary* or shift-invarient if $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}', \mathbf{0})$. Intuitively, when used as part of a GP prior, we are specifying that the output covariance of two inputs depends only on their relative position. The procedure used for approximating hinges on the use of stationary kernels since it appeals to Bochner's theorem[18]:

Theorem 3.2.1 (Bochner's theorem). *A continuous, stationary kernel is positive definite if and only if it is the Fourier transform of a non-negative, finite measure.*

As a consequence, every stationary kernel k has an associated density $s(\mathbf{w})$, known as the *spectral density*, which is the Fourier dual of k:

Define the associated probability density function by normalizing the spectral density: $p(\mathbf{w}) = s(\mathbf{w})/\alpha$ where $\alpha = \int s(\mathbf{w})d\mathbf{w}$.

$$k(\mathbf{x}, \mathbf{x}') = \int e^{-i\mathbf{w}^\top(\mathbf{x}-\mathbf{x}')} s(\mathbf{w})d\mathbf{w} \tag{3.8}$$

$$s(\mathbf{w}) = \frac{1}{(2\pi)^d} \int e^{i\mathbf{w}^\top \tau k(\tau,\mathbf{0})} d\tau \tag{3.9}$$

We are also able to generate shift invariant kernels given a spectral density $s(\mathbf{w})$, an idea we will return to in chapter 6. By symmetry, equation 3.8 gives us

$$\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= \alpha \int e^{-i\mathbf{w}^\top(\mathbf{x}-\mathbf{x}')} p(\mathbf{w})d\mathbf{w} && (3.10)\\
&= \alpha \int \frac{1}{2} \left[e^{-i\mathbf{w}^\top(\mathbf{x}-\mathbf{x}')} + e^{i\mathbf{w}^\top(\mathbf{x}-\mathbf{x}')} \right] p(\mathbf{w})d\mathbf{w} && (3.11)\\
&= \alpha \int \frac{1}{2} \cos(\mathbf{w}^\top\mathbf{x} - \mathbf{w}^\top\mathbf{x}') p(\mathbf{w})d\mathbf{w} && (3.12)\\
&= \alpha \mathbb{E}_{p(\mathbf{w})}[\cos(\mathbf{w}^\top\mathbf{x} - \mathbf{w}^\top\mathbf{x}') p(\mathbf{w})] && (3.13)
\end{aligned}$$

Notice that $\int_0^{2\pi} \cos(a + 2b)db = 0$ for any constant $a \in \mathbb{R}$. In particular

$$\begin{aligned}
0 &= \int_0^{2\pi} \cos(\mathbf{w}^\top\mathbf{x} + \mathbf{w}^\top\mathbf{x}' + 2b)db && (3.14)\\
&= \int_0^{2\pi} \frac{1}{2\pi} \cos(\mathbf{w}^\top\mathbf{x} + \mathbf{w}^\top\mathbf{x}' + 2b)db && (3.15)\\
&= \mathbb{E}_{p(b)}[\cos(\mathbf{w}^\top\mathbf{x} + \mathbf{w}^\top\mathbf{x}' + 2b)] && (3.16)
\end{aligned}$$

where $p(b)$ is the uniform distribution over $[0, 1]$. Using the identity $2\cos(x)\cos(y) = \cos(x - y) + \cos(x + y)$, continuing our derivation from equation 3.13

$$\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= \alpha \mathbb{E}_{p(\mathbf{w})}[\cos(\mathbf{w}^\top\mathbf{x} - \mathbf{w}^\top\mathbf{x}')] + \alpha \mathbb{E}_{p(b)}[\cos(\mathbf{w}^\top\mathbf{x} + \mathbf{w}^\top\mathbf{x}' + 2b)] && (3.17)\\
&= \alpha \mathbb{E}_{p(\mathbf{w},b)}[\cos(\mathbf{w}^\top\mathbf{x} + b - \mathbf{w}^\top\mathbf{x}' - b) + \cos(\mathbf{w}^\top\mathbf{x} + b + \mathbf{w}^\top\mathbf{x}' + b)] && (3.18)\\
&= 2\alpha \mathbb{E}_{p(\mathbf{w},b)}[\cos(\mathbf{w}^\top\mathbf{x} + b)\cos(\mathbf{w}^\top\mathbf{x}' + b)] && (3.19)
\end{aligned}$$

To reduce the variance of our estimator we can average over stacked samples or size m,

$\mathbf{W}$, $\mathbf{b}$ where $[\mathbf{W}]_i \sim p(\vec{w})$ and $[\mathbf{b}]_i \sim p(b)$.

$$k(\mathbf{x}, \mathbf{x}') = \frac{2\alpha}{m}\mathbb{E}_{p(\mathbf{W},\mathbf{b})}[\cos(\mathbf{W}\mathbf{x} + \mathbf{b})^\top \cos(\mathbf{W}\mathbf{x}' + \mathbf{b})] \tag{3.20}$$

$$= \mathbb{E}_{p(\phi)}[\phi(\mathbf{x})^\top \phi(\mathbf{x})] \tag{3.21}$$

where $\phi(\mathbf{x}) = \sqrt{2\alpha/m}\cos(\mathbf{W}\mathbf{x} + \mathbf{b})$ and $\phi \sim p(\phi)$ is equivalent to $\mathbf{W}, \mathbf{b} \sim p(\mathbf{W}, \mathbf{b})$.

Consider a linear model $f(\mathbf{x}) = \phi(\mathbf{x})^\top\theta$ where $\theta \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. Also assume that observations are corrupted by Gaussian noise - $y_i \sim \mathcal{N}(f(\mathbf{x}_i), \sigma^2)$. Let $\mathcal{D}_n$ be a set of n observations. Then the posterior is also normal, $p(\theta|\mathcal{D}_n, \phi) = \mathcal{N}(\mathbf{m}, \mathbf{V})$ [12] [18] where

$$\mathbf{m} = (\mathbf{\Phi}^\top\mathbf{\Phi} + \sigma^2\mathbf{I})^{-1}\mathbf{\Phi}^\top\mathbf{y} \tag{3.22}$$

$$\mathbf{V} = (\mathbf{\Phi}^\top\mathbf{\Phi} + \sigma^2\mathbf{I})^{-1}\sigma^2 \tag{3.23}$$

and where $[\mathbf{\Phi}]_i = \phi(\mathbf{x}_i)$ and $[\mathbf{y}]_i = y_i$. The predictive distribution is also Gaussian $\mathcal{N}(y; \mu_n(\mathbf{x}), v_n(\mathbf{x}))$ where

$$\mu_n(\mathbf{x}) = \phi(\mathbf{x})^\top(\mathbf{\Phi}^\top\mathbf{\Phi} + \sigma^2\mathbf{I})^{-1}\mathbf{\Phi}^\top\mathbf{y} \tag{3.24}$$

$$v_n(\mathbf{x}) = \phi(\mathbf{x})^\top(\mathbf{\Phi}^\top\mathbf{\Phi} + \sigma^2\mathbf{I})^{-1}\phi(\mathbf{x})^\top\sigma^2 \tag{3.25}$$

Applying the Matrix inversion lemma (appendix A.3 [37]) we can rewrite the predictive distribution mean and variance as:

$$\mu_n(\mathbf{x}) = \phi(\mathbf{x})^\top\mathbf{\Phi}^\top(\mathbf{\Phi}\mathbf{\Phi}^\top + \sigma^2\mathbf{I})^{-1}\mathbf{y} \tag{3.26}$$

$$v_n(\mathbf{x}) = \phi(\mathbf{x})^\top\phi(\mathbf{x}) - \phi(\mathbf{x})^\top\mathbf{\Phi}^\top(\mathbf{\Phi}\mathbf{\Phi}^\top + \sigma^2\mathbf{I})^{-1}\mathbf{\Phi}\phi(\mathbf{x}). \tag{3.27}$$

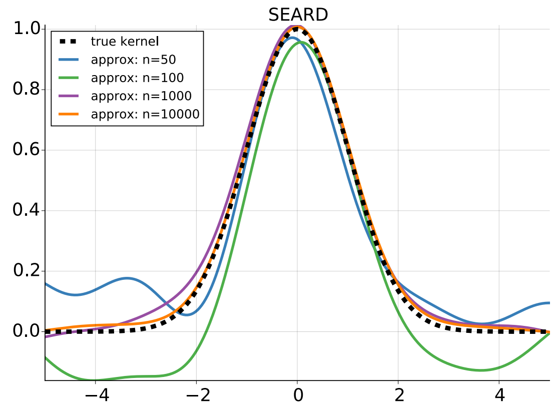
The expectations of these expression over $p(\mathbf{W}, \mathbf{b})$ yield

$$\mu_n(\mathbf{x}) = k(\mathbf{x}, \mathbf{X})^\top(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \tag{3.28}$$

$$v_n(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{X})^\top[\mathbf{K} + \sigma^2\mathbf{I}]^{-1}k(\mathbf{X}, \mathbf{x}'). \tag{3.29}$$

the mean and variance of the GP predictive posterior (with zero prior mean) as in equations 2.7 and 2.8.

In summary, to approximate a sample from $p(\mathbf{x}_\star|\mathcal{D}_n)$ we first sample $\mathbf{W}$ from the spectral density of the kernel k, $\mathbf{b}$ from the uniform distribution over $[0, 2\pi]$ and θ from $\mathcal{N}(0, \mathbf{I})$.



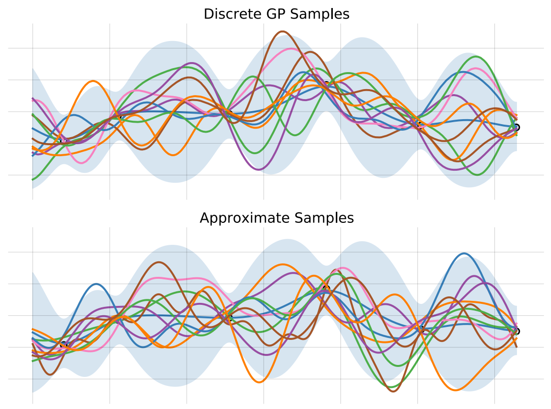


Figure 3.1: *Left*: Approximating the SEARD and MaternARD kernel with 50, 100, and 1000 random features. *Right:* GP samples and random feature approximate samples from a GP predictive distribution.

Treating $f(\mathbf{x}) = \phi(\mathbf{x})^\top \theta$ as a sample from our GP we consider the maximizer as the required sample.

3.2.2 Approximating $p(y|\mathcal{D}_n, \mathbf{x}, \mathbf{x}_\star)$

Given approximate samples of $\mathbf{x}_\star \sim p(\mathbf{x}|\mathcal{D}_n)$, we need to be able to compute the maximizer-conditioned predictive distribution $p(y|\mathcal{D}_n, \mathbf{x}, \mathbf{x}_\star) = \int p(y|f(\mathbf{x}))p(f(\mathbf{x})|\mathcal{D}_n, \mathbf{x}, \mathbf{x}_\star)$. The likelihood component of the integrand $p(y|f(\mathbf{x}))$ is Gaussian by assumption but unfortunately the constraint that $\mathbf{x}$ is a global maximizer of f, that is $f(\mathbf{z}) \leq f(\mathbf{x}_\star)$ for all $\mathbf{z} \in \mathcal{X}$, renders the distribution $p(f(\mathbf{x})|\mathcal{D}_n, \mathbf{x}, \mathbf{x}_\star)$ intractable. To approximate this distribution Hernández-Lobato et al. use a set of weaker constraints (conditions on $\mathbf{x}_\star$) [18] :

A. $\mathbf{x}_\star$ is a local maximizer of f. Although we cannot enforce that $\mathbf{x}_\star$ is a global maximizer, we can condition on it being a local maximizer by requiring that:

A1. $\nabla f(\mathbf{x}_\star) = \mathbf{O}$

A2. $\nabla^2 f(\mathbf{x}_\star)$ is negative definite.

A3. Although not necessary to ensure that $f(\mathbf{x}_\star)$ is a local optima, we make the simplifying assumption that off diagonal elements of the Hessian are zero, upper$[\nabla^2 f(\mathbf{x}_\star)] = 0$. This simplifies A2. to be diag$[\nabla^2 f(\mathbf{x}_\star)] = \mathbf{0}$.

B. $f(\mathbf{x}_\star)$ is larger than past observations. We would like to ensure that $f(\mathbf{x}_\star) \geq f(\mathbf{x}_i)$ for $i \leq n$ but we only have access to the noisy observations y_i and our predictive posterior at $\mathbf{x}_i$. To avoid doing inference in this posterior we make the simplifying assumption that $f(\mathbf{x}_\star) > y_{\max} + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ where $y_{\max}$ is the largest observation so far.
C. $f(\mathbf{x})$ is smaller than $f(\mathbf{x})$. This is to say that we will be approximating the density at $\mathbf{x}$ given that $f(\mathbf{x}) \leq f(\mathbf{x}_\star)$.

Intuitively, instead of conditioning on $\mathbf{x}_\star$ being a global maximizer, we are assuming it is a maximizer i) locally, ii) over the data we have seen so far and ii) over the relevant datapoint under consideration by our distribution. This is a natural approximation since knowledge of the maximizer $f(\mathbf{x})$ is equivalent to $|\mathcal{X}|$ constraints of the form $f(\mathbf{x}) \leq f(\mathbf{x}_\star)$. We are approximating this infinite collection of constraints with a finite one.

We now describe the procedure used by Hernández-Lobato et al. to approximate $p(f(\mathbf{x})|\mathcal{D}_n, \mathbf{A.}, \mathbf{B.}, \mathbf{C.})$. Note that the dependence on the hyperparameter values have been dropped from our notation.

Using the notation outlined in [18], define the latent variables $\mathbf{z} = [f(\mathbf{x}_\star); \mathrm{diag}[\nabla^2(f(\mathbf{x}_\star))]]$ and $\mathbf{c} = [\mathbf{y}_n; \nabla f(\mathbf{x}_\star); \mathrm{upper}[\nabla^2 f(\mathbf{x}_\star)]] = [\mathbf{y}_n; \mathbf{0}; \mathbf{0}]$, constructed by vector concatenation, and consider their distributions: Since f is distributed according to a GP, the vector $[\mathbf{z}, \mathbf{c}]$ is jointly Gaussian. Notice that $\mathbf{c}$ contains the random variables that we need to condition on to enforce A1 and A3. We can compute the kernel matrix $\mathbf{K}$ yielding the covariance of $p([\mathbf{x}, \mathbf{c}])$ and then condition on the required variables using the method described in [43]. Writing

$$\mathbf{K} = \begin{bmatrix} \mathbf{K_c} & \mathbf{K_{zc}} \\ \mathbf{K_{zc}^\top} & \mathbf{K_c} \end{bmatrix} \tag{3.30}$$

we can obtain the conditional Gaussian distribution

$$p(\mathbf{x}|\mathcal{D}_n, \mathrm{A1}) = p(\mathbf{z}|\mathbf{c}) = \mathcal{N}(\mathbf{x}; \mathbf{m}_0, \mathbf{V}_0) \tag{3.31}$$

$$\mathbf{m}_0 = \mathbf{K_{zc}} \mathbf{K_c}^{-1} \tag{3.32}$$

$$\mathbf{V}_0 = \mathbf{K_z} - \mathbf{K_{zc}} \mathbf{K_c}^{-1} \mathbf{K_{zc}^\top} \tag{3.33}$$

The remaining constraints are incorporated into the distribution in observation space via products of terms which enforce these constraints. The term $\Phi_{\sigma^2}(f(\mathbf{x}_\star) - y_{\max})$, where Φ_{σ^2} is the zero-mean and σ^2-variance Gaussian cdf, where incorporates the soft constraint B. where $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2)$ has been marginalized out. The term $\prod_{i=1}^{d} \mathbb{I}\left([\nabla^2 f(\mathbf{x})]_{ii} \leq 0\right)$ incorporates constraint A2. We obtain the following expression

$$p(\mathbf{z}|\mathcal{D}_n, \mathrm{A}, \mathrm{B}) \propto \Phi_{\sigma^2}(f(\mathbf{x}_\star) - y_{\max}) \cdot \left[\prod_{i=1}^{d} \mathbb{I}\left([\nabla^2 f(\mathbf{x})]_{ii} \leq 0\right)\right] \cdot \mathcal{N}(\mathbf{z}; \mathbf{m}_0, \mathbf{V}_0) \tag{3.34}$$

A Guassian approximation $q(z)$ of $p(\mathbf{z}|\mathcal{D}_n, \mathrm{A}, \mathrm{B})$ is computed using Expectation Propagation (EP) [31] [33], to help with integral computations over this distribution. EP replaces each non-Gaussian factor with a gaussian one in the approximation so the result of the EP approximation can be written $q(\mathbf{z}) \propto [\prod_{i=1}^{d} \mathcal{N}(z_i|\widetilde{m}_i.\widetilde{v}_i)]\mathcal{N}(\mathbf{z}|\mathbf{m}_0, \mathbf{V}_0)$.

The next step in the approximation is to focus on the predictive distribution conditioned on the two constraints considered so far, A and B. Let $\boldsymbol{f} = [f(\mathbf{x}); f(\mathbf{x}_\star)]$. Considering the joint distribution of $\boldsymbol{f}$ and $\mathbf{z}$, we further approximate

$$\begin{aligned} p(\boldsymbol{f}|\mathcal{D}_n, \mathrm{A}, \mathrm{B}) &= \int p(\boldsymbol{f}|\mathbf{z}, \mathcal{D}_n, \mathrm{A}, \mathrm{B}) p(\mathbf{z}|\mathcal{D}_n, \mathrm{A}, \mathrm{B}) d\mathbf{z} && (3.35)\\ &\approx \int p(\boldsymbol{f}|\mathbf{z}, \mathcal{D}_n, \mathrm{A}, \mathrm{B}) q(\mathbf{z}) d\mathbf{z} && (3.36)\\ &\approx \int p(\boldsymbol{f}|\mathbf{z}, \mathcal{D}_n, \mathrm{A1}) q(\mathbf{z}) d\mathbf{z} && (3.37)\\ &= \mathcal{N}(\boldsymbol{f}; \mathbf{m}_{\boldsymbol{f}}, \mathbf{V}_{\boldsymbol{f}}) && (3.38) \end{aligned}$$

The last approximation is an assumption that $\boldsymbol{f}$ is independent of condition A2 and B given $\mathbf{z}$. To incorporate the last constrain, C, we can multiply by $\mathbb{I}[f(\mathbf{x}) < f(\mathbf{x}_\star)]$:

$$p(f(\mathbf{x})|\mathcal{D}_m, \mathrm{A}, \mathrm{B}, \mathrm{C}) \approx \frac{1}{Z} \int \mathbb{I}[f(\mathbf{x}) < f(\mathbf{x}_\star)] \mathcal{N}(\boldsymbol{f}; \mathbf{m}_{\boldsymbol{f}}, \mathbf{V}_{\boldsymbol{f}}) d\boldsymbol{f}. \tag{3.39}$$

Note that $\boldsymbol{f}$ is a finite dimensional vector in the above integral. We can approximate this distribution with a Gaussian by moment matching. The variance is given by

$$v_n(\mathbf{x}|\mathbf{x}_\star) = [\mathbf{V}_{\boldsymbol{f}}]_{1,1} - s^{-1}\beta(\beta+\alpha)\left[[\mathbf{V}_{\mathbf{f}}]1,1 - [\mathbf{V}_{\boldsymbol{f}}]_{1,2}\right]^2 \tag{3.40}$$

$$s = [-1,1]^\top \mathbf{V}_{\boldsymbol{f}}[-1,1] \tag{3.41}$$

$$\mu = [-1,1]^\top \mathbf{m}_{\boldsymbol{f}} \tag{3.42}$$

$$\alpha = \frac{\mu}{\sqrt{s}} \tag{3.43}$$

$$\beta = \frac{\phi(\alpha)}{\Phi(\alpha)} \tag{3.44}$$

$$\tag{3.45}$$

where Φ and ϕ are the standard normal cdf and pdf, respectively. Using this approximation the entropy required for PES can be computed as

$$p(y|\mathcal{D}_n, \mathbf{x}, \mathbf{x}_\star) \approx \frac{1}{2}\log\left[2\pi\exp(v_n(\mathbf{x}|\mathbf{x}_n))\right] \tag{3.46}$$

3.3 Problems With PES

In this section, problems with the PES approach and issues limiting its applications are discussed. Solutions that are investigated in this thesis are proposed.

Treatment of Hyperparameters

Up to this point we have mostly ignored the treatment of the hyperparamters of our Gaussian process. One options is to fix and use the hyperparamters that maximizing the marginal likelihood [37]. However, in a data-scares setting such as Bayesian optimization it is very likely that we will overfit the hyperparameters using this strategy. We are uncertain about the values of these hyperparameters and, as with our function values, we would like capture this uncertainty and harness it to guide exploration. This will not be possible using point estimates and so we would like to marginalize out the hyperparameters. In the current implementation of PES this marginalization is performed using the Monte Carlo method outlined by Snoek et al. in [40].

Equation 2.11 framed acquisition functions in terms of the expectations of utility functions where the expectations are taking over the output space values and the hyperparameters:

$$\alpha(\mathbf{x}; \mathcal{D}_n) = \mathbb{E}_\theta \mathbb{E}_v [U(\mathbf{x}, v, \theta)]$$

And so this marginalization is performed "outside" of the utility function. For acquisition functions such as PI this outside marginalization makes sense since the order of integration does not matter.

$$\begin{aligned}
\alpha_{\mathrm{PI}}(\mathbf{x} : \mathcal{D}_n) &= \int_\theta p(\theta|\mathcal{D}_n) \left[\alpha_{\mathrm{PI}}(\mathbf{x} : \mathcal{D}_n, \theta)\right] d\theta & (3.47)\\
&= \int_\theta p(\theta|\mathcal{D}_n) \left[\int_v p(v|\mathcal{D}, \theta) \mathbb{I}[v > \tau] dv\right] d\theta & (3.48)\\
&= \int_v \left[\int_\theta p(v, \theta|\mathcal{D}) \mathbb{I}[v > \tau] d\right] \theta dv & (3.49)\\
&= \int_v p(v|\mathcal{D}) \mathbb{I}[v > \tau] dv & (3.50)
\end{aligned}$$

Where $p(v|\mathcal{D})$ is the marginal (in terms of the hyperparameters) predictive posterior distribution of our GP - we are incorporating the uncertainty in the hyperparameter values into our predictions. The same argument applies to the PI acquisition function.

Indeed, in the current implementation of PES uses this approach:

$$\begin{aligned}
\alpha_{\mathrm{PES}}(\mathbf{x} : \mathcal{D}_n) &= \int_\theta p(\theta|\mathcal{D}_n) \left[\alpha_{\mathrm{PES}}(\mathbf{x} : \mathcal{D}_n, \theta)\right] d\theta & (3.51)\\
&= \int_\theta p(\theta|\mathcal{D}_n) \Big[\mathrm{H}\big[p(y_\mathbf{x}|\mathcal{D}_n, \theta)\big] - \mathbb{E}_{p(\mathbf{x}_\star|\mathcal{D}_n, \theta)}\Big[\mathrm{H}\big[p(y_\mathbf{x}|\mathcal{D}_n, \mathbf{x}_\star, \theta)\big]\Big]\Big] d\theta & (3.52)\\
&\approx \sum_{j=1}^{N} \left[\mathrm{H}\big[p(y_\mathbf{x}|\mathcal{D}_n, \theta^j)\big] - \sum_{i=1}^{M} \mathrm{H}\big[p(y_\mathbf{x}|\mathcal{D}_n, \mathbf{x}_\star^i, \theta^j)\big]\right] & (3.53)
\end{aligned}$$

where $\mathbf{x}_\star^i \sim p(\mathbf{x}_\star|\mathcal{D}_n, \theta)$ and $\theta^j \sim p(\theta|\mathcal{D}_n)$. Unfortunately, the integrals in this expression (the integrals in the entropy and expectation calculations and the θ marginalizing integral) do not commute. As a result, this expression is not equivalent to the acquisition function that we want, which we call Integrated Predictive Entropy Search (IPES):

$$\alpha_{\text{IPES}}(\mathbf{x} : \mathcal{D}_n) = \mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n)\Big] - \mathbb{E}_{p(\mathbf{x}_\star|\mathcal{D}_n)}\Big[\mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n, \mathbf{x}_\star)\Big]\Big] \tag{3.54}$$

$$\approx \mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n)\Big] - \sum_{i=1}^{M} \mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n, \mathbf{x}_\star^i)\Big] \tag{3.55}$$

where $\mathbf{x}_\star^i \sim p(\mathbf{x}_\star|\mathcal{D}_n)$.

The PES treatment of hyperparameters is computationally convenient since, with the previously discussed approximations, the entropies required for the PES acquisition function are computed over univariate Gaussian distributions - this has an analytic solution.

Note that the PES acquisition function computes the expectation over hyperparameter values of the decrease in entropy of the distribution over $\mathbf{x}_\star$ that results from the selection of a datapoint $\mathbf{x}$ **conditioned** on a fixed set of hyperparameters. Suppose, for Bayesian optimization, we employ the strategy of using a fixed set point estimate of hyperparameter values that we do not change throughout our algorithm. This acquisition returns the datapoint that, on average, yields the most information about the location of the maximizer under this strategy. But we know that we **do** change our hyperparameter values, or in the hierarchical Bayesian setting update our distribution over hyperparameters, in light of new information. This acquisition function cannot take into account any reduction in the uncertainty in our hyperparameter values when recommending the next datapoint.

The IPES aquisition function is identical to the original PES aquision function where the hyperparameters have been marginalized out of the predictive distribution and so it is able to recommend points taking into account the uncertainty and reduction in uncertainty in the hyperparameters.

Kernel Restrictions

Our procedure for sampling from $p(\mathbf{x}_\star|\mathcal{D}_n)$ required us to sample from the spectral density of our Gaussian process kernel. Some kernels, such as the periodic kernel, are stationary, but sampling from their spectral density can be difficult. This procedure also appealed to Bochner's theorem and relied on the use of stationary kernels. Although this simplification makes regression easier, it greatly restricts the space of functions that we can model well. In particular, it is difficult to model non-stationary functions with such GPs. This is a problem since many of the objective functions for applied problems for which Bayesian optimization is well suited, such as optimizing the hyperparameters of a machine learning algorithm [41], are non-stationary.

In chapter 6 I explore three strategies to overcome these issues in order to make (I)PES applicable to more Bayesian optimization problems. As a first approach, a more flexible kernel called the spectral mixture (SM) kernel [45] is implemented. As the name suggests, the kernel was designed to have a Gaussian mixture model spectral density, from which it is easy to sample. Secondly, I examine input space warping and implement the BetaCDF input space warping examined by Snoek et al. [41] to better model non-stationary functions. Thirdly, I attempt to remove the dependency on stationary kernels all together by introducing an alternate sampling technique for $p(\mathbf{x}_\star|\mathcal{D}_n)$.

Chapter 4

Entropy Approximations of Gaussian Mixture Models

4.1 GMMs in IPES

The IPES acquision function

$$\alpha_{\mathrm{IPES}}(\mathbf{x}:\mathcal{D}_n) = \mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n)\Big] - \mathbb{E}_{p(\mathbf{x}_\star|\mathcal{D}_n)}\Big[\mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n,\mathbf{x}_\star)\Big]\Big] \tag{4.1}$$

requires computing the entropy over marginal predictive distributions $p(y_{\mathbf{x}}|\mathcal{D}_n)$ and $p(y_{\mathbf{x}}|\mathcal{D}_n,\mathbf{x}_\star)$. Using the procedure from Snoek et al. [42] we can sample hyperparameters from the posterior $p(\theta|\mathcal{D}_n)$ allowing us to obtain Monte Carlo approximations of these marginal predictive distributions

$$p(y_{\mathbf{x}}|\mathcal{D}_n) = \sum_{j=1}^{M} p(y_{\mathbf{x}}|\mathcal{D}_n,\theta^j) \quad \text{and} \quad p(y_{\mathbf{x}}|\mathcal{D}_n,\mathbf{x}_\star) = \sum_{j=1}^{M} p(y_{\mathbf{x}}|\mathcal{D}_n,\mathbf{x}_\star,\theta^j) \tag{4.2}$$

Note that the PES appoximation of $p(y_{\mathbf{x}}|\mathcal{D}_n,\mathbf{x}_\star,\theta^j)$ yielded a Gaussian distribution. Utilizing this approximation, both $p(y_{\mathbf{x}}|\mathcal{D}_n,\theta^j)$ and $p(y_{\mathbf{x}}|\mathcal{D}_n,\mathbf{x}_\star,\theta^j)$ are Gaussians. And so, the approximations in 4.2 yield Gaussian mixture models (GMM).

Continuing along these lines, our IPES aquision function approximation requires that we compute the entropies of GMMs. Some approaches for doing so are discussed in the following section.

4.2 Taylor Approximations

Let us define our GMM density function as $p(\mathbf{y}) = \sum_{m=1}^{M} c_m \mathcal{N}(\mathbf{y}; \mu_m, \sigma_m)$. Then we are interested in computing

$$\mathrm{H}[\mathbf{y}] = -\int p(\mathbf{y}) \log p(\mathbf{y}) d\mathbf{y} \tag{4.3}$$

In this section I describe a technique proposed by Huber et al. [22] for approximating the entropy of a GMM by cleaverly utilizing its Taylor series expansion. They propose a method to approximate Gaussian mixture entropies for multivariate GMMs but in our case $p(y|\mathcal{D}_n, \mathbf{x})$ is univariate which simplifies the situation quite a bit. We will focus on this case.

To distinguish between the GMM pdf inside and outside of the logarithm in the computation of 4.4, the authors write

$$\mathrm{H}[y] = -\int f(y) \log g(y) dy \tag{4.4}$$

with $f(y) = g(y) = p(y)$.

Let $T_N^m(y)$ be the N-th taylor approximation of $\log g(y)$ about μ_m, the mean of the Gaussian component $\mathcal{N}(\mathbf{y}; \mu_m, \sigma_m)$:

$$T_N^m(y) = \sum_{n=1}^{N} \frac{g^{(n)}(\mu_m)}{n!}(y - \mu_m)^n \tag{4.5}$$

where $g^n(y)$ is the n-th derivative of $g(y)$. The approxiation is achieved by writing

$$\begin{aligned}
\mathrm{H}[y] &= -\int f(y) \log g(y) dy && (4.6)\\
&= -\int \sum_{m=1}^{M} c_m \mathcal{N}(y; \mu_m, \sigma_m) \log g(x) dy && (4.7)\\
&= -\sum_{m=1}^{M} c_m \int \mathcal{N}(y; \mu_m, \sigma_m) \log g(x) dy && (4.8)\\
&\approx -\sum_{m=1}^{M} c_m \int \mathcal{N}(y; \mu_m, \sigma_m) T_N^m(y) dy && (4.9)\\
&= -\sum_{m=1}^{M} c_m \sum_{n=1}^{N} \frac{g^{(n)}(\mu_m)}{n!} \int \mathcal{N}(y; \mu_m, \sigma_m)(y-\mu_m)^n dy && (4.10)
\end{aligned}$$

which is a weighted sum of gaussian moments. Note that these moments have a closed from

$$\int \mathcal{N}(y; \mu_m, \sigma_m)(y-\mu_m)^n dy = \begin{cases} 0 & n \text{ is odd} \\ (\sigma_m)^n \cdot (n-1) \cdot (n-3) \cdot \ldots \cdot 3 \cdot 1 & n \text{ is even} \end{cases} \tag{4.11}$$

Now, all that are needed for the approximation are the derivatives $g^{(n)}(y)$. For a fixed N these can be computed by hand. In the experimentation, a Taylor approximation up to order four was computed by hand. In order to observe the effect of increasing the degree of the Taylor approximations on the accuracy of our approximation, arbitrary order Taylor approximations were implemented using the automatic differentiation software `autograd`[1] from the Harvard Intelligent Probabilistic Systems Group.

4.3 Quadrature Methods

Since the distributions we are approximating are univariate, it is possible that quadrature methods for approximating integrals would perform well. A quadrature approximation has a nice feature in that the computational complexity of the entropy calculation does not grow with the number of components of our GMM. Of course, most quadrature methods are already going to be much slower than the Taylor-based entropy approximation presented in the previous section. Experiments with quadrature

[1]https://github.com/HIPS/autograd

were performed using the python scientific computing package `scipy`[2], in particular the `scipy.integrate.quad`[3] method which uses a numerical integration technique taken from the `Fortran` library `QUADPACK`.

For our experiments, we are unable to compute the entropy analytically so we need to deterime a target for comparison. For this purpose, I implemented a simple Riemann Sum approximation with a fine resolution partition of the domain (approximately 10^y sample points). The limits of the apprximate integral were set to the points where the function approximately vanishes.

4.4 Experimental results

We first compare our Reimann sum approximation that we consider the ground truth (except in the case of a single gaussian where we compute the entropy analytically) to the `scipy quad` method. For this comparison we approximate the entropy of gaussian mixture models whose means are drawn uniformly on the interval $[-20, 20]$ and variances drawn uniformly from $[0, 5]$.

As to be expected, the two quadrature approximate integrals match almost exactly, as shown in figure 4.1.

Since the results were almost identical and the `scipy quad` method runs much faster (about 10^5 times faster), for our Taylor approximation experiments we will use the `scipy quad` approximation as our ground truth.

For the Taylor approximations, entropies of 30 randomly generated GMM with $0, 11, 21, 31$ and 41 Gaussian components were approximated. The enropy of each GMM was approximated using `scipy quad` (as the ground trouth), our “manual” Taylor approximations, where the derivatives were calculated and implemented by hand for Taylor polynomials 2 and 4, and “auto” Taylor approximations where the derivatives were handled by `autograd` for Taylor polynomials $2, 4, 6$ and 8. Note that odd moments of Gaussians are zero so it was not necessary to test odd Taylor approximations. The results of these experiments are shown in figure 4.2.

[2]https://www.scipy.org/

[3]http://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html

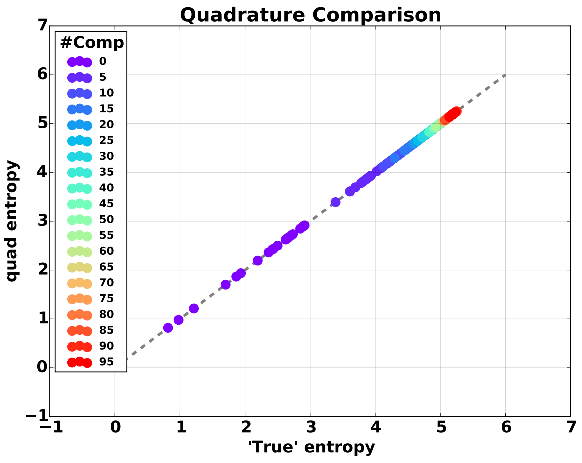


Figure 4.1: A comparison of "true" GMM entropy approximations given by implemented Riemann sum method and the `scipy quad` method. The legend shows the number of components

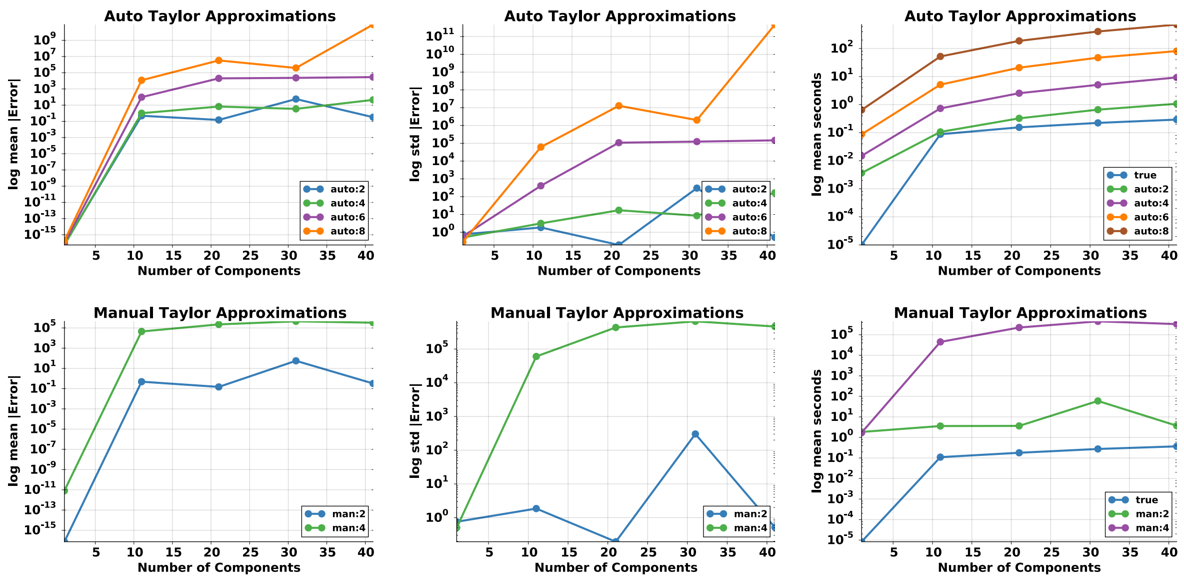


Figure 4.2: Taylor series entropy approximations. The top row of plots were produced using `Autograd` derivatives and the bottom row using manually implemented derivatives. The left and centre plots are the log mean absolute difference, and log standard deviation absolute difference between the taylor approximations and the quadrature approximation produced by `scipy quad` respectively. Each datapoint represents 30 runs. The right plots are the mean wallclock time to produce each approximation in seconds. All y-axis are on the log scale.

As expected, the mean and standard deviaitons of errors for the manual and auto Taylor approximations 2 and 4 line up almost exactly. Unsurprisingly, the accuracy and robustness, in term of standard deviation, degrade as more Gaussian components are used in the GMM. It is surprising, however, that after about 4 components, increasing the number of Taylor components in our approximation seems to decrease the accuracy and drastically increase the variance in our approximation. This is possibly due to numerical issues when evaluating derivatives that vanish at the means of the GMM.

The implementation of the Taylor approximations is also significantly slower than the `scipy quad` implementation. The Taylor approximations become slower as more Taylor components are added - additions that apparently make the approximations worse.

As a consequence of these experiments the `scipy quad` based entropy approximations were used in my implementation of IPES in pybo and reggie.

Chapter 5

Integrated Predictive Entropy Search

5.1 The IPES Approximation: Putting It All Together

The IPES acquisition function

$$\alpha_{\text{IPES}}(\mathbf{x} : \mathcal{D}_n) = \mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n)\Big] - \mathbb{E}_{p(\mathbf{x}_\star|\mathcal{D}_n)}\Big[\mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n, \mathbf{x}_\star)\Big]\Big] \tag{5.1}$$

$$\tag{5.2}$$

is computed as follows: a set of N samples from the hyperparameter posterior are gathered using MCMC slice sampling as described in [40] and [34]. Using this hyperparameter sample, the hyperparameters are marginalized out of the GP predictive posterior using a Monte Carlo approximation

$$p(y_x|\mathcal{D}_n) \approx \sum_{j=1}^{N} p(y_x|\mathcal{D}_n) \tag{5.3}$$

resulting in a Gaussian mixture model. The entropy of this GMM is computed using a quadrature approximation, as discussed in chapter 4. By using Fourier approximate samples from the GP posterior we collect a sample of maximizer locations $\mathbf{x}_\star$. The PES approximation for the maximizer and hyperparameter conditioned predictive posterior yields a Gaussian. This approximation is used in the computation of the IPES acquisition function. Again, we can marginalize the hyperparameters out of this distribution using Monte Carlo with the same hyperparameter sample, and compute the entropy using quadrature. Finally, the expectation, of this entropy is computed using Monte Carlo via our maximizer sample. The difference between the PES and IPES computations is highlighted in equations 5.4 and 5.5.

$$\alpha_{\text{PES}}(\mathbf{x}) = \sum_{j=1}^{N} \left[\mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n, \theta^j)\Big] - \sum_{i=1}^{M} \mathrm{H}\Big[p(y_{\mathbf{x}}|\mathcal{D}_n, \mathbf{x}_\star^i, \theta^j)\Big] \right] \tag{5.4}$$

$$\alpha_{\text{IPES}}(\mathbf{x}) = \mathrm{H}\Big[\sum_{i=1}^{N} p(y_{\mathbf{x}}|\mathcal{D}_n, \theta^j)\Big] - \sum_{i=1}^{M} \mathrm{H}\Big[\sum_{i=1}^{N} p(y_{\mathbf{x}}|\mathcal{D}_n, \mathbf{x}_\star^i, \theta^j)\Big] \tag{5.5}$$

5.1.1 Implementation Issues

Although we are willing to pay the extra computational cost from Modelling our objective an solving our acquisition function to produce recommendations, this overhead makes experimentation a bit difficult.

Since quadrature is used to compute the acquisition function, gradients based optimization methods such as LBFGS cannot be used to optimize α.

Implemented in `pybo` is the `Grid` domain object that equally partitions the problem domain into a d-dimensional grid of candidate query points. The domain is then treated as discrete and the acquisition function optimized over these point. This domain object performed well for low dimensional problems when we were able to use a sufficiently fine grid resolution. To apply IPES to higher dimensional problems I implemented a `Sampled` domain object which uniformly sampled a collection of candidate query points from our domain at each iteration. Using this method allows increasing the dimensionality of our problem without increasing computational cost but performance will obviously degrade as we will explore fewer regions in our domain. We appeal to randomness to hopefully not miss out on important parts of our domain that can fall through the cracks of the grid.

5.2 Experimental Results

I present the results of experiments to determine the performance of Bayesian optimization using the IPES acquisition function when compared to using the acquisition functions discussed in chapter 2. In particular, we are interested in measuring its performance against PES.

For the experiments, broad uninformative uniform or log normal prior distributions for the hyperparameters were used. The kernel function SEARD implement `pybo` was used. We compare the (immediate) regret of the different methods as a function of objective function evaluations. Many of the objectives used in this section are implemented in the `benchfunk`[1] python package by Hoffman and Shahriari intended for experimentation with `Pybo`.

Note that the summary statistics for each plot is only over 50 random initializations starting from a sample of three points due to the high computational costs of these methods. The variance in each case is pretty large due to the small sample size however, we will still draw conclusions from the expectations.

5.2.1 Experiment 1: Sinusoidal Function

For our first set of experiments, we use the one dimensional sinusoidal function defined below:

$$f(x) = -\cos(x) - \sin(3x) \quad 0 \leq x \leq 2\pi \tag{5.6}$$

The Thompson Sampling (TS), Upper Confidence Bound (UCB), Probability of Improvement (PI), Expected Improvement (EI), Predictive Entropy Search (PES) and Integrated Predictive Entropy Search (IPES) methods were all used. Again note the large variance over such few runs, especially in the case of Probability of Improvement. On this objective, TS does pretty well, especially for $\sigma^2 = 0.01$. Notice that in all cases, both PES and IPES are consistently toward the bottom of the collection of curves and perform better than the other acquisitions in the higher noise settings of $\sigma^2 = 0.1$ and 0.5. For the lower noise settings, IPES initially performs worse than PES but eventually

[1]https://github.com/mwhoffman/benchfunk

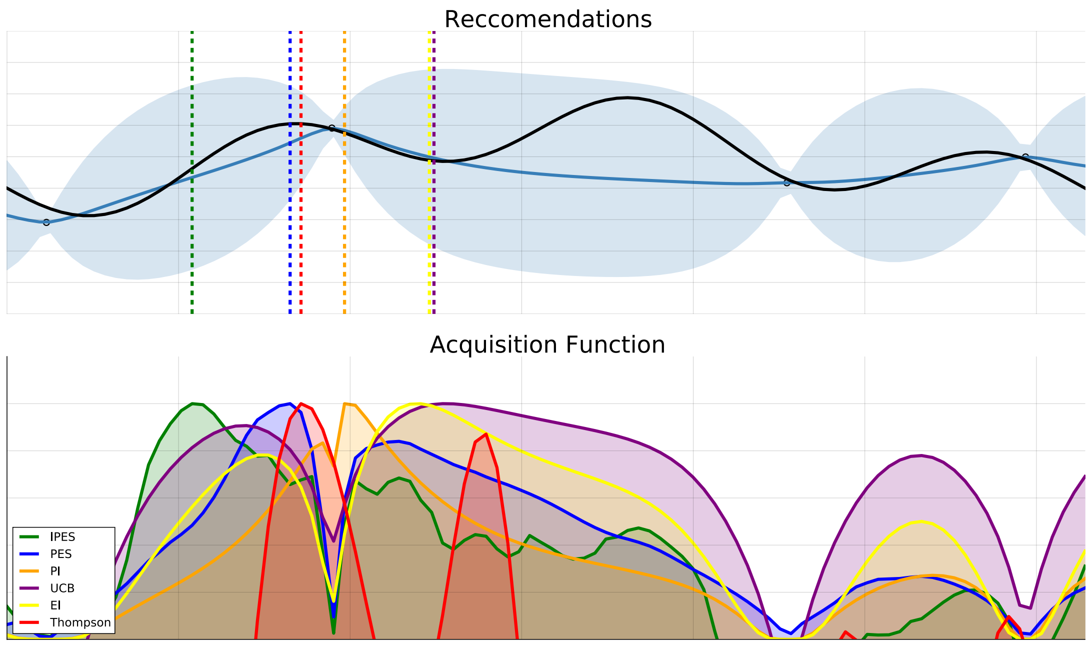


Figure 5.1: A visualization of the recommendations produced by the acquisition functions tested. Notice the difference in the estimates yielded by PES and IPES of the reduction in predictive entropy resulting from queries locations.

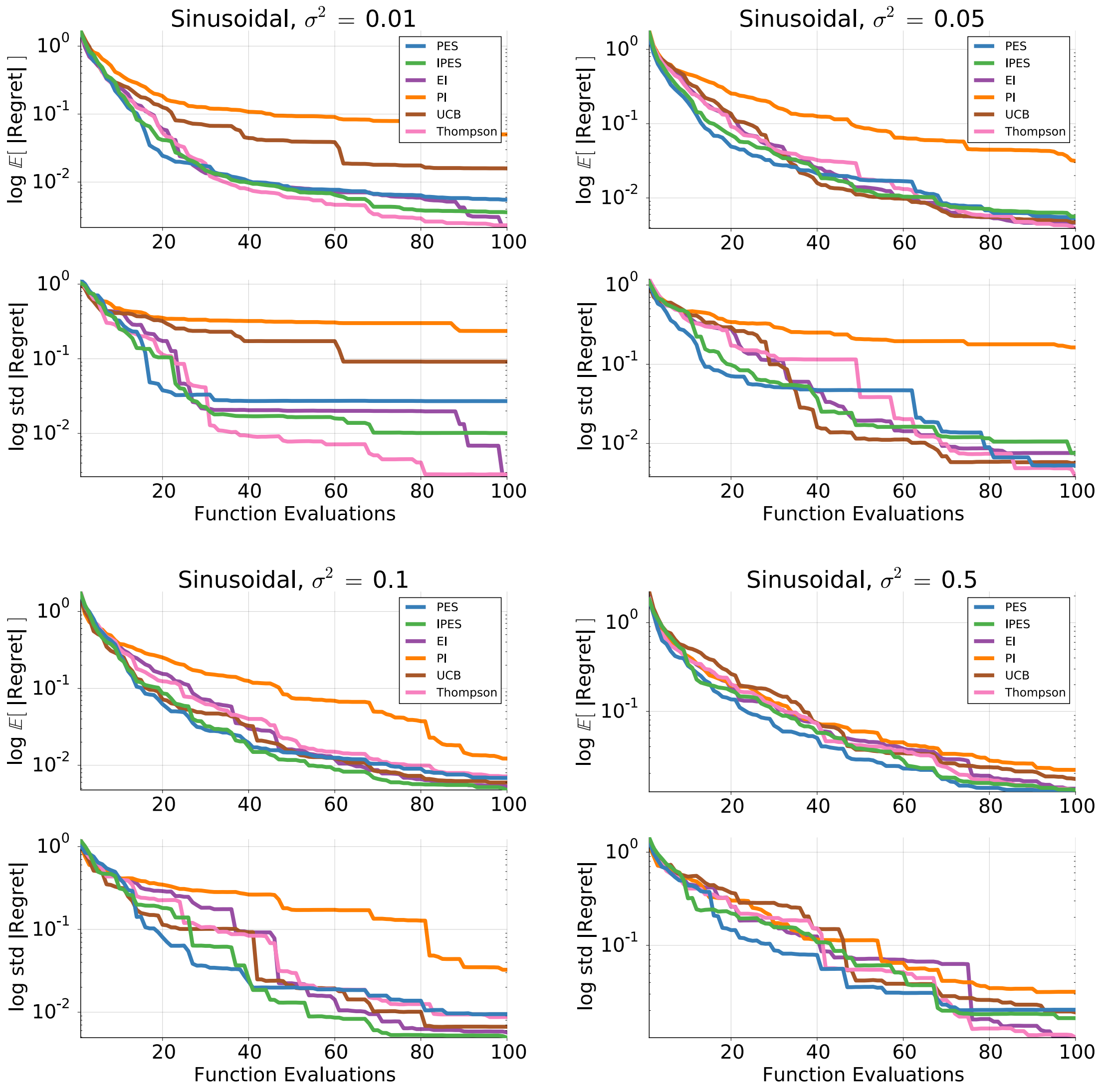


Figure 5.2: A comparison of various acquisition functions applied to the sinusoidal function (equation 5.6), with likelihood noise 0.01, 0.05, 0.1 and 0.5

overtakes or equals with the error values of PES. Notice that, generally, acquisitions functions that are known to perform more exploration (TS, USB) perform better after many iterations than those that are known to be more myopic (PI, EI).

5.2.2 Experiment 2: The Gramacy Function

For our second set of experiments we use the one dimensional objective used by Gramacy and Lee in [14]:

$$f(x) = \frac{\sin(10\pi x)}{2x} - (x-1)^4. \tag{5.7}$$

Again, on this objective, we compare TS, UCB, PI, EI, PES and IPES methods were examined. As mentioned before, no acquisition function is known to be optimal on all problems and it is evident when comparing acquisition performance on the sinusoidal objective to that on the Gramacy objective. Thompson sampling does surprisingly well, especially for $\sigma^2 = 0.01$. Whereas in the sinusoidal case, functions that spent more effort exploring performed well, here they do not. Notice that PI, the worst performing acquisition function in on the sinusoidal objective is the best performing one here. Note that IPES performs rather poorly in this case and more poorly than PES in the low likelihood noise settings. When likelihood noise is larger IPES performs about on par or better than PES, perhaps because learning about the likelihood noise is more important in these cases - although this was not observed on the sinusoidal objective.

Although not specifically designed with the exploration/exploitation trade-off in mind, the results on the sinusoidal and Gramacy function suggest that IPES is, unsurprisingly, more on the exploration end of the trade-off.

5.2.3 Experiment 3: The Modified Branin Function

For the treatment of the hyperparameters to matter it must be the case that gaining information about the hyperparameters is important to the task of optimizing. If we have too few hyperparameters, a large spectrum of hyperparameters values perform pretty well, or learning about the functions is more important than learning about the hyperparameters we should not expect to see a performance difference between IPES and

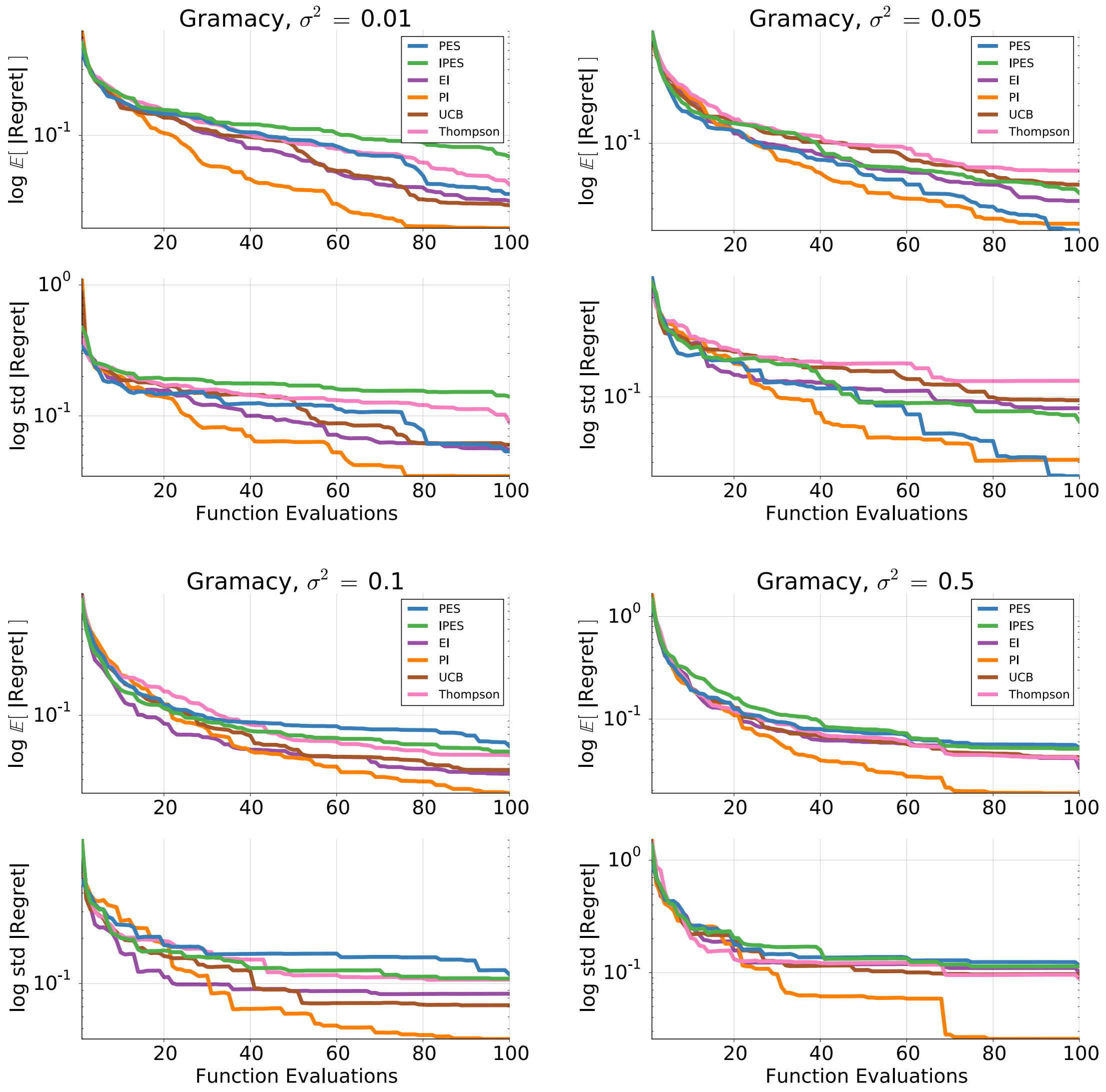


Figure 5.3: A comparison of various acquisition functions applied to the Gramacy function with likelihood noise 0.01, 0.05, 0.1 and 0.5

PES. Also, in a data scares setting, as we increase the number of hyperparameters in our model the more important a fully Bayesian treatment of the hyperparameters becomes. In an attempt to draw a more clear distinction between these two methods, I attempted to modify the two dimensional Branin function and embed it in four dimensional space. The objective used was defined as

$$f(x_1, x_2, x_3, x_4) = \frac{1}{10}\left(x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + (1 - \frac{1}{8\pi})\cos x_1 + \frac{1}{2}x_3^2 - 10x_4^2 \quad (5.8)$$

$$-5 \leq x_1 \leq 10, \quad 0 \leq x_2 \leq 15, \quad -1 \leq x_3 \leq 1, \quad -1 \leq x_4 \leq 1 \quad (5.9)$$

This example failed to drastically distinguish between the two methods, although IPES does perform slightly better overall on this objective (except perhaps between iterations 40 and 80 on $\sigma^2 = 0.05$)

HyperParameter learning

We expect the hyperparameter posteriors to converge more quickly, and hopefully better estimates, using IPES compared to PES. This is difficult to evaluate because we do not know what a good hyperparameter distribution at iteration n of Bayesian optimization would be. The entropy of the hyperparameter posterior might be an interesting metric (although one I did not examine it) and one we can estimate via Monte Carlo methods. It is possible, however, that this metric would reward overfitting of the hyperparameters and overconfidence in estimates.

In this section we plot visualizations of samples from the hyperparameter posteriors at various iterations of Bayesian optimization using PES and IPES. Densities for each sample were are estimated and plotted using the Gaussian kernel density estimation from `scikit-learn`[2].

Figure 5.5 shows four dimensional length scale samples at iterations 1,17, 50, 99. Figure 5.6 shows kernel variance hyperparameter samples at iterations 1, 17, 25, and 100. Figure 5.7 shows likelihood variance samples at iterations 1, 25, 50, 99.

[2]http://scikit-learn.org

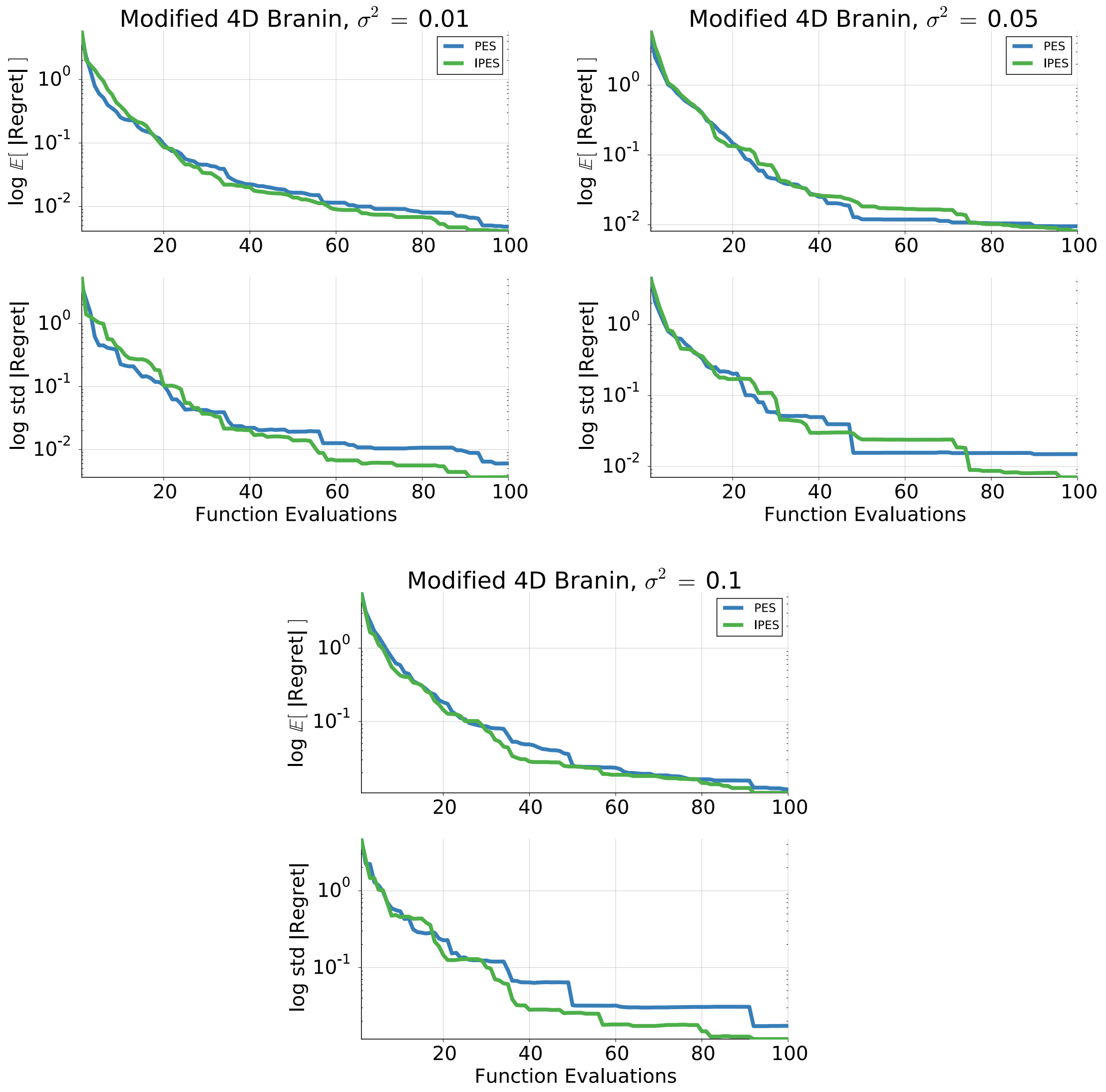


Figure 5.4: A comparison of various PES and IPES acquisition functions applied to the modified four-dimensional Gramacy function with likelihood noise 0.01, 0.05, and 0.1

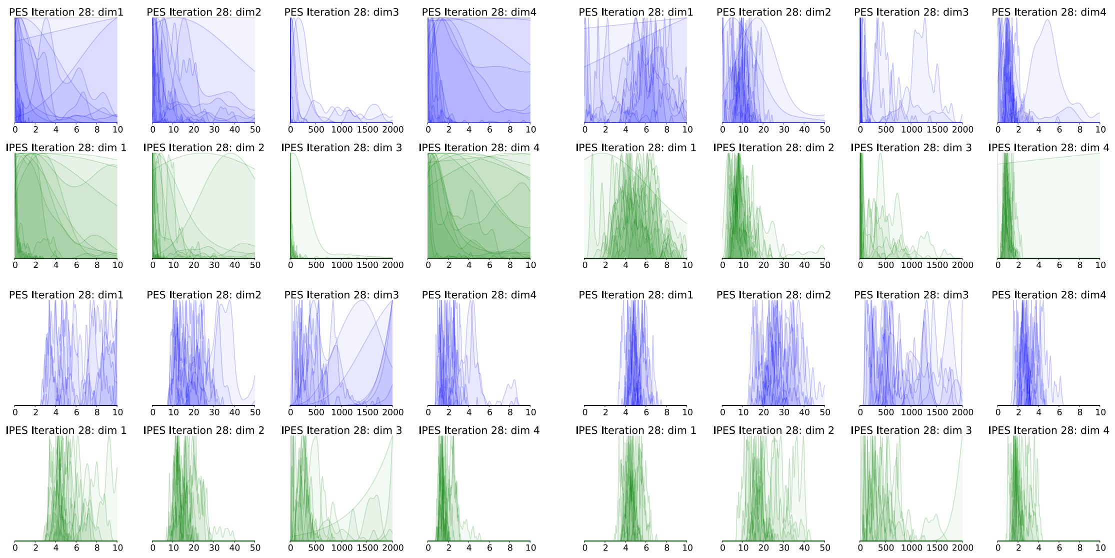


Figure 5.5: Four dimensional kernel length scale hyperparameters samples from the posterior at iterations 1 (*top left*), 17 (*top right*), 50 (*bottom left*) and 99 (*top right*). Blue corresponds to the PES method and green corresponds to the IPES method.

Generally, we observe that after 100 samples, the samples from each posterior taken when using PES or IPES look similar immediately after initialization and after 100 iterations. We take the fact that both sets of distributions yield similar looking collections of samples after 100 iterations as evidence that these final distributions are pretty good.

The sample profiles from the two methods separate from each other - suggesting that they are learning about the hyperparameters at different rates. Interestingly, when hyperparameters samples collapse to being located around a smaller location when compared initial samples, IPES does so more quickly - evidence that the quires that IPES makes yields more information about the hyperparameters. However, when the hyperparameter samples expand, they do so more slowly for IPES.

5.2.4 Experiment 4: The Six Dimensional Hartmann Function

Again, to distinguish between the performance and behaviour of PES and IPES, we increase the number of relevant hyperparameters and apply these methods to an even

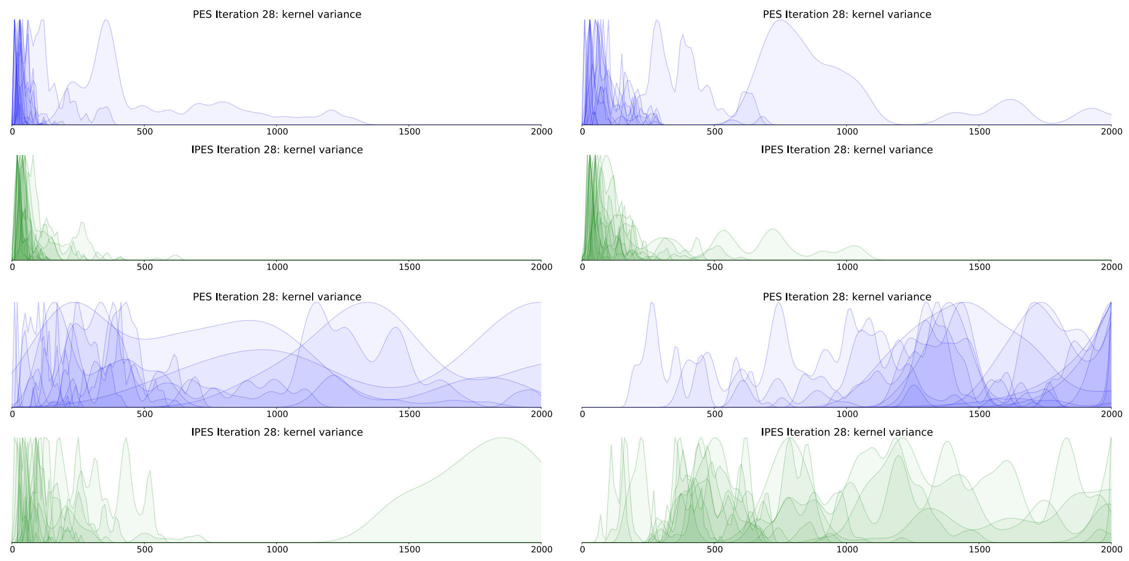


Figure 5.6: Kernel variance samples from the posterior at iterations 1 (*top left*), 17 (*top right*), 25 (*bottom left*) and 100 (*top right*). Blue corresponds to the PES method and green corresponds to the IPES method.

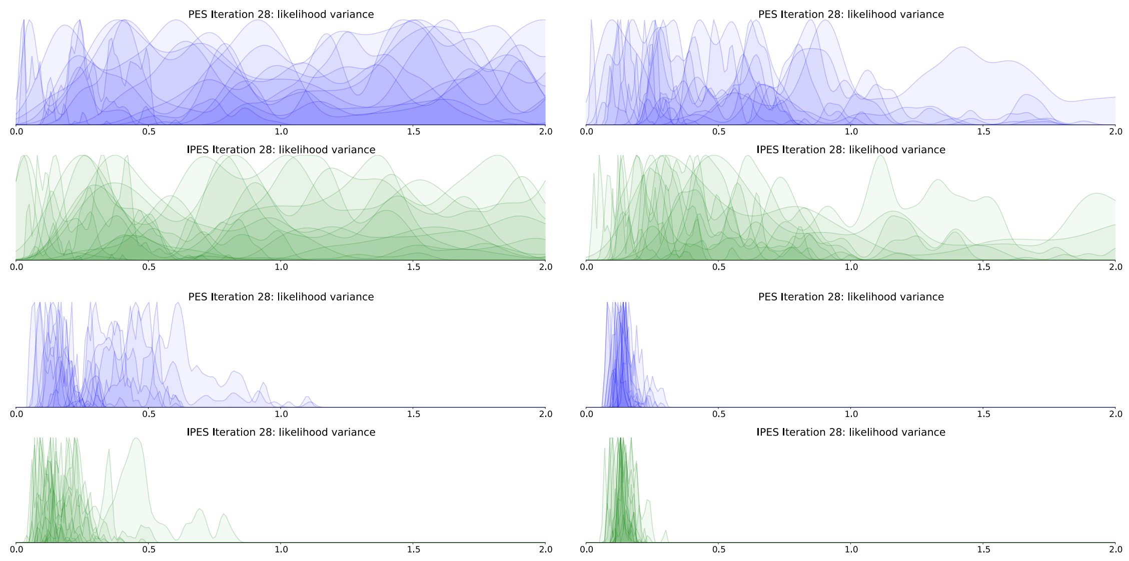


Figure 5.7: Likelihood variance hyperparameters samples from the posterior at iterations 17 (*top left*), 25 (*top right*), 50 (*bottom left*) and 99 (*top right*). Blue corresponds to the PES method and green corresponds to the IPES method.

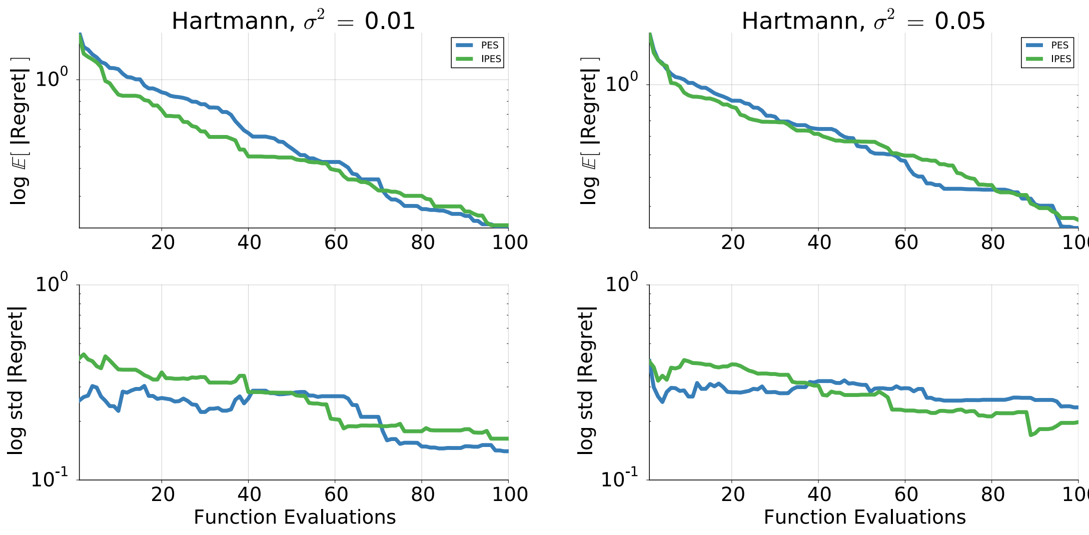


Figure 5.8: A comparison of various PES and IPES acquisition functions applied to the the Hartmann six-dimensional function with likelihood noise 0.01, 0.05, and 0.1

higher dimensional problem - the six Dimensional Hartmann Function [1]:

$$f(\mathbf{x}) = -\sum_i c_i \exp\left(-\sum_{j=0}^{n-1} A_{i,j}(x_j = p_{i,j})^2\right) \quad \text{where} \tag{5.10}$$

$$A = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix} \quad c = [1, 1.2, 3, 3.2]^\top \tag{5.11}$$

$$p = \begin{bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{bmatrix} \tag{5.12}$$

Figure 5.8 shows that the performance trend of the previous experiments continues - performance of both methods are very close. In this case IPES performs slightly better than PES in the early stages of Bayesian optimization and PES performing better later. Note that the larger variance in results is due to statistics being computed over 30 runs instead of 50.

Chapter 6

Extending the Applicability of IPES

In order to implement a kernel $k(\mathbf{x}, \mathbf{x}')$ for its use in PES and IPES in the `Pybo`:

1. We must be able to implement the first and second partial derivatives with respect to the input space: $\frac{\partial k}{\partial \mathbf{x}}$ and $\frac{\partial^2 k}{\partial \mathbf{x}' \partial \mathbf{x}}$.

2. The kernel must be stationary: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{a}, \mathbf{x}' - \mathbf{a})$ for any $\mathbf{a} \in \mathbb{R}^D$.

3. We must be able to sample from the (normalized) kernel spectral density $p(\mathbf{w}) = \frac{1}{(2\pi)^d \alpha} \int e^{i\mathbf{w}^\top \tau k(\tau, \mathbf{0})} d\tau$

These conditions severely restrict the class of kernels that can be used for (I)PES, particularly the stationarity requirement, and the performance of the IPES method on real world problems. In the following sections we examine some more flexible kernels than SEARD that satisfy these conditions and examine some strategies for weakening these conditions in order to extend the class of problems on which (I)PES can be effective.

6.1 Flexible Kernels

The first approach will be to implement more flexible kernels that help better model the types of real world objectives that we are interested in yet still satisfy the above requirements.

6.1.1 Spectral Mixture Kernel

The spectral mixture kernel (SM) was introduced by Wilson and Adams in [45]. Rather than defining a kernel and then attempting to derive its spectral density, the authors take the reverse approach my first defining the class of kernels having a spectral density of the form $S(\mathbf{s}) = [\phi(\mathbf{s}) + \phi(-\mathbf{s})]/2$ where

$$\phi(\mathbf{s}) = \sum_{q=1}^{Q} w_q \cdot \mathcal{N}(\mathbf{s}; \mu_q, \mathbf{M}_q), \tag{6.1}$$

$\mu_q = [\mu_q^{(1)}, \ldots, \mu_q^{(D)}]$, $\mathbf{M}_q = \operatorname{diag}[v_q^{(1)}, \ldots, v_q^{(D)}]$, $w_q \in \mathbb{R}$, and D is the dimension of our input space.

We can easily sample from the spectral density because it is simply a linear combination of Gaussians. Since we can easily sample from the kernel spectral density, this kernel is well suited for the Thompson sampling method as well as producing Fourier approximate GP samples from the GP posterior used to approximate samples from $p(\mathbf{x}_\star|\mathcal{D}_n)$ in (I)PES.

The authors derive the kernel defined by this spectral density, as discussed in chapter 3.6 - the Spectral Mixture kernel:

$$k(\tau) = \sum_{q=1}^{Q} w_q \cos(2\pi {\tau_p}^2 {v_p}^{(p)}) \exp(-2\pi \tau_p {\mu_q}^{(p)}) \tag{6.2}$$

The SM kernel is very flexible and the authors were able to well approximate many popular kernels with various hyperparameter values (see figure 6.1 In the implementation of this kernel, it was necessary to compute and implements its partial derivatives mentioned above. The details of this implementation is given in appendix ???????

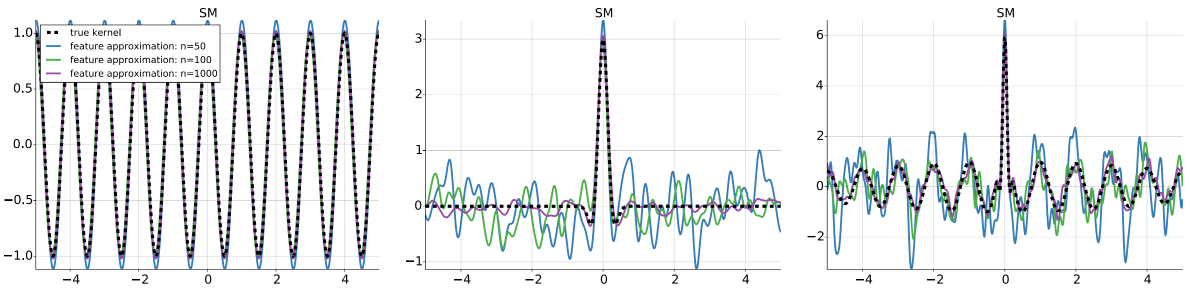


Figure 6.1: Visualizations of the Spectral Mixture (SM) kernel and Fourier approximations. *Left:* Approximating a periodic kernel using a SM kernel with one component with large variance hyperparameter. *Centre:* SM kernel with one component. *Right.* SM kernel with three components.

6.1.2 Input Space Warping

Although it is required the kernel be stationary on the space on which it is applied, we are free to first map our objective input space to a new kernel input space:

$$\underbrace{\mathbb{R}^D \times \mathbb{R}^D}_{\text{input space}} \xrightarrow{w} \underbrace{\mathbb{R}^{D'} \times \mathbb{R}^{D'}}_{\text{warped space}} \xrightarrow{k} \mathbb{R} \tag{6.3}$$

This procedure is called input warping. Similar to the way that kernel functions allows us to extend finite dimensional euclidean inner products to more general and flexible measures of similarity, such as the inner product of an infinite dimensional vector space embedding (chapter 6 from [4]), this simple but powerful idea allows us to add a lot of flexibility to our class of stationary kernels and model non-stationary behaviour.

Snoek et al. [41] introduce a particular input space warping using the Beta distribution's cdf functions. The warping was introduced specifically for their use with stationary kernels in Bayesian optimization. They modify the kernel $k(\mathbf{x}, \mathbf{x}') = k'(w(\mathbf{x}, \mathbf{x}'))$ defining $w([x_1, \ldots, x_D]) = [w_1(x_1), \ldots, w_D(x_D]$ and

$$w_d(x_d) = \text{BetaCDF}(x_d; \alpha_d, \beta_d) \tag{6.4}$$

$$= \int_0^{x_d} \frac{\mu^{\alpha_d - 1}(1-\mu)^{\beta_d - 1}}{B(\alpha_d, \beta_d)} d\mu \tag{6.5}$$

where $B(\alpha, \beta)$ is the normalization constant for the Beta cdf function BetaCDF.

This warping function allows us to approximate linear, exponential, The authors describe placing particular priors over the hyperparameters α and β to capture prior beliefs about the type of warping to apply.

Implantation of the BetaCDF warping kernel in `Pybo` is relatively straightforward. Although the BetaCDF has no closed form solution for non-integer values [41], accurate approximations are available using the `scipy stats` python package. The gradients with respect to the input space are passed through the original gradient computations using chain rule. The BetaCDF warping function was applied in particular to the SEARD kernel. The hyperparameters α and β are treated as the other kernel hyperparameters and can be learned through optimization or sampled from hyperparameter distributions using MCMC slice sampling.

6.2 Loosening the Kernel Requirements

The second approach examined in this dissertation is to loosen or remove these requirements which restrict the class of kernels that can be applied to (I)PES

6.3 Bootstrap Sampling

Both the stationarity and the spectral sampling requirements are due to the Fourier approximate samples from our GP, discussed in chapter 3. In `Pybo`, I implemented a new approximate sampling procedure yielding analytic samples from our GP that can me optimized to produce approximate samples from $p(\mathbf{x}_\star|\mathcal{D}_n)$:

Algorithm 2 Bootstrap Sampling

```
Input: GP posterior at Bayesian optimization iteration n, $\mathcal{GP}(\mu_n, k_n|\mathcal{D}_n)$
1: Sample $m$ input space values $\mathbf{x}_i \in \mathcal{X}$.
2: Sample $m$ $y$-values from $\mathcal{N}(\mathbf{y}; \mathbf{m}, \mathbf{V})$ where $m$ and $V$ are the mean vector and Co-
   variance matrix resulting from $\mu_n, k_n$ applied to the sampled input locations.
3: Define $\mathcal{D}'_n = \mathcal{D}_n \cup \{(\mathbf{x}_i, y_i)\}_{i=1}^m$. Recondition (a copy) the GP posterior on $\mathcal{D}'_n$ yielding
   $\mathcal{GP}(\mu'_n, k'_n|\mathcal{D}'_n)$
return $\mathbf{x}_\star = \text{argmax}_{\mathbf{x}\in\mathcal{X}}\, \mu'_n(\mathbf{x})$
```

Over the input space, a uniform sampling procedure inside the convex hull defined by

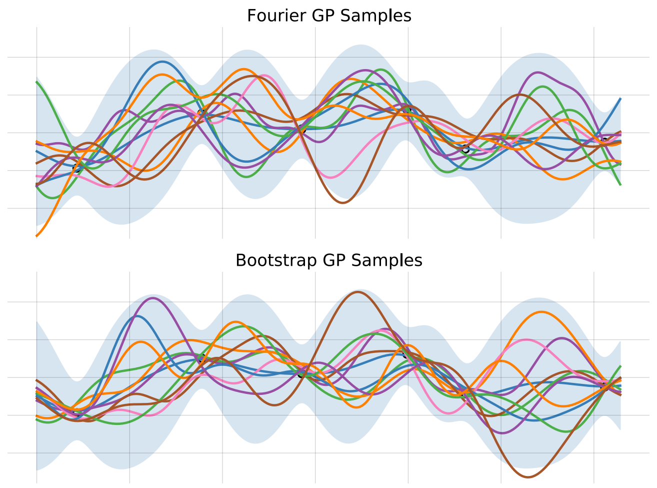


Figure 6.2: A visualization of Fourier approximate GP samples and bootstrap approximate GP samples.

the current set of observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ was implemented. Generally, for a GP with zero prior mean, sampling outside of this region will not capture information about the GP or lead to more accurate approximate samples.

Note this allows for the use of the Thompson sampling strategy with non-stationary kernels as well. A downside to this procedure is it slows down our sampling process since the we need to compute the predictive mean conditioned on each of the bootstrapped samples from our GP, a procedure which is not necessary using our Fourier approximations.

Note that although this it does require reconditioning our GP for every sample of $\mathbf{x}_\star$ generated and is much more expensive than Fourier sampling. This cost is not too expensive at early stages of Bayesian optimization and depends on the number input samples used.

6.4 Removing Gradient Conditioning

A computational and implementational bottleneck is the requirement of the *kernel* gradients. The particular gradients in condition 1. do not aid optimization but instead are used as for constraints in the approximation procedure to obtain $p(y|\mathcal{D}_n, \mathbf{x}_\star, \mathbf{x})$. Whether

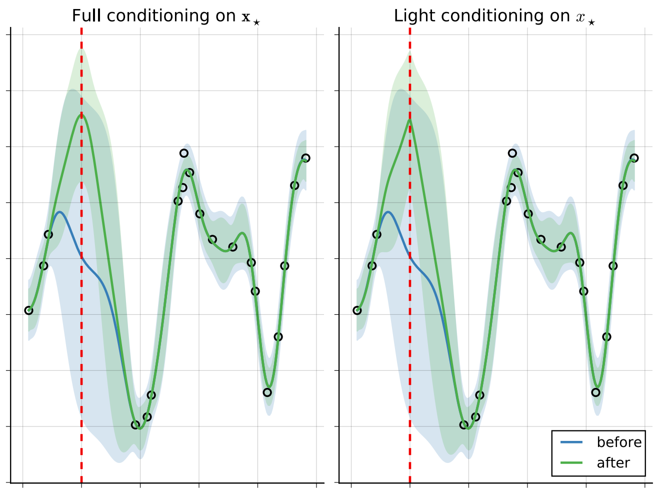


Figure 6.3: Remove the local optima condition in the approximation of $p(y|\mathcal{D}_n, \mathbf{x}_\star, \mathbf{x})$

implemented by hand or using automatic gradient packages such as `autograd`[1], computing these gradients slows down the computation of the PES and IPES acquisition functions, especially using discrete domain approximation techniques. The question is, how much do these gradients contribute to this approximation and would removing them degrade performance?

Recall from section 3.2.2 that these gradients are used to condition the predictive posterior $p(y|\mathcal{D}, \mathbf{x})$ on the $\mathbf{x}_\star$ being a local maximizer. Removing this constraint still yields a valid approximation, but relies on the GP mean and covariance to pull the function downward away from $\mathbf{x}_\star$ given the remaining conditions - that $f(\mathbf{x}_\star) \geq f(\mathbf{x}')$ where $\mathbf{x}' \in \{\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{x}\}$. Care should be taken to ensure that the interaction between this approximation and the GP used for Bayesian optimization produces the desired effective conditional $p(y|\mathcal{D}_n, \mathbf{x}_\star, \mathbf{x})$ (see figure 6.3).

[1]https://github.com/HIPS/autograd

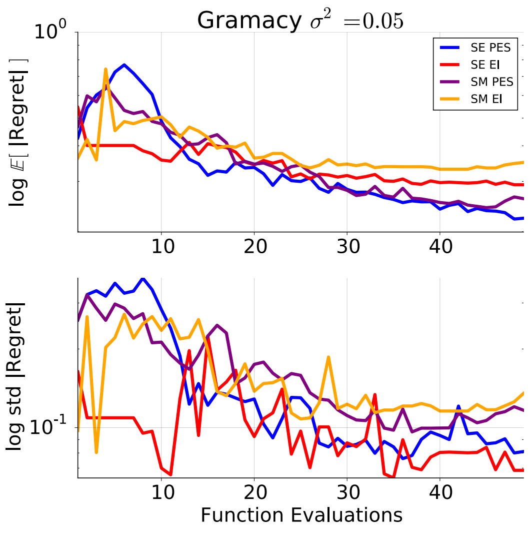


Figure 6.4: Comparison of the SM and SE kernels.

6.5 Experimental Results

6.5.1 Experiment 1: Flexible Kernels

As a comparison of the SM kernel with three components and SE(ARD) kernel, both were applied using the EI and PES acquisition functions to the Gramacy objective with likelihood noise $\sigma^2 = 0.05$. Note that due to the slow performance when using a heavily parametrized kernel, this experiment was performed using optimized point samples for hyperparameters values. In this case, with only one hyperparameter sample, IPES is identical PES. PES outperforms EI in either case but it seems as thought the SE is sufficiently powerful for this one dimensional problem and the SE kernel perhaps overly parametrized. Without a fully Bayesian treatment of the hyperparameters it is possible that the values were overfit or stuck in a local optima.

6.5.2 Experiment 2: Removing Kernel Conditions

Let us take a closer look at the question posed earlier - does removing the local optima condition in the approximation of $p(y|\mathcal{D}_n, \mathbf{x}_\star, \mathbf{x})$ degrade performance of PES or IPES. These two methods were applied using the standard approximations in PES and IPES

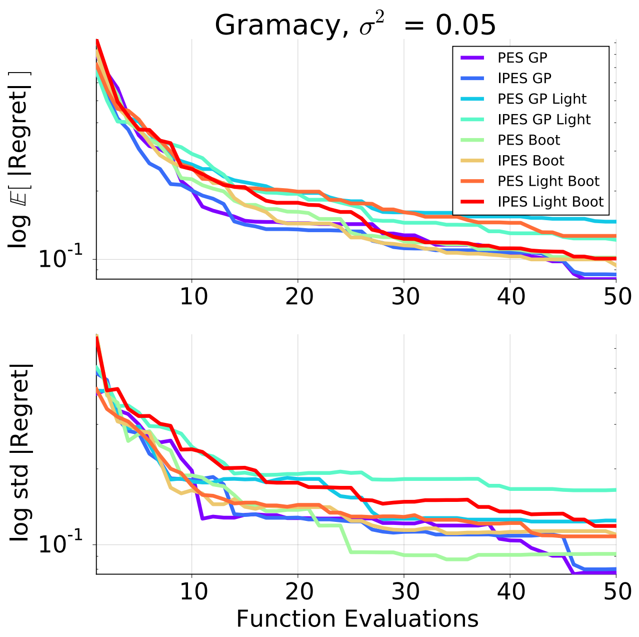


Figure 6.5: Comparison of using bootstrap approximate sampling (Boot), Light Conditioning (Light), and both on performance (Light Boot). GP is the standard implementation.

discussed in chapter 3 (GP in figure 6.5), Bootstrap sampling (Boot) discussed in 6.3, light conditioning discussed in section 6.4, and both (Light Boot). Indeed, for this example the performance of both IPES and PES suffers. The light conditioning seems to have the greatest negative effect. Interestingly, IPES seems to be more robust to these modified approximations. Although this suggests that the use of the modified approximations should be avoided if possible, they provide access to kernels that we were not able to use before. For some problems, x We will examine this further in experiment 3.

6.5.3 Experiment 3: Input Space Warping

Although it is possible to implement the kernel spectrum has to be sampled over the warped space and the the approximate Fourier must also be applied to the warped space. This is one of the rare instance where it is difficult to retain the modularity of `Pybo` as the current infrastructure would have to be modified to support this procedure. Instead, I

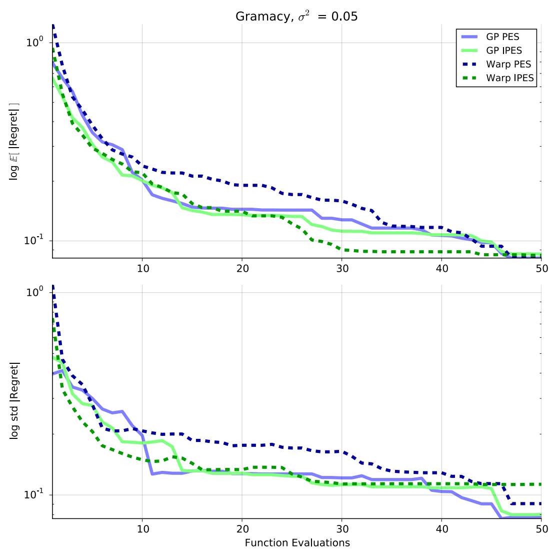


Figure 6.6: Comparison of the performance of Warped SEARD with light conditioning and bootstrap sampling and SEARD with the standard (I)PES implementation.

implemented the Warped SEARD kernel using Bootstrap sampling. For this experiment I also used light conditioning. Figure 6.6 shows a comparison of the performance of PES and IPES using these two kernels. From this figure we can draw two observations. Firstly, notice this is an example where the IPES treatment of the extra hyperparameters in the warped SEARD kernel pays of as the warped SEARD IPES method performs the best and there is a large dependency between the performance of the warped SEARD kernel using PES and using IPES. For this example it is better to use standard SEARD kernel with fewer parameter if PES is to be used. Secondly, notice that it is better to pay the potential performance loss due to the approximations examined in experiment 3 for the performance gain yielded from the use of the Warped SEARD kernel paired with the IPES acquisition function. This answers our previous question - if a GP with a particular kernel provides a much better model your objective than those that satisfy conditions 1, 2, and 3 above, it is most likely worth it to use Bootstrap sampling and weakened conditions if it we are able to then use it.

Chapter 7

Conclusion

In this dissertation, I have introduced the Integrated Predictive Entropy Search acquisition function which improves upon the Predictive Entropy Search acquisition by using the marginal predictive distributions in its computation. In this way, we estimate the reduction in entropy in $\mathbf{x}_\star$ using predictions where the uncertainty in the hyperparameters has been marginalized out - a more theoretically sound treatment of the hyperparameters.

Experiments show that the performance of PES and IPES are very similar using kernel functions with few hyperparameters on low dimensional objectives. Experiment 3 from chapter 6 demonstrated a scenario where the proper fully Bayesian treatment of the hyperparameters provided by IPES yielded better results over PES.

By examining each of the acquisition functions times in the Bayesian optimization procedure (see figure 5.1), it is clear that the IPES procedure produces a different estimate in the entropy reduction in $\mathbf{x}$ from PES. Although, with the simple examples investigated it was not clear that the IPES method was substantially better than PES - certainly not for all objectives. It would provide an excellent candidate in a portfolio of objective functions, as in [19] and [38]. Alternate approximations in the computation of PES and IPES, as well as more flexible kernels examined in chapter 6, extend the applicability of PES and IPES to more realistic objective functions that occur in real-world applications

Although experimentation has provided some positive results in favour of IPES, I believe more experimentation would be fruitful, especially experiments applying IPES to high dimensional problems where complicated kernel compositions are necessary to yield good results. In such a scenario, I suspect that information gained about the hyperparameters may more valuable. It may also prove to be a useful tool in structure discovery when such composition of kernels are applied [11]. In order to perform such experiments, however, efforts need to be made to improve the speed of the PES/IPES methods.

For Bayesian optimization applications where it is very expensive to query our objective function, we are willing to pay the extra computation cost modelling the objective and optimizing the acquisition function. This trade-off makes sense in practice but makes experimentation difficult when we have the opposite situation - cheaper simulated objectives and expensive acquisition functions. For this reason, it is worth devoting some effort towards speeding up the Bayesian optimization process. It has been shown that from our collection of current acquisition functions, no single one performs best on all applications or objectives. With a faster experimental framework, we can better navigate our current choices and outline their strengths and weaknesses. We can even work toward automating the process of recommending acquisitions functions, recommending hyperparameter priors, and Bayesian optimization in general.

References

[1] Ernesto P Adorio and U Diliman. Mvf-multivariate test functions library in c for unconstrained global optimization. *Quezon City, Metro Manila, Philippines*, 2005. (page 44)

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002. (page 10)

[3] Daniel Beck, Adrià de Gispert, Gonzalo Iglesias, Aurelien Waite, and Bill Byrne. Speed-constrained tuning for statistical machine translation using bayesian optimization. *CoRR*, abs/1604.05073, 2016. URL `http://arxiv.org/abs/1604.05073`. (page 2)

[4] Christopher M Bishop. Pattern recognition and machine learning. *Machine Learning*, 128, 2006. (page 47)

[5] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010. (page 9)

[6] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23, 2016. (page 2)

[7] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011. (page 11)

[8] Dennis D Cox and Susan John. Sdo: A statistical method for global optimization. *Multidisciplinary design optimization: state of the art*, pages 315–329, 1997. (page 10)

[9] David Duvenaud. *Automatic model construction with Gaussian processes.* PhD thesis, University of Cambridge, 2014. (pages 5 and 6)

[10] David Duvenaud. The kernel cookbook, Accessed, 2016. URL `http://www.cs.toronto.edu/~duvenaud/cookbook/index.html`. (page 6)

[11] David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1166–1174, 2013. (page 56)

[12] Zoubin Ghahramani. Lecture notes on gaussian processes, machine learning 4f13, cambridge, October 2015. (pages 6 and 19)

[13] Zoubin Ghahramani. Slides on bayesian inference, machine learning summer school, tübingen, July 2015. URL `http://webdav.tuebingen.mpg.de/mlss2013/2015/slides/ghahramani/gp-neural-nets15.pdf`. (page 5)

[14] Robert B Gramacy and Herbert KH Lee. Cases for the nugget in modeling computer experiments. *Statistics and Computing*, 22(3):713–722, 2012. (page 38)

[15] Robert B Gramacy, Genetha A Gray, Sébastien Le Digabel, Herbert KH Lee, Pritam Ranjan, Garth Wells, and Stefan M Wild. Modeling an augmented lagrangian for blackbox constrained optimization. *Technometrics*, 58(1):1–11, 2016. (page 2)

[16] Steffen Grünewälder, Jean-Yves Audibert, Manfred Opper, and John Shawe-Taylor. Regret bounds for gaussian process bandit problems. In *AISTATS*, pages 273–280, 2010. (page 7)

[17] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *The Journal of Machine Learning Research*, 13(1):1809–1837, 2012. (pages 1, 12, and 15)

[18] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems*, pages 918–926, 2014. (pages 2, 4, 12, 16, 17, 19, 20, and 21)

[19] Matthew D Hoffman, Eric Brochu, and Nando de Freitas. Portfolio allocation for bayesian optimization. In *UAI*, pages 327–336. Citeseer, 2011. (page 55)

[20] Matthew W Hoffman and Bobak Shahriari. Modular mechanisms for bayesian optimization. In *NIPS Workshop on Bayesian Optimization*. Citeseer, 2014. (page 14)

[21] Matthew William Hoffman. *Decision making with inference and learning methods*. PhD thesis, University of British Columbia, 2013. (pages 7, 9, and 10)

[22] Marco F Huber, Tim Bailey, Hugh Durrant-Whyte, and Uwe D Hanebeck. On entropy approximation for gaussian mixture random vectors. In *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, pages 181–188. IEEE, 2008. (page 28)

[23] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011. (page 4)

[24] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993. (page 7)

[25] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4): 455–492, 1998. (page 9)

[26] Harold 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. (pages 4 and 8)

[27] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985. (page 10)

[28] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. (page 7)

[29] Daniel James Lizotte. *Practical bayesian optimization.* PhD thesis, University of Alberta, 2008. (page 9)

[30] David JC MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992. (page 15)

[31] Thomas P Minka. *A family of algorithms for approximate Bayesian inference.* PhD thesis, Massachusetts Institute of Technology, 2001. (page 22)

[32] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978. (pages 9 and 11)

[33] Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012. (page 22)

[34] Iain Murray and Ryan P Adams. Slice sampling covariance hyperparameters of latent gaussian models. In *Advances in Neural Information Processing Systems*, pages 1732–1740, 2010. (page 33)

[35] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006. (page 7)

[36] Michael A Osborne, Roman Garnett, and Stephen J Roberts. Gaussian processes for global optimization. In *3rd international conference on learning and intelligent optimization (LION3)*, pages 1–15. Citeseer, 2009. (page 7)

[37] Carl E. Rasmussen and Christopher K.I. Williams. *Gaussian Processes for Machine Learning*, volume 38. The MIT Press, Cambridge, MA, USA, 2006. (pages 5, 19, and 23)

[38] Bobak Shahriari, Ziyu Wang, Matthew W Hoffman, Alexandre Bouchard-Côté, and Nando de Freitas. An entropy search portfolio for bayesian optimization. *arXiv preprint arXiv:1406.4625*, 2014. (pages 11, 14, 17, and 55)

[39] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016. (pages 2, 3, 4, 5, 7, 8, 10, and 11)

[40] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012. (pages 10, 23, and 33)

[41] Jasper Snoek, Kevin Swersky, Richard S Zemel, and Ryan P Adams. Input warping for bayesian optimization of non-stationary functions. In *ICML*, pages 1674–1682, 2014. (pages 26, 47, and 48)

[42] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md Patwary, Mostofa Ali, Ryan P Adams, et al. Scalable bayesian optimization using deep neural networks. *arXiv preprint arXiv:1502.05700*, 2015. (pages 2, 4, and 27)

[43] Ercan Solak, Roderick Murray-Smith, William E Leithead, Douglas J Leith, and Carl Edward Rasmussen. Derivative observations in gaussian process models of dynamic systems. 2003. (page 21)

[44] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009. (page 10)

[45] Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian process kernels for pattern discovery and extrapolation. In *ICML (3)*, pages 1067–1075, 2013. (pages 26 and 46)