A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Eric Brochu, Vlad M. Cora and Nando de Freitas

December 14, 2010

Abstract

We present a tutorial on Bayesian optimization, a method of finding the maximum of expensive cost functions. Bayesian optimization employs the Bayesian technique of setting a prior over the objective function and combining it with evidence to get a posterior function. This permits a utility-based selection of the next observation to make on the objective function, which must take into account both exploration (sampling from areas of high uncertainty) and exploitation (sampling areas likely to offer improvement over the current best observation). We also present two detailed extensions of Bayesian optimization, with experiments—active user modelling with preferences, and hierarchical reinforcement learning—and a discussion of the pros and cons of Bayesian optimization based on our experiences.

1 Introduction

An enormous body of scientific literature has been devoted to the problem of optimizing a nonlinear function $f(\mathbf{x})$ over a compact set $\mathcal{A}$. In the realm of optimization, this problem is formulated concisely as follows:

$$\max_{\mathbf{x}\in\mathcal{A}\subset\mathbb{R}^d} f(\mathbf{x})$$

One typically assumes that the *objective* function $f(\mathbf{x})$ has a known mathematical representation, is convex, or is at least cheap to evaluate. Despite the influence of classical optimization on machine learning, many learning problems do not conform to these strong assumptions. Often, evaluating the objective function is expensive or even impossible, and the derivatives and convexity properties are unknown.

In many realistic sequential decision making problems, for example, one can only hope to obtain an estimate of the objective function by simulating future scenarios. Whether one adopts simple Monte Carlo simulation or adaptive schemes, as proposed in the fields of planning and reinforcement learning, the process of simulation is invariably expensive. Moreover, in some applications, drawing samples $f(\mathbf{x})$ from the function corresponds to expensive processes: drug trials, destructive tests or financial investments. In active user modeling, $\mathbf{x}$ represents attributes of a user query, and $f(\mathbf{x})$ requires a response from the human. Computers must ask the right questions and the number of questions must be kept to a minimum so as to avoid annoying the user.

1.1 An Introduction to Bayesian Optimization

Bayesian optimization is a powerful strategy for finding the extrema of objective functions that are expensive to evaluate. It is applicable in situations where one does not have a closed-form expression for the objective function, but where one can obtain observations (possibly noisy) of this function at sampled values. It is particularly useful when these evaluations are costly, when one does not have access to derivatives, or when the problem at hand is non-convex.

Bayesian optimization techniques are some of the most efficient approaches in terms of the number of function evaluations required (see, e.g. [Močkus, 1994, Jones *et al.*, 1998, Streltsov and Vakili, 1999, Jones, 2001, Sasena, 2002]). Much of the efficiency stems from the ability of Bayesian optimization to incorporate prior belief about the problem to help direct the sampling, and to trade off exploration and exploitation of the search space. It is called *Bayesian* because it uses the famous "Bayes' theorem", which states (simplifying somewhat) that the *posterior* probability of a model (or theory, or hypothesis) M given evidence (or data, or observations) E is proportional to the *likelihood* of E given M multiplied by the *prior* probability of M:

$$P(M|E) \propto P(E|M)P(M).$$

Inside this simple equation is the key to optimizing the objective function. In Bayesian optimization, the *prior* represents our belief about the space of possible objective functions. Although the cost function is unknown, it is reasonable to assume that there exists prior knowledge about some of its properties, such as smoothness, and this makes some possible objective functions more plausible than others.

Let's define $\mathbf{x}_i$ as the ith sample, and $f(\mathbf{x}_i)$ as the observation of the objective function at $\mathbf{x}_i$. As we accumulate observations[1] $\mathcal{D}_{1:t} = \{\mathbf{x}_{1:t}, f(\mathbf{x}_{1:t})\}$, the prior distribution is combined with the likelihood function $P(\mathcal{D}_{1:t}|f)$. Essentially, given what we think we know about the prior, how likely is the data we have seen? If our prior belief is that the objective function is very smooth and noise-free, data with high variance or oscillations should be considered less likely than data that barely deviate from the mean. Now, we can combine these to obtain our posterior distribution:

$$P(f|\mathcal{D}_{1:t}) \propto P(\mathcal{D}_{1:t}|f)P(f).$$

[1] Here we use subscripts to denote sequences of data, i.e. $y_{1:t} = \{y_1, \ldots, y_t\}$.

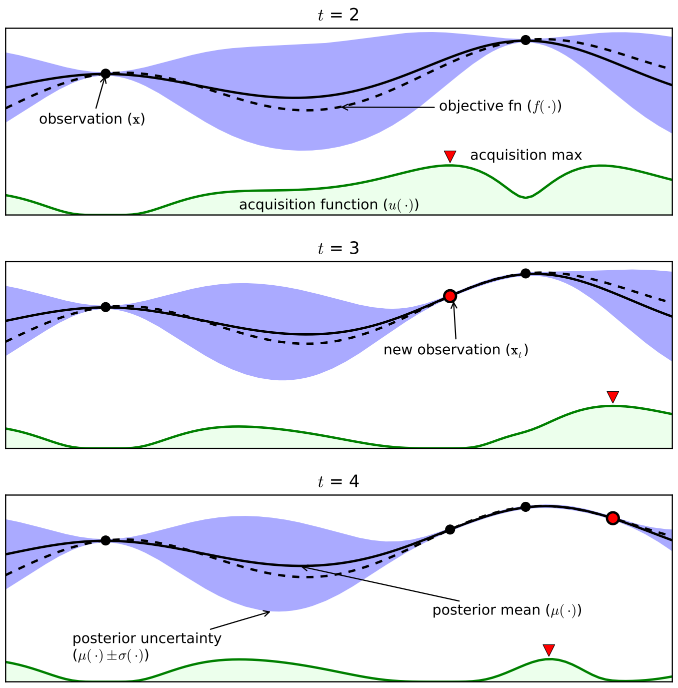


Figure 1: *An example of using Bayesian optimization on a toy 1D design problem. The figures show a Gaussian process (GP) approximation of the objective function over four iterations of sampled values of the objective function. The figure also shows the acquisition function in the lower shaded plots. The acquisition is high where the GP predicts a high objective (exploitation) and where the prediction uncertainty is high (exploration)—areas with both attributes are sampled first. Note that the area on the far left remains unsampled, as while it has high uncertainty, it is (correctly) predicted to offer little improvement over the highest observation.*

The posterior captures our updated beliefs about the unknown objective function. One may also interpret this step of Bayesian optimization as estimating the objective function with a *surrogate function* (also called a *response surface*), described formally in §2.1 with the posterior mean function of a Gaussian process.

To sample efficiently, Bayesian optimization uses an acquisition function to determine the next location $\mathbf{x}_{t+1} \in \mathcal{A}$ to sample. The decision represents an automatic trade-off between exploration (where the objective function is very uncertain) and exploitation (trying values of $\mathbf{x}$ where the objective function is expected to be high). This optimization technique has the nice property that it aims to minimize the number of objective function evaluations. Moreover, it is likely to do well even in settings where the objective function has multiple local maxima.

Figure 1 shows a typical run of Bayesian optimization on a 1D problem. The optimization starts with two points. At each iteration, the acquisition function is maximized to determine where next to sample from the objective function—the acquisition function takes into account the mean and variance of the predictions over the space to model the utility of sampling. The objective is then sampled at the argmax of the acquisition function, the Gaussian process is updated and the process is repeated. +One may also interpret this step of Bayesian optimization as estimating the objective function with a *surrogate function* (also called a *response surface*), described formally in §2.1 with the posterior mean function of a Gaussian process.

1.2 Overview

In §2, we give an overview of the Bayesian optimization approach and its history. We formally present Bayesian optimization with Gaussian process priors (§2.1) and describe covariance functions (§2.2), acquisition functions (§2.3) and the role of Gaussian noise (§2.4). In §2.5, we cover the history of Bayesian optimization, and the related fields of kriging, GP experimental design and GP active learning.

The second part of the tutorial builds on the basic Bayesian optimization model. In §3 and §4 we discuss extensions to Bayesian optimization for active user modelling in preference galleries, and hierarchical control problems, respectively. Finally, we end the tutorial with a brief discussion of the pros and cons of Bayesian optimization in §5.

2 The Bayesian Optimization Approach

Optimization is a broad and fundamental field of mathematics. In order to harness it to our ends, we need to narrow it down by defining the conditions we are concerned with.

Our first restriction is to simply specify that the form of the problem we are concerned with is *maximization*, rather than the more common form of minimization. The maximization of a real-valued function $\mathbf{x}^\star = \operatorname{argmax}_{\mathbf{x}} f(\mathbf{x})$ can be regarded as the minimization of the transformed function

$$g(\mathbf{x}) = -f(\mathbf{x}).$$

We also assume that the objective is *Lipschitz-continuous*. That is, there exists some constant C, such that for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{A}$:

$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq C\|\mathbf{x}_1 - \mathbf{x}_2\|,$$

though C may be (and typically is) unknown.

We can narrow the problem down further by defining it as one of *global*, rather than *local* optimization. In local maximization problems, we need only find a point $\mathbf{x}^{(\star)}$ such that

$$f(\mathbf{x}^{(\star)}) \geq f(\mathbf{x}), \forall \mathbf{x} \text{ s.t. } \|\mathbf{x}^{(\star)} - \mathbf{x}\| < \epsilon.$$

If $-f(\mathbf{x})$ is convex, then any local maximum is also a global maximum. However, in our optimization problems, we cannot assume that the negative objective function is convex. It *might* be the case, but we have no way of knowing before we begin optimizing.

It is common in global optimization, and true for our problem, that the objective is a *black box* function: we do not have an expression of the objective function that we can analyze, and we do not know its derivatives. Evaluating the function is restricted to querying at a point $\mathbf{x}$ and getting a (possibly noisy) response. Black box optimization also typically requires that all dimensions have bounds on the search space. In our case, we can safely make the simplifying assumption these bounds are all axis-aligned, so the search space is a hyperrectangle of dimension d.

A number of approaches exist for this kind of global optimization and have been well-studied in the literature (e.g., [Törn and Žilinskas, 1989, Mongeau *et al.*, 1998, Liberti and Maculan, 2006, Zhigljavsky and Žilinskas, 2008]). Deterministic approaches include interval optimization and branch and bound methods. *Stochastic approximation* is a popular idea for optimizing unknown objective functions in machine learning contexts [Kushner and Yin, 1997]. It is the core idea in most reinforcement learning algorithms [Bertsekas and Tsitsiklis, 1996, Sutton and Barto, 1998], learning methods for Boltzmann machines and deep belief networks [Younes, 1989, Hinton and Salakhutdinov, 2006] and parameter estimation for nonlinear state space models [Poyiadjis *et al.*, 2005, Martinez–Cantin *et al.*, 2006]. However, these are generally unsuitable for our domain because they still require many samples, and in the active user-modelling domain drawing samples is *expensive*.

Even in a noise-free domain, evaluating an objective function with Lipschitz continuity C on a d-dimensional unit hypercube, guaranteeing the best observation $f(\mathbf{x}^{+}) \geq f(\mathbf{x}^{\star}) - \epsilon$ requires $(C/2\epsilon)^d$ samples [Betrò, 1991]. This can be an incredibly expensive premium to pay for insurance against unlikely scenarios. As a result, the idea naturally arises to relax the guarantees against pathological worst-case scenarios. The goal, instead, is to use evidence and prior knowledge to maximize the posterior at each step, so that each new evaluation decreases the distance between the true global maximum and the expected maximum given the model. This is sometimes called "one-step" [Močkus, 1994] "average-case" [Streltsov and Vakili, 1999] or "practical" [Lizotte, 2008] optimization. This average-case approach has weaker demands on computation than the worst-case approach. As a result, it may provide faster solutions in many practical domains where one does not believe the worst-case scenario is plausible.

Bayesian optimization uses the prior and evidence to define a posterior distribution over the space of functions. The Bayesian model allows for an elegant means by which informative priors can describe attributes of the objective function, such as smoothness or the most likely locations of the maximum, even when the function itself is not known. Optimizing follows the principle of *maximum expected utility*, or, equivalently, *minimum expected risk*. The process of deciding where to sample next requires the choice of a utility function and a

Algorithm 1 Bayesian Optimization

```
1: for $t = 1, 2, \ldots$ do
2:     Find $\mathbf{x}_t$ by optimizing the acquisition function over the GP: $\mathbf{x}_t = \text{argmax}_{\mathbf{x}}\, u(\mathbf{x}|\mathcal{D}_{1:t-1})$.
3:     Sample the objective function: $y_t = f(\mathbf{x}_t) + \varepsilon_t$.
4:     Augment the data $\mathcal{D}_{1:t} = \{\mathcal{D}_{1:t-1}, (\mathbf{x}_t, y_t)\}$ and update the GP.
5: end for
```

way of optimizing the expectation of this utility with respect to the posterior distribution of the objective function. This secondary optimization problem is usually easier because the utility is typically chosen so that it is easy to evaluate, though still nonconvex. To make clear which function we are discussing, we will refer to this utility as the *acquisition function* (also sometimes called the *infill function*). In §2.3, we will discuss some common acquisition functions.

In practice, there is also have the possibility of measurement noise, which we will assume is Gaussian. We define $\mathbf{x}_i$ as the ith sample and $y_i = f(\mathbf{x}_i) + \varepsilon_i$, with $\varepsilon_i \overset{iid}{\sim} \mathcal{N}(0, \sigma^2_{\text{noise}})$, as the noisy observation of the objective function at $\mathbf{x}_i$. We will discuss noise in more detail in §2.4.

The Bayesian optimization procedure is shown in Algorithm 1. As mentioned earlier, it has two components: the posterior distribution over the objective and the acquisition function. Let us focus on the posterior distribution first and come back to the acquisition function in §2.3. As we accumulate observations $\mathcal{D}_{1:t} = \{\mathbf{x}_{1:t}, y_{1:t}\}$, a prior distribution $P(f)$ is combined with the likelihood function $P(\mathcal{D}_{1:t}|f)$ to produce the posterior distribution: $P(f|\mathcal{D}_{1:t}) \propto P(\mathcal{D}_{1:t}|f)P(f)$. The posterior captures the updated beliefs about the unknown objective function. One may also interpret this step of Bayesian optimization as estimating the objective function with a *surrogate function* (also called a *response surface*). In §2.1, we will discuss how Gaussian process priors can be placed on f.

2.1 Priors over functions

Any Bayesian method depends on a prior distribution, by definition. A Bayesian optimization method will converge to the optimum if (i) the acquisition function is continuous and approximately minimizes the risk (defined as the expected deviation from the global minimum at a fixed point $\mathbf{x}$); and (ii) conditional variance converges to zero (or appropriate positive minimum value in the presence of noise) if and only if the distance to the nearest observation is zero [Močkus, 1982, Močkus, 1994]. Many models could be used for this prior—early work mostly used the Wiener process (§2.5). However, Gaussian process (GP) priors for Bayesian optimization date back at least to the late 1970s [O'Hagan, 1978, Žilinskas, 1980]. Močkus [1994] explicitly set the framework for the Gaussian process prior by specifying the additional "simple and natural" conditions that (iii) the objective is continuous; (iv) the prior is homogeneous; (v) the optimization is independent of the m^{th} differences. This

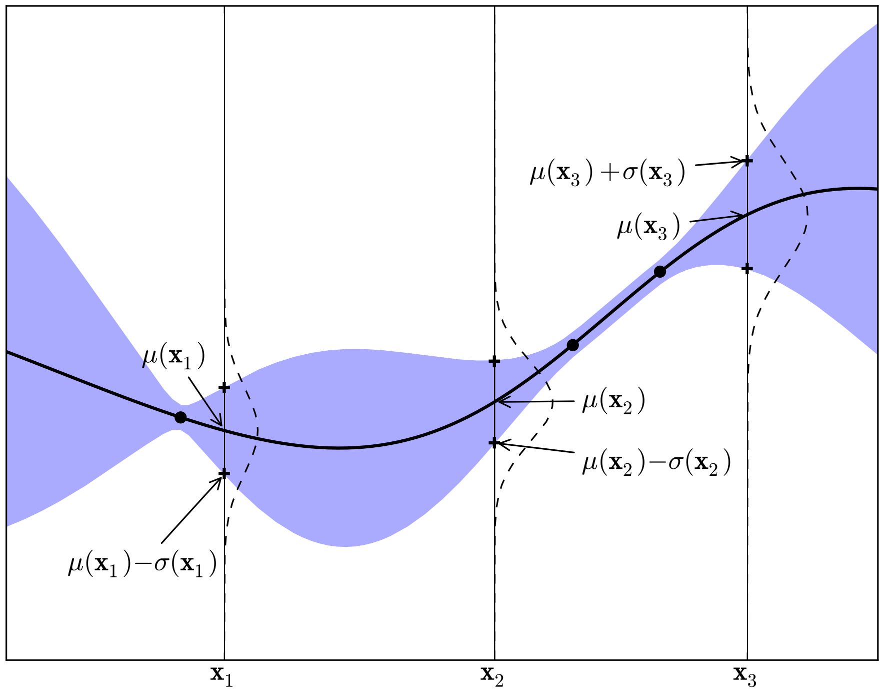


Figure 2: *Simple 1D Gaussian process with three observations. The solid black line is the GP surrogate mean prediction of the objective function given the data, and the shaded area shows the mean plus and minus the variance. The superimposed Gaussians correspond to the GP mean and standard deviation ($\mu(\cdot)$ and $\sigma(\cdot)$) of prediction at the points,* $\mathbf{x}_{1:3}$.

includes a very large family of common optimization tasks, and Močkus showed that the GP prior is well-suited to the task.

A GP is an extension of the multivariate Gaussian distribution to an infinite-dimension stochastic process for which any finite combination of dimensions will be a Gaussian distribution. Just as a Gaussian distribution is a distribution over a random variable, completely specified by its mean and covariance, a GP is a distribution over functions, completely specified by its mean function, m and covariance function, k:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')).$$

It is often useful to intuitively think of a GP as analogous to a function, but instead of returning a scalar $f(\mathbf{x})$ for an arbitrary $\mathbf{x}$, it returns the mean and variance of a normal distribution (Figure 2) over the possible values of f at $\mathbf{x}$. Stochastic processes are sometimes called “random functions”, by analogy to random variables.

For convenience, we assume here that the prior mean is the zero function $m(\mathbf{x}) = 0$; alternative priors for the mean can be found in, for example [Martinez–Cantin *et al.*, 2009, Brochu *et al.*, 2010a]. This leaves us the more interesting question of defining the covariance function k. A very popular choice is the squared exponential function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right). \tag{1}$$

Note that this function approaches 1 as values get close together and 0 as they get further apart. Two points that are close together can be expected to have a very large influence on each other, whereas distant points have almost none. This is a necessary condition for convergence under the assumptions of [Močkus, 1994]. We will discuss more sophisticated kernels in §2.2.

If we were to sample from the prior, we would choose $\{\mathbf{x}_{1:t}\}$ and sample the values of the function at these indices to produce the pairs $\{\mathbf{x}_{1:t}, \mathbf{f}_{1:t}\}$, where $\mathbf{f}_{1:t} = f(\mathbf{x}_{1:t})$. The function values are drawn according to a multivariate normal distribution $\mathcal{N}(0, \mathbf{K})$, where the kernel matrix is given by:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \ldots & k(\mathbf{x}_1, \mathbf{x}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_t, \mathbf{x}_1) & \ldots & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix}.$$

Of course, the diagonal values of this matrix are 1 (each point is perfectly correlated with itself), which is only possible in a noise-free environment. We will discuss noise in §2.4. Also, recall that we have for simplicity chosen the zero mean function.

In our optimization tasks, however, we will use data from an external model to fit the GP and get the posterior. Assume that we already have the observations $\{\mathbf{x}_{1:t}, \mathbf{f}_{1:t}\}$, say from previous iterations, and that we want to use Bayesian optimization to decide what point $\mathbf{x}_{t+1}$ should be considered next. Let us denote the value of the function at this arbitrary point as $f_{t+1} = f(\mathbf{x}_{t+1})$. Then, by the properties of Gaussian processes, $\mathbf{f}_{1:t}$ and f_{t+1} are jointly Gaussian:

$$\begin{bmatrix} \mathbf{f}_{1:t} \\ f_{t+1} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix}\right),$$

where

$$\mathbf{k} = \begin{bmatrix} k(\mathbf{x}_{t+1}, \mathbf{x}_1) & k(\mathbf{x}_{t+1}, \mathbf{x}_2) & \cdots & k(\mathbf{x}_{t+1}, \mathbf{x}_t) \end{bmatrix}$$

Using the Sherman-Morrison-Woodbury formula (see, e.g., [Rasmussen and Williams, 2006, Press *et al.*, 2007]), one can easily arrive at an expression for the predictive distribution:

$$P(f_{t+1}|\mathcal{D}_{1:t}, \mathbf{x}_{t+1}) = \mathcal{N}\left(\mu_t(\mathbf{x}_{t+1}), \sigma_t^2(\mathbf{x}_{t+1})\right)$$

where

$$\begin{aligned} \mu_t(\mathbf{x}_{t+1}) &= \mathbf{k}^T\mathbf{K}^{-1}\mathbf{f}_{1:t} \\ \sigma_t^2(\mathbf{x}_{t+1}) &= k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{k}. \end{aligned}$$

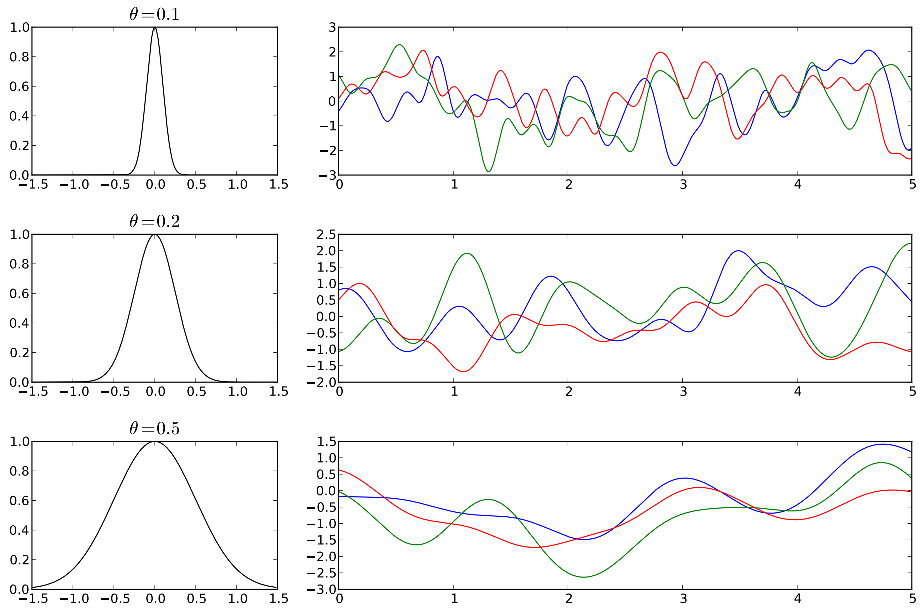


Figure 3: *The effect of changing the kernel hyperparameters. Shown are squared exponential kernels with* $\theta = 0.1, 0.2, 0.5$. *On the left is the function* $k(0, \mathbf{x})$. *On the right are some one-dimensional functions sampled from a GP with the hyperparameter value.*

That is, $\mu_t(\cdot)$ and $\sigma_t^2(\cdot)$ are the sufficient statistics of the predictive posterior distribution $P(f_{t+1}|\mathcal{D}_{1:t}, \mathbf{x}_{t+1})$. For legibility, we will omit the subscripts on μ and σ except where it might be unclear. In the sequential decision making setting, the number of query points is relatively small and, consequently, the GP predictions are easy to compute.

2.2 Choice of covariance functions

The choice of covariance function for the Gaussian Process is crucial, as it determines the smoothness properties of samples drawn from it. The squared exponential kernel in Eqn (1) is actually a little naive, in that divergences of all features of $\mathbf{x}$ affect the covariance equally.

Typically, it is necessary to generalize by adding *hyperparameters*. In an isotropic model, this can be done with a single hyperparameter θ, which controls the width of the kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2\theta^2}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right).$$

For anisotropic models, a very popular choice is the squared exponential kernel with a vector of automatic relevance determination (ARD) hyperparameters $\boldsymbol{\theta}$

[Rasmussen and Williams, 2006, page 106]:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\big(-\tfrac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \operatorname{diag}(\boldsymbol{\theta})^{-2}(\mathbf{x} - \mathbf{x}')\big),$$

where $\operatorname{diag}(\boldsymbol{\theta})$ is a diagonal matrix with d entries $\boldsymbol{\theta}$ along the diagonal. Intuitively, if a particular θ_ℓ has a small value, the kernel becomes independent of ℓ-th input, effectively removing it automatically. Hence, irrelevant dimensions are discarded. Figure 3 shows examples of different hyperparameter values on the squared exponential function and what functions sampled from those values look like. Typically, the hyperparameter values are learned by "seeding" with a few random samples and maximizing the log-likelihood of the evidence given $\boldsymbol{\theta}$ [Jones *et al.*, 1998, Sasena, 2002, Santner *et al.*, 2003, Rasmussen and Williams, 2006]. This can often be aided with an informative hyperprior on the hyperparameters, often a log normal prior [Lizotte, 2008, Frean and Boyle, 2008]. Methods of learning these values more efficiently is currently an active subfield of research (*e.g.* [Osborne, 2010, Brochu *et al.*, 2010a]).

Another important kernel for Bayesian optimization is the Matérn kernel [Matérn, 1960, Stein, 1999], which incorporates a smoothness parameter ς to permit greater flexibility in modelling functions:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2^{\varsigma-1}\Gamma(\varsigma)}\left(2\sqrt{\varsigma}\,\|\mathbf{x}_i - \mathbf{x}_j\|\right)^{\varsigma} H_\varsigma\left(2\sqrt{\varsigma}\,\|\mathbf{x}_i - \mathbf{x}_j\|\right),$$

where $\Gamma(\cdot)$ and $H_\varsigma(\cdot)$ are the Gamma function and the Bessel function of order ς. Note that as $\varsigma \to \infty$, the Matérn kernel reduces to the squared exponential kernel, and when $\varsigma = 0.5$, it reduces to the unsquared exponential kernel. As with the squared exponential, length-scale hyperparameter are often incorporated.

While the squared exponential and Matérn are the most common kernels for GPs, numerous others have been examined in the machine learning literature (see, e.g., [Genton, 2001] or [Rasmussen and Williams, 2006, Chapter 4] for an overview). Appropriate covariance functions can also be used to extend the model in other interesting ways. For example, the recent sequential sensor work of Osborne, Garnett and colleagues uses GP models with extensions to the covariance function to model the characteristics of changepoints [Osborne *et al.*, 2010] and the locations of sensors in a network [Garnett *et al.*, 2010a]. A common additional hyperparameter is simply a scalar applied to k to control the magnitude of the variance.

Determining which of a set of possible kernel functions to use for a problem typically requires a combination of engineering and automatic model selection, either hierarchical Bayesian model selection [Mackay, 1992] or cross-validation. However, these methods require fitting a model given a representative sample of data. In [Brochu *et al.*, 2010a], we discuss how model selection can be performed using models believed to be similar. The techniques introduced in [Brochu *et al.*, 2010b] could also be applied to model selection, though that is outside the scope of this tutorial.

2.3 Acquisition Functions for Bayesian Optimization

Now that we have discussed placing priors over smooth functions and how to update these priors in light of new observations, we will focus our attention on the acquisition component of Bayesian optimization. The role of the acquisition function is to guide the search for the optimum. Typically, acquisition functions are defined such that high acquisition corresponds to *potentially* high values of the objective function, whether because the prediction is high, the uncertainty is great, or both. Maximizing the acquisition function is used to select the next point at which to evaluate the function. That is, we wish to sample f at $\operatorname{argmax}_{\mathbf{x}} u(\mathbf{x}|\mathcal{D})$, where $u(\cdot)$ is the generic symbol for an acquisition function.

2.3.1 Improvement-based acquisition functions

The early work of Kushner [1964] suggested maximizing the *probability of improvement* over the incumbent $f(\mathbf{x}^+)$, where $\mathbf{x}^+ = \operatorname{argmax}_{\mathbf{x}_i \in \mathbf{x}_{1:t}} f(\mathbf{x}_i)$, so that

$$\begin{aligned} \mathrm{PI}(\mathbf{x}) &= P(f(\mathbf{x}) \geq f(\mathbf{x}^+)) \\ &= \Phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma(\mathbf{x})}\right), \end{aligned}$$

where $\Phi(\cdot)$ is the normal cumulative distribution function. This function is also sometimes called *MPI* (for "maximum probability of improvement") or "the P-algorithm" (since the utility is the probability of improvement).

The drawback, intuitively, is that this formulation is pure exploitation. Points that have a high probability of being infinitesimally greater than $f(\mathbf{x}^+)$ will be drawn over points that offer larger gains but less certainty. As a result, a modification is to add a trade-off parameter $\xi \geq 0$:

$$\begin{aligned} \mathrm{PI}(\mathbf{x}) &= P(f(\mathbf{x}) \geq f(\mathbf{x}^+) + \xi) \\ &= \Phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})}\right), \end{aligned} \tag{2}$$

The exact choice of ξ is left to the user, though Kushner recommended a schedule for ξ, so that it started fairly high early in the optimization, to drive exploration, and decreased toward zero as the algorithm continued. Several researchers have studied the empirical impact of different values of ξ in different domains [Törn and Žilinskas, 1989, Jones, 2001, Lizotte, 2008].

An appealing characteristic of this formulation for perceptual and preference models is that while maximizing $\mathrm{PI}(\cdot)$ is still greedy, it selects the point most likely to offer an improvement of at least ξ. This can be useful in psychoperceptual tasks, where there is a threshold of perceptual difference.

Jones [2001] notes that the performance of $\mathrm{PI}(\cdot)$

> "is truly impressive. It would be quite natural if the reader, like so many others, became enthusiastic about this approach. But if there is a single lesson to be taken away from this paper, it is that

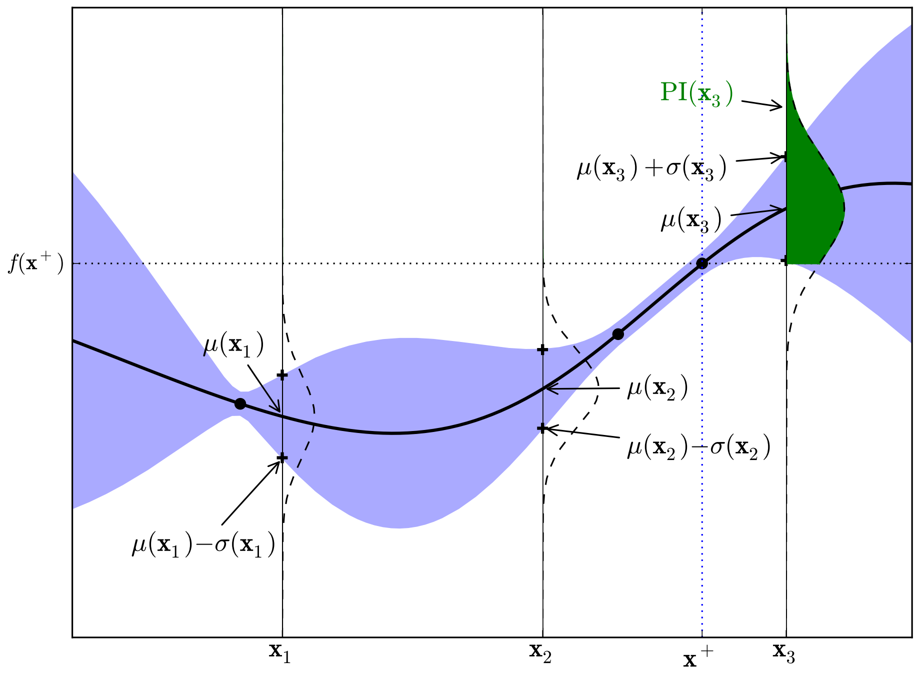


Figure 4: *Gaussian process from Figure 2, additionally showing the region of probable improvement. The maximum observation is at* $\mathbf{x}^+$*. The darkly-shaded area in the superimposed Gaussian above the dashed line can be used as a measure of improvement,* $I(\mathbf{x})$*. The model predicts almost no possibility of improvement by observing at* $\mathbf{x}_1$ *or* $\mathbf{x}_2$*, while sampling at* $\mathbf{x}_3$ *is more likely to improve on* $f(\mathbf{x}^+)$*.*

> nothing in this response-surface area is so simple. There always seems to be a counterexample. In this case, the difficulty is that [the PI($\cdot$) method] is extremely sensitive to the choice of the target. If the desired improvement is too small, the search will be highly local and will only move on to search globally after searching nearly exhaustively around the current best point. On the other hand, if [ξ] is set too high, the search will be excessively global, and the algorithm will be slow to fine-tune any promising solutions."

A somewhat more satisfying alternative acquisition function would be one that takes into account not only the probability of improvement, but the magnitude of the improvement a point can potentially yield. In particular, we want to minimize the expected deviation from the true maximum $f(\mathbf{x}^\star)$, when choosing a new trial point:

$$\begin{aligned}\mathbf{x}_{t+1} &= \operatorname*{argmin}_{\mathbf{x}} \mathbb{E}(\|f_{t+1}(\mathbf{x}) - f(\mathbf{x}^\star)\| \,|\mathcal{D}_{1:t}) \\ &= \operatorname*{argmin}_{\mathbf{x}} \int \|f_{t+1}(\mathbf{x}) - f(\mathbf{x}^\star)\| P(f_{t+1}|\mathcal{D}_{1:t}) df_{t+1},\end{aligned}$$

Note that this decision process is myopic in that it only considers one-step-ahead choices. However, if we want to plan two steps ahead, we can easily apply recursion:

$$\mathbf{x}_{t+1} = \operatorname*{argmin}_{\mathbf{x}} \mathbb{E}\left(\min_{\mathbf{x}'} \mathbb{E}(\|f_{t+2}(\mathbf{x}') - f(\mathbf{x}^\star)\| \,|\mathcal{D}_{t+1}) \,|\mathcal{D}_{1:t}\right)$$

One could continue applying this procedure of dynamic programming for as many steps ahead as desired. However, because of its expense, Močkus *et al.* [1978] proposed the alternative of maximizing the expected improvement with respect to $f(\mathbf{x}^+)$. Specifically, Močkus defined the improvement function as:

$$\mathrm{I}(\mathbf{x}) = \max\{0, f_{t+1}(\mathbf{x}) - f(\mathbf{x}^+)\}.$$

That is, $\mathrm{I}(\mathbf{x})$ is positive when the prediction is higher than the best value known thus far. Otherwise, $\mathrm{I}(\mathbf{x})$ is set to zero. The new query point is found by maximizing the expected improvement:

$$\mathbf{x} = \operatorname*{argmax}_{\mathbf{x}} \mathbb{E}(\max\{0, f_{t+1}(\mathbf{x}) - f(\mathbf{x}^+)\} \,|\mathcal{D}_t)$$

The likelihood of improvement I on a normal posterior distribution characterized by $\mu(\mathbf{x}), \sigma^2(\mathbf{x})$ can be computed from the normal density function,

$$\frac{1}{\sqrt{2\pi}\sigma(\mathbf{x})} \exp\left(-\frac{(\mu(\mathbf{x}) - f(\mathbf{x}^+) - \mathrm{I})^2}{2\sigma^2(\mathbf{x})}\right).$$

The expected improvement is the integral over this function:

$$\begin{aligned}\mathbb{E}(\mathrm{I}) &= \int_{\mathrm{I}=0}^{\mathrm{I}=\infty} \mathrm{I}\, \frac{1}{\sqrt{2\pi}\sigma(\mathbf{x})} \exp\left(-\frac{(\mu(\mathbf{x}) - f(\mathbf{x}^+) - \mathrm{I})^2}{2\sigma^2(\mathbf{x})}\right) d\,\mathrm{I} \\ &= \sigma(\mathbf{x}) \left[\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma(\mathbf{x})} \Phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma(\mathbf{x})}\right) + \phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma(\mathbf{x})}\right)\right]\end{aligned}$$

The expected improvement can be evaluated analytically [Močkus *et al.*, 1978, Jones *et al.*, 1998], yielding:

$$\begin{aligned}\mathrm{EI}(\mathbf{x}) &= \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+))\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \qquad (3) \\ Z &= \frac{\mu(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma(\mathbf{x})}\end{aligned}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ denote the PDF and CDF of the standard normal distribution respectively. Figure 4 illustrates a typical expected improvement scenario.

It should be said that being myopic is not a requirement here. For example, it is possible to derive analytical expressions for the two-step ahead expected improvement [Ginsbourger *et al.*, 2008] and multistep Bayesian optimization [Garnett *et al.*, 2010b]. This is indeed a very promising recent direction.

2.3.2 Exploration-exploitation trade-off

The expectation of the improvement function with respect to the predictive distribution of the Gaussian process enables us to balance the trade-off of exploiting and exploring. When exploring, we should choose points where the surrogate variance is large. When exploiting, we should choose points where the surrogate mean is high.

It is highly desirable for our purposes to express $\mathrm{EI}(\cdot)$ in a generalized form which controls the trade-off between global search and local optimization (exploration/exploitation). Lizotte [2008] suggests a $\xi \geq 0$ parameter such that:

$$\mathrm{EI}(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases}, \tag{4}$$

where

$$Z = \begin{cases} \frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})} & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases}.$$

This ξ is very similar in flavour to the ξ used in Eqn (2), and to the approach used by Jones *et al.* [2001]. Lizotte's experiments suggest that setting $\xi = 0.01$ (scaled by the signal variance if necessary) works well in almost all cases, and interestingly, setting a cooling schedule for ξ to encourage exploration early and exploitation later does *not* work well empirically, contrary to intuition (though Lizotte did find that a cooling schedule for ξ might slightly improve performance on short runs ($t < 30$) of PI optimization).

2.3.3 Confidence bound criteria

Cox and John [1992, 1997] introduce an algorithm they call "Sequential Design for Optimization", or *SDO*. Given a random function model, SDO selects points for evaluation based on the *lower confidence bound* of the prediction site:

$$\mathrm{LCB}(\mathbf{x}) = \mu(\mathbf{x}) - \kappa\sigma(x),$$

where $\kappa \geq 0$. While they are concerned with minimization, we can maximize by instead defining the upper confidence bound:

$$\mathrm{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(x).$$

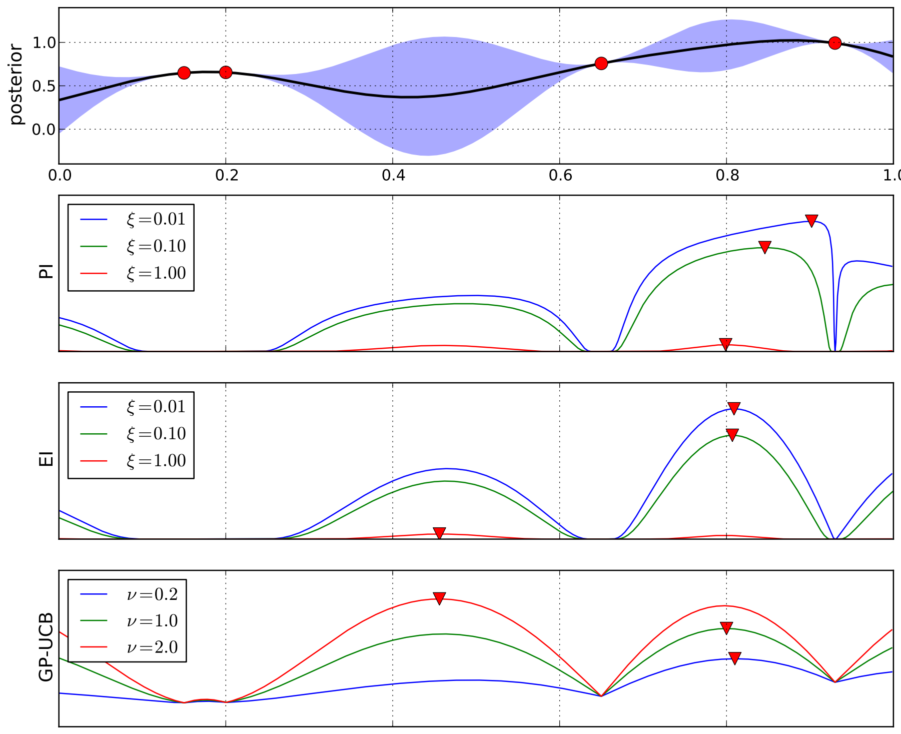


Figure 5: *Examples of acquisition functions and their settings. The GP posterior is shown at top. The other images show the acquisition functions for that GP. From the top: probability of improvement (Eqn (2)), expected improvement (Eqn (4)) and upper confidence bound (Eqn (5)). The maximum of each function is shown with a triangle marker.*

Like other parameterized acquisition models we have seen, the parameter κ is left to the user. However, an alternative acquisition function has been proposed by Srinivas *et al.* [2010]. Casting the Bayesian optimization problem as a multi-armed bandit, the acquisition is the instantaneous regret function

$$r(\mathbf{x}) = f(\mathbf{x}^\star) - f(\mathbf{x}).$$

The goal of optimizing in the framework is to find:

$$\min \sum_t^T r(\mathbf{x}_t) = \max \sum_t^T f(\mathbf{x}_t),$$

where T is the number of iterations the optimization is to be run for.

Using the *upper confidence bound* selection criterion with $\kappa_t = \sqrt{\nu \tau_t}$ and the hyperparameter $\nu > 0$ Srinivas *et al.* define

$$\text{GP-UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \sqrt{\nu \tau_t}\sigma(\mathbf{x}). \tag{5}$$

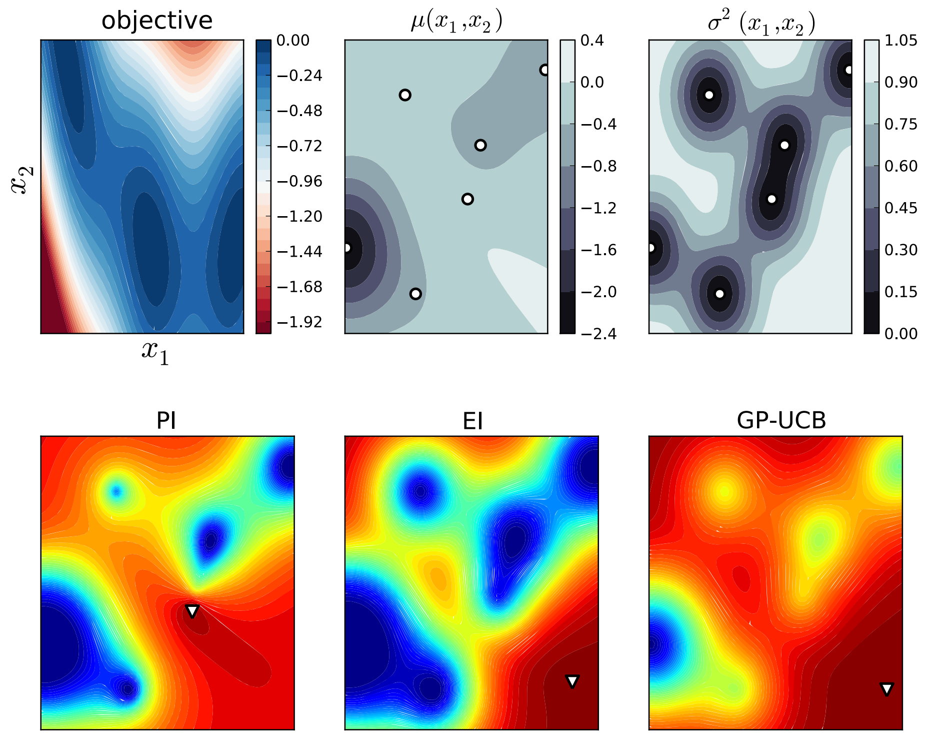


Figure 6: *Examples of acquisition functions and their settings in 2 dimensions. The top row shows the objective function (which is the Branin function here), and the posterior mean and variance estimates $\mu(\cdot)$ and $\sigma^2(\cdot)$. The samples used to train the GP are shows with white dots. The second row shows the acquisition functions for the GP. From left to right: probability of improvement (Eqn (2)), expected improvement (Eqn (4)) and upper confidence bound (Eqn (5)). The maximum of each function is shown with a triangle marker.*

With $\nu = 1$ and $\tau_t = 2\log(t^{d/2+2}\pi^2/3\delta)$, it can be shown[2] with high probability that this method is *no regret*, i.e. $\lim_{T\to\infty} R_T/T = 0$, where R_T is the cumulative regret

$$R_T = \sum_{t=1}^{T} f(\mathbf{x}^\star) - f(\mathbf{x}_t).$$

This in turn implies a lower-bound on the convergence rate for the optimization problem.

Figures 5 and 6 show how with the same GP posterior, different acquisition functions with different maxima are defined. Figure 7 gives an example of how

[2] These bounds hold for *reasonably smooth* kernel functions, where the exact formulation of the bounds depends upon the form of kernel used. We refer the interested reader to the original paper [Srinivas *et al.*, 2010].

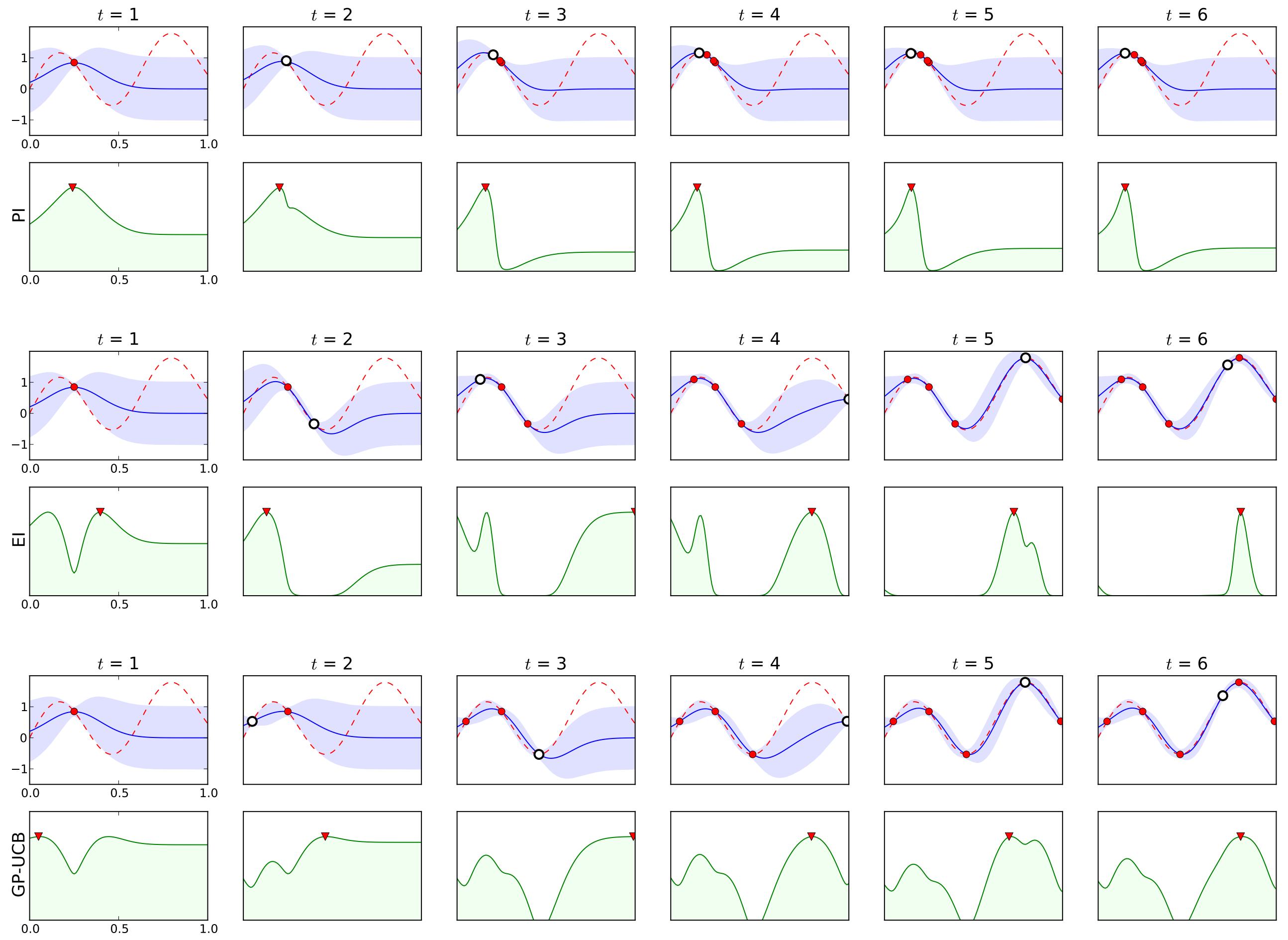


Figure 7: *Comparison of probability of improvement (top), expected improvement (middle) and upper confidence bound (bottom) acquisition functions on a toy 1D problem. In the upper rows, the objective function is shown with the dotted red line, the solid blue line is the GP posterior mean. In the lower rows, the respective infill functions are shown, with a star denoting the maximum. The optimizations are initialized with the same two points, but quickly follow different sampling trajectories. In particular, note that the greedy* EI *algorithm ignores the region around* $x = 0.4$ *once it is determined there is minimal chance of improvement, while* GP-UCB *continues to explore.*

PI, EI and GP-UCB give rise to distinct sampling behaviour over time.

With several different parameterized acquisition functions in the literature, it is often unclear which one to use. Brochu *et al.* [2010b] present one method of utility selection. Instead of using a single acquisition function, they adopt a portfolio of acquisition functions governed by an online multi-armed bandit strategy, which almost always outperforms the best individual acquisition function of a suite of standard test problems.

2.3.4 Maximizing the acquisition function

To find the point at which to sample, we still need to maximize the constrained objective $u(\mathbf{x})$. *Unlike the original unknown objective function,* $u(\cdot)$ *can be cheaply sampled.* We optimize the acquisition function using *DIRECT* [Jones *et al.*, 1993], a deterministic, derivative-free optimizer. It uses the existing samples of the objective function to decide how to proceed to DIvide the feasible space into finer RECTangles. A particular advantage in active learning applications is that DIRECT can be implemented as an "any-time" algorithm, so that as long as the user is doing something else, it continues to optimize, and when interrupted, the program can use the best results found to that point in time. Methods such as Monte Carlo and multistart have also been used, and seem to perform reasonably well [Močkus, 1994, Lizotte, 2008].

2.4 Noise

The model we've used so far assumes that we have perfectly noise-free observations. In real life, this is rarely possible, and instead of observing $f(\mathbf{x})$, we can often only observe a noisy transformation of $f(\mathbf{x})$.

The simplest transformation arises when $f(\mathbf{x})$ is corrupted with Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2_{\text{noise}})$ [Rasmussen and Williams, 2006]. If the noise is additive, we can easily add the noise distribution to the Gaussian distribution $\mathcal{N}(0, \mathbf{K})$ and define

$$y_i = f(\mathbf{x}_i) + \epsilon_i.$$

Since the mean is zero, this type of noise simply requires that we replace the kernel $\mathbf{K}$ with the following kernel for the noisy observations of $f(\cdot)$:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_t, \mathbf{x}_1) & \dots & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix} + \sigma^2_{\text{noise}} I \tag{6}$$

This yields the predictive distribution:

$$P(y_{t+1} | \mathcal{D}_{1:t}, \mathbf{x}_{t+1}) = \mathcal{N}(\mu_t(\mathbf{x}_{t+1}), \sigma^2_t(\mathbf{x}_{t+1}) + \sigma^2_{\text{noise}}),$$

and the sufficient statistics

$$\begin{aligned} \mu_t(\mathbf{x}_{t+1}) &= \mathbf{k}^T [\mathbf{K} + \sigma^2_{\text{noise}} I]^{-1} \mathbf{y}_{1:t} \\ \sigma^2_t(\mathbf{x}_{t+1}) &= k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}^T [\mathbf{K} + \sigma^2_{\text{noise}} I]^{-1} \mathbf{k}. \end{aligned}$$

In a noisy environment, we also change the definition of the incumbent in the PI and EI acquisition functions. Instead of using the best observation, we use the distribution at the sample points, and define as the incumbent, the point with the highest expected value,

$$\mu^+ = \operatorname*{argmax}_{\mathbf{x}_i \in \mathbf{x}_{1:t}} \mu(\mathbf{x}_i).$$

This avoids the problem of attempting to maximize probability or expected improvement over an unreliable sample. It is also possible to resample potential incumbents to get more reliable estimates of the values in a noisy environment [Bartz-Beielstein *et al.*, 2005, Huang *et al.*, 2006, Hutter *et al.*, 2009], a process sometimes called *intensification.* Nonstationary noise models are also possible, such as autoregressive moving-average noise [Murray-Smith and Girard, 2001] and heteroskedastic Gaussian noise [Goldberg *et al.*, 1998].

2.5 A brief history of Bayesian optimization

The earliest work we are aware of resembling the modern Bayesian optimization approach is the early work of Kushner [1964], who used Wiener processes for unconstrained one-dimensional problems. Kushner's decision model was based on maximizing the probability of improvement (§2.3.1). He also included a parameter that controlled the trade-off between 'more global' and 'more local' optimization, in the same spirit as the exploration-exploitation trade-off. A key difference is that in a (one-dimensional) Wiener process, the intervals between samples are independent, and Kushner was concerned with the problem of selecting from a finite set of intervals. Later work extended Kushner's technique to multidimensional optimization, using, for example, interpolation in a Delauney triangulation of the space [Elder, 1992] or projecting Wiener processes between sample points [Stuckman, 1988].

Meanwhile, in the former Soviet Union, Močkus and colleagues developed a multidimensional Bayesian optimization method using linear combinations of Wiener fields. This was first published in English as [Močkus *et al.*, 1978]. This paper also, significantly, describes an acquisition function that is based on myopic expected improvement of the posterior, which has been widely adopted in Bayesian optimization as the expected improvement function (§2.3.1). A more recent review of Močkus' approach is [Močkus, 1994].

At the same time, a large, related body of work emerged under the name *kriging* (§2.6), in honour of the South African student who developed this technique at the University of the Witwatersrand [Krige, 1951], though largely popularized by Matheron and colleagues (e.g. [Matheron, 1971]). In kriging, the goal is interpolation of a random field via a linear predictor. The errors on this model are typically assumed to *not* be independent, and are modelled with a Gaussian process.

More recently, Bayesian optimization using Gaussian processes has been successfully applied to derivative-free optimization and experimental design, where it is called Efficient Global Optimization, or *EGO* (§2.7).

There exist several consistency proofs for this algorithm in the one-dimensional setting [Locatelli, 1997] and one for a simplification of the algorithm using simplicial partitioning in higher dimensions [Žilinskas and Žilinskas, 2002]. The convergence of the algorithm using multivariate Gaussian processes has been recently established in [Vasquez and Bect, 2008].

2.6 Kriging

Kriging has been used in geostatistics and environmental science since the 1950s and remains important today. We will briefly summarize the connection to Bayesian optimization here. More detailed examinations can be found in, for example, [Stein, 1999, Sasena, 2002, Diggle and Ribeiro, 2007]. This section is primarily drawn from these sources.

In many modelling techniques in statistics and machine learning, it is assumed that samples drawn from a process with independent, identically distributed residuals, typically, $\varepsilon \sim \mathcal{N}(0, \sigma^2_{\text{noise}})$:

$$y(\mathbf{x}) = f(\mathbf{x}) + \varepsilon$$

In kriging, however, the usual assumption is that errors are *not* independent, and are, in fact, spatially correlated: where errors are high, it is expected that nearby errors will also be high. Kriging is a combination of a linear regression model and a stochastic model fitted to the residual errors of the linear model. The residual is modelled with a zero-mean Gaussian process, so ε is actually parameterized by $\mathbf{x}$: $\varepsilon(\mathbf{x}) \sim \mathcal{N}(0, \sigma^2(\mathbf{x}))$.

The actual regression model depends on the type of kriging. In *simple kriging*, f is modelled with the zero function, making it a zero-mean GP model. In *ordinary kriging*, f is modelled with a constant but unknown function. *Universal kriging* models f with a polynomial of degree k with bases m and coefficients β, so that

$$y(\mathbf{x}) = \sum_{j=1}^{k} \beta_j m_j(\mathbf{x}) + \varepsilon(\mathbf{x}).$$

Other, more exotic types of kriging are also used.

Clearly, kriging and Bayesian optimization are very closely related. There are some key differences in practice, though. In Bayesian optimization, models are usually fit through maximum likelihood. In kriging, models are usually fit using a *variogram*, a measure of the average dissimilarity between samples versus their separation distance. Fitting is done using least squares or similar numerical methods, or interactively, by an expert visually inspecting the variogram plot with specially-designed software. Kriging also often restricts the prediction model to use only a small number of neighbours, making it fit locally while ignoring global information. Bayesian optimization normally uses all the data in order to learn a global model.

2.7 Experimental design

Kriging has been applied to experimental design under the name *DACE*, after "Design and Analysis of Computer Experiments", the title of a paper by Sacks *et al.* [1989] (and more recently a book by Santner *et al.* [2003]). In DACE, the regression model is a best linear unbiased predictor (BLUP), and the residual model is a noise-free Gaussian process. The goal is to find a design point or points that optimizes some criterion.

The "efficient global optimization", or *EGO*, algorithm is the combination of DACE model with the sequential expected improvement (§2.3.1) acquisition criterion. It was published in a paper by Jones *et al.* [1998] as a refinement of the *SPACE* algorithm (Stochastic Process Analysis of Computer Experiments) [Schonlau, 1997]. Since EGO's publication, there has evolved a body of work devoted to extending the algorithm, particularly in adding constraints to the optimization problem [Audet *et al.*, 2000, Sasena, 2002, Boyle, 2007], and in modelling noisy functions [Bartz-Beielstein *et al.*, 2005, Huang *et al.*, 2006, Hutter *et al.*, 2009, Hutter, 2009].

In so-called "classical" experimental design, the problem to be addressed is often to learn the parameters $\boldsymbol{\zeta}$ of a function $g_{\boldsymbol{\zeta}}$ such that

$$y_i = g_{\boldsymbol{\zeta}}(\mathbf{x}_i) + \varepsilon_i, \forall i \in 1, \ldots, t$$

with noise ε_i (usually Gaussian) for scalar output y_i. $\mathbf{x}_i$ is the i^{th} set of experimental conditions. Usually, the assumption is that $g_{\boldsymbol{\zeta}}$ is linear, so that

$$y_i = \boldsymbol{\zeta}^T \mathbf{x}_i + \varepsilon_i.$$

An experiment is represented by a design matrix $\mathbf{X}$, whose rows are the inputs $\mathbf{x}_{1:t}$. If we let $\varepsilon \sim \mathcal{N}(0, \sigma)$, then for the linear model, the variance of the parameter estimate $\widehat{\boldsymbol{\zeta}}$ is

$$Var(\widehat{\boldsymbol{\zeta}}) = \sigma^2 (\mathbf{X}^T \mathbf{X})^{-1},$$

and for an input $\mathbf{x}_i$, the prediction is

$$Var(\widehat{y}_t) = \sigma^2 \mathbf{x}_i^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{x}_i.$$

An optimal design is a design matrix that minimizes some characteristic of the inverse moment matrix $(\mathbf{X}^T\mathbf{X})^{-1}$. Common criteria include A-optimality, which minimizes the trace; D-optimality, which minimizes the determinant; and E-optimality, which minimizes the maximum eigenvalue.

Experimental design is usually non-adaptive: the entire experiment is designed before data is collected. However, *sequential design* is an important and active subfield (e.g. [Williams *et al.*, 2000, Busby, 2009].

2.8 Active learning

Active learning is another area related to Bayesian optimization, and of particular relevance to our task. Active learning is closely related to experimental design and, indeed, the decision to describe a particular problem as active learning or experimental design is often arbitrary. However, there are a few distinguishing characteristics that most (but by no means all) active learning approaches use.

- Active learning is most often *adaptive*: a model exists that incorporates all the data seen, and this is used to sequentially select candidates for labelling. Once labelled, the data are incorporated into the model and new candidates selected. This is, of course, the same strategy used by Bayesian optimization.
- Active learning employs an oracle for data labelling, and this oracle is very often a human being, as is the case with interactive Bayesian optimization.
- Active learning is usually concerned with selecting candidates from a finite set of available data (*pool-based* sampling). Experimental design and optimization are usually concerned with continuous domains.
- Finally, active learning is usually used to learn a model for classification, or, less commonly, regression. Usually the candidate selection criterion is the maximization of some informativeness measure, or the minimization of uncertainty. These criteria are often closely related to the alphabetic criteria of experimental design.

An excellent recent overview of active learning is [Settles, 2010]. We are particularly interested in active learning because it often uses a human oracle for label acquisition. An example using GPs is the object categorization of Kapoor *et al.* [2007]. In this work, candidates are selected from a pool of unlabelled images so as to maximize the margin of the GP classifier and minimize the uncertainty. Osborne *et al.* [2010] also use active learning in a Gaussian process problem, deciding when to sample variables of interest in a sequential sampling problem with faults and changepoints. Chu and Ghahramani [2005a] briefly discuss how active learning could be used for *ranking* GPs, by selecting sample points that maximize entropy gained with a new preference relation.

Interesting recent work with GPs that straddles the boundary between active learning and experimental design is the sensor placement problem of Krause *et al.* [2008]. They examine several criteria, including maximum entropy, and argue for using mutual information. Ideally, they would like to simultaneously select a set of points to place the entire set of sensors in a way that maximizes the mutual information. This is essentially a classical experimental design problem with maximum mutual information as the design criterion. However, maximizing mutual information over a set of samples is NP-complete, so they use an active learning approach. By exploiting the submodularity of mutual information, they are able to show that sequentially selecting sensor locations that greedily maximize mutual information, they can bound the divergence of the active learning approach from the experimental design approach. This work influenced the GP-UCB acquisition function (§2.3.3).

Finally, as an aside, in active learning, a common acquisition strategy is selecting the point of maximum uncertainty. This is called *uncertainty sampling*

[Lewis and Gale, 1994]. GPs have in the useful property that the posterior variance (interpreted as uncertainty) is independent of the actual observations! As a result, if this is the criterion, the entire active learning scheme can be designed before a single observation are made, making adaptive sampling unnecessary.

2.9 Applications

Bayesian optimization has recently begun to appear in the machine learning literature as a means of optimizing difficult black box optimizations. A few recent examples include:

- Lizotte *et al.* [2007, 2008] used Bayesian optimization to learn a set of robot gait parameters that maximize velocity of a Sony AIBO ERS-7 robot. As an acquisition function, the authors used maximum probability of improvement (§2.3.1). They show that the Bayesian optimization approach not only outperformed previous techniques, but used drastically fewer evaluations.
- Frean and Boyle [2008] use Bayesian optimization to learn the weights of a neural network controller to balance two vertical poles with different weights and lengths on a moving cart.
- Cora's MSc thesis [2008] uses Bayesian optimization to learn a hierarchical policy for a simulated driving task. At the lowest level, using the vehicle controls as inputs and fitness to a course as the response, a policy is learned by which a simulated vehicle performs various activities in an environment.
- Martinez–Cantin *et al.* [2009] also applied Bayesian optimization to policy search. In this problem, the goal was to find a policy for robot path planning that would minimize uncertainty about its location and heading, as well as minimizing the uncertainty about the environmental navigation landmarks.
- Hutter's PhD thesis [2009] studies methods of automatically tuning algorithm parameters, and presents several sequential approaches using Bayesian optimization.
- The work of Osborne, Garnett *et al.* [Osborne, 2010, Osborne *et al.*, 2010, Garnett *et al.*, 2010b] uses Bayesian optimization to select the locations of a set of (possibly heterogenous) sensors in a dynamic system. In this case, the samples are a function of the locations of the entire set of sensors in the network, and the objective is the root mean squared error of the predictions made by the sensor network.

3 Bayesian Optimization for Preference Galleries

The model described above requires that each function evaluation have a scalar response. However, this is not always the case. In applications requiring human judgement, for instance, preferences are often more accurate than ratings. Prospect theory, for example, employs utility models based on relation to a reference point, based on evidence that the human perceptual apparatus is attuned to evaluate differences rather than absolute magnitudes [Kahneman and Tversky, 1979, Tversky and Kahneman, 1992]. We present here a Bayesian optimization application based on discrete choice for a "preference gallery" application, originally presented in [Brochu *et al.*, 2007a, Brochu *et al.*, 2007b].

In the case of a person rating the suitability of a procedurally-generated animation or image, each sample of valuation involves creating an instance with the given parameters and asking a human to provide feedback, which is interpreted as the function response. This is a very expensive class of functions to evaluate! Furthermore, it is in general impossible to even sample the function directly and get a consistent response from users. Asking humans to rate an animation on a numerical scale has built-in problems—not only will scales vary from user to user, but human evaluation is subject to phenomena such as *drift*, where the scale varies over time, *anchoring*, in which early experiences dominate the scale [Siegel and Castellan, 1988, Payne *et al.*, 1993]. However, human beings *do* excel at comparing options and expressing a preference for one over others [Kingsley, 2006]. This insight allows us to approach the optimization function in another way. By presenting two or more realizations to a user and requiring only that they indicate preference, we can get far more robust results with much less cognitive burden on the user [Kendall, 1975]. While this means we can't get responses for a valuation function directly, we model the valuation as a latent function, inferred from the preferences, which permits a Bayesian optimization approach.

Probability models for learning from discrete choices have a long history in psychology and econometrics [Thurstone, 1927, McFadden, 1980, Stern, 1990]. They have been studied extensively, for example, in rating chess players, and the Elo system [Élő, 1978] was adopted by the World Chess Federation FIDE to model the probability of one player beating another. It has since been adopted to many other two-player games such as Go and Scrabble, and, more recently, online computer gaming [Herbrich and Graepel, 2006].

Parts of §3.1 are based on [Chu and Ghahramani, 2005b], which presents a preference learning method using probit models and Gaussian processes. They use a Thurstone–Mosteller model (below), but with an innovative nonparametric model of the valuation function.

3.1 Probit model for binary observations

The *probit model* allows us to deal with binary observations of $f(\cdot)$ in general. That is, every time we try a value of $\mathbf{x}$, we get back a binary variable, say either zero or one. From the binary observations, we have to infer the latent function $f(\cdot)$. In order to marry the presentation in this section to the user modeling applications discussed later, we will introduce probit models in the particular case of preference learning.

Assume we have shown the user M pairs of items from a set of N instances.

In each case, the user has chosen which item she likes best. The data set therefore consists of the ranked pairs:

$$\mathcal{D} = \{\mathbf{r}_i \succ \mathbf{c}_i;\;\; i = 1, \ldots, M\},$$

where the symbol $\succ$ indicates that the user prefers $\mathbf{r}$ to $\mathbf{c}$. We use $\mathbf{x}_{1:t}$ to denote the t distinct elements in the training data. That is, $\mathbf{r}_i$ and $\mathbf{c}_i$ correspond to two elements of $\mathbf{x}_{1:t}$. $\mathbf{r}_i \succ \mathbf{c}_i$ can be interpreted as a binary variable that takes value 1 when $\mathbf{r}_i$ is preferred to $\mathbf{c}_i$ and is 0 otherwise.

In the probit approach, we model the value functions $v(\cdot)$ for items $\mathbf{r}$ and $\mathbf{c}$ as follows:

$$\begin{aligned} v(\mathbf{r}_i) &= f(\mathbf{r}_i) + \varepsilon \\ v(\mathbf{c}_i) &= f(\mathbf{c}_i) + \varepsilon, \end{aligned} \tag{7}$$

where the noise terms are Gaussian: $\varepsilon \sim \mathcal{N}(0, \sigma^2_{\text{noise}})$. Following [Chu and Ghahramani, 2005b], we assign a nonparametric Gaussian process prior to the unknown mean valuation: $f(\cdot) \sim \mathcal{GP}(0, K(\cdot, \cdot))$. That is, at the t training points:

$$P(\mathbf{f}) = |2\pi\mathbf{K}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\mathbf{f}^T\mathbf{K}^{-1}\mathbf{f}\right),$$

where $\mathbf{f} = \{f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_t)\}$.

Random utility models such as (7) have a long and influential history in psychology and the study of individual choice behaviour in economic markets. Daniel McFadden's Nobel Prize speech [McFadden, 2001] provides a glimpse of this history. Many more comprehensive treatments appear in classical economics books on discrete choice theory.

Under our Gaussian utility models, the probability that item $\mathbf{r}$ is preferred to item $\mathbf{c}$ is given by:

$$\begin{aligned} P(\mathbf{r}_i \succ \mathbf{c}_i | f(\mathbf{r}_i), f(\mathbf{c}_i)) &= P(v(\mathbf{r}_i) > v(\mathbf{c}_i) | f(\mathbf{r}_i), f(\mathbf{c}_i)) \\ &= P(\varepsilon - \varepsilon < f(\mathbf{r}_i) - f(\mathbf{c}_i)) \\ &= \Phi(Z_i), \end{aligned}$$

where

$$Z_i = \frac{f(\mathbf{r}_i) - f(\mathbf{c}_i)}{\sqrt{2}\sigma_{\text{noise}}}$$

and $\Phi(\cdot)$ is the CDF of the standard normal distribution. This model, relating binary observations to a continuous latent function, is known as the Thurstone-Mosteller law of comparative judgement [Thurstone, 1927, Mosteller, 1951]. In statistics it goes by the name of binomial-probit regression. Note that one could also easily adopt a logistic (sigmoidal) link function $\varphi(Z_i) = (1 + \exp(-Z_i))^{-1}$. In fact, such choice is known as the Bradley-Terry model [Stern, 1990]. If the user had more than two choices one could adopt a polychotomous regression model [Holmes and Held, 2006]. This multi-category extension would, for example, enable the user to state no preference, or a degree of preference for any of the two items being presented.

Note that this approach is related to, but distinct from the binomial logistic-linear model used in geostatistics [Diggle *et al.*, 1998], in which the responses y represent the outcomes of Bernoulli trials which are conditionally independent given the model (i.e., the responses are binary observations $y_t \in \{0, 1\}$ for $\mathbf{x}_t$, rather than preference observations between $\{\mathbf{r}_t, \mathbf{c}_t\}$).

Our goal is to estimate the posterior distribution of the latent utility function given the discrete data. That is, we want to maximize

$$P(\mathbf{f}|\mathcal{D}) \propto P(\mathbf{f}) \prod_{i=1}^{M} P(\mathbf{r}_i \succ \mathbf{c}_i | f(\mathbf{r}_i), f(\mathbf{c}_i)).$$

Although there exist sophisticated variational and Monte Carlo methods for approximating this distribution, we favour a simple strategy: Laplace approximation. The Laplace approximation follows from Taylor-expanding the log-posterior about a set point $\widehat{\mathbf{f}}$:

$$\log P(\mathbf{f}|\mathcal{D}) = \log P(\widehat{\mathbf{f}}|\mathcal{D}) + \mathbf{g}^T(\mathbf{f} - \widehat{\mathbf{f}}) - \frac{1}{2}(\mathbf{f} - \widehat{\mathbf{f}})^T \mathbf{H}(\mathbf{f} - \widehat{\mathbf{f}}),$$

where $\mathbf{g} = \nabla_{\mathbf{f}} \log P(\mathbf{f}|\mathcal{D})$ and $\mathbf{H} = -\nabla_{\mathbf{f}} \nabla_{\mathbf{f}} \log P(\mathbf{f}|\mathcal{D})$. At the mode of the posterior ($\widehat{\mathbf{f}} = \mathbf{f}_{\text{MAP}}$), the gradient $\mathbf{g}$ vanishes, and we obtain:

$$P(\mathbf{f}|\mathcal{D}) \approx P(\widehat{\mathbf{f}}|\mathcal{D}) \exp\left[-\frac{1}{2}(\mathbf{f} - \widehat{\mathbf{f}})\mathbf{H}(\mathbf{f} - \widehat{\mathbf{f}})\right]$$

In order to obtain this approximation, we need to compute the maximum a posteriori (MAP) estimate $\mathbf{f}_{\text{MAP}}$, the gradient $\mathbf{g}$ and the information matrix $\mathbf{H}$.

The gradient is given by:

$$\begin{aligned}
\mathbf{g} &= \nabla_{\mathbf{f}} \log P(\mathbf{f}|\mathcal{D}) \\
&= \nabla_{\mathbf{f}} \left[const - \frac{1}{2}\mathbf{f}^T \mathbf{K}^{-1} \mathbf{f} + \sum_{i=1}^{M} \log \Phi(Z_i) \right] \\
&= -\mathbf{K}^{-1}\mathbf{f} + \nabla_f \left[\sum_{i=1}^{M} \log \Phi(Z_i) \right] = -\mathbf{K}^{-1}\mathbf{f} + \mathbf{b},
\end{aligned}$$

where the j-th entry of the N-dimensional vector $\mathbf{b}$ is given by:

$$b_j = \frac{1}{\sqrt{2}\sigma_{\text{noise}}} \sum_{i=1}^{M} \frac{\phi(Z_i)}{\Phi(Z_i)} \left[\frac{\partial}{\partial f(\mathbf{x}_j)} (f(\mathbf{r}_i) - f(\mathbf{c}_i)) \right],$$

where $\phi(\cdot)$ denotes the PDF of the standard normal distribution. Clearly, the derivative $h_i(\mathbf{x}_j) = \frac{\partial}{\partial f(\mathbf{x}_j)}(f(\mathbf{r}_i) - f(\mathbf{c}_i))$ is 1 when $\mathbf{x}_j = \mathbf{r}_i$, -1 when $\mathbf{x}_j = \mathbf{c}_i$ and 0 otherwise. Proceeding to compute the second derivative, one obtains the

preferences
$0.2 \succ 0.1$
$0.35 \succ 0.5$
$0.2 \succ 0.35$
$0.2 \succ 0.6$
$0.8 \succ 0.7$

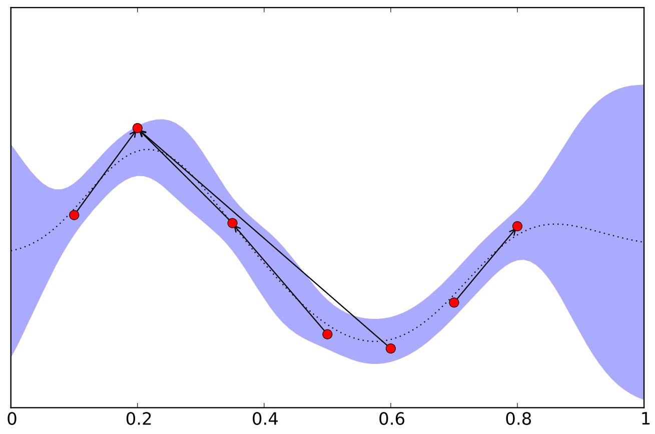


Figure 8: *Example of a set of preference relations (left table) used to infer a GP (right plot) on a toy problem. The preferences are indicated as a set of preferences between two points in the space, which serves as input to a function that finds a Gaussian process that takes into account all the available preference information, as well as prior information on the smoothness and noise.*

Hessian: $\mathbf{H} = \mathbf{K}^{-1} + \mathbf{C}$, where the matrix $\mathbf{C}$ has entries

$$\begin{aligned}\mathbf{C}_{m,n} &= -\frac{\partial^2}{\partial f(\mathbf{x}_m)\partial f(\mathbf{x}_n)} \sum_{i=1}^{M} \log \Phi(Z_i) \\ &= \frac{1}{2\sigma^2} \sum_{i=1}^{M} h_i(\mathbf{x}_m) h_i(\mathbf{x}_n) \left[\frac{\phi(Z_i)}{\Phi^2(Z_i)} + \frac{\phi^2(Z_i)}{\Phi(Z_i)} Z_i \right]\end{aligned}$$

The Hessian is a positive semi-definite matrix. Hence, one can find the MAP estimate with a simple Newton–Raphson recursion:

$$\mathbf{f}^{\text{new}} = \mathbf{f}^{\text{old}} - \mathbf{H}^{-1}\mathbf{g} \mid_{\mathbf{f}=\mathbf{f}^{\text{old}}} .$$

At $\mathbf{f} = \mathbf{f}_{\text{MAP}}$, we have

$$P(\mathbf{f}|\mathcal{D}) \approx \mathcal{N}\left(\mathbf{K}\mathbf{b}, (\mathbf{K}^{-1} + \mathbf{C})^{-1}\right).$$

with $\mathbf{b} = \mathbf{K}^{-1}\mathbf{f}_{\text{MAP}}$. The goal of our derivation, namely the predictive distribution $P(f_{t+1}|\mathcal{D})$, follows by straightforward convolution of two Gaussians:

$$\begin{aligned}P(f_{t+1}|\mathcal{D}) &= \int P(f_{t+1}|\mathbf{f}_{\text{MAP}}) P(\mathbf{f}_{\text{MAP}}|\mathcal{D}) d\mathbf{f}_{\text{MAP}} \\ &\propto \mathcal{N}(\mathbf{k}^T \mathbf{K}^{-1} \mathbf{f}_{\text{MAP}}, k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}^T (\mathbf{K} + \mathbf{C}^{-1})^{-1} \mathbf{k}).\end{aligned}$$

An example of the procedure on a toy 1D problem is shown in Figure 8. We can see that the inferred model explains the observed preferences, while also adhering to prior information about smoothness and noise.

3.2 Application: Interactive Bayesian optimization for material design

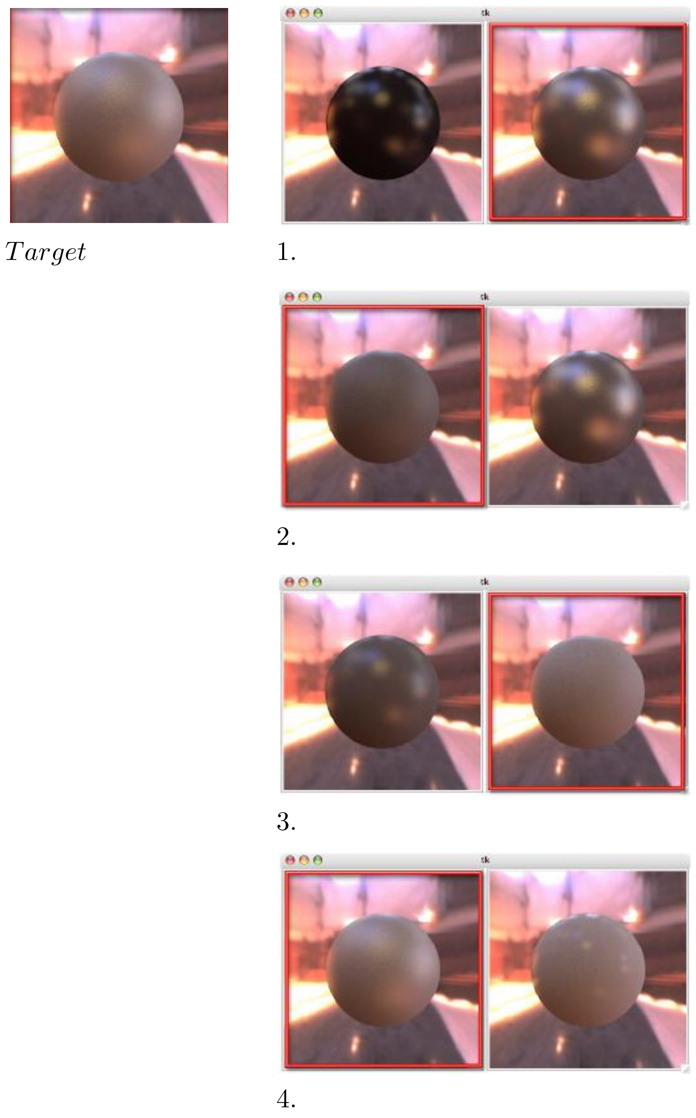


Figure 9: *A shorter-than-average but otherwise typical run of the BRDF preference gallery tool. At each (numbered) iteration, the user is provided with two images generated with parameter instances and indicates the one they think most resembles the target image (top-left) they are looking for. The boxed images are the user's selections at each iteration.*

Properly modeling the appearance of a material is a necessary component of realistic image synthesis. The appearance of a material is formalized by the notion of the Bidirectional Reflectance Distribution Function (BRDF). In computer graphics, BRDFs are most often specified using various analytical models. Analytical models that are of interest to realistic image synthesis are the ones that observe the physical laws of reciprocity and energy conservation while typically also exhibiting shadowing, masking and Fresnel reflectance phenomenon. Realistic models are therefore fairly complex with many parameters that need to be adjusted by the designer for the proper material appearance. Unfortunately these parameters can interact in non-intuitive ways, and small adjustments to certain settings may result in non-uniform changes in the appearance. This can make the material design process quite difficult for the artist end user, who is not expected to be an expert in the field, but who knows the look that she desires for a particular application without necessarily being interested in understanding the various subtleties of reflection. We attempt to deal with this using a preference gallery approach, in which users are simply required to view two or more images rendered with different material properties and indicate which they prefer, in an iterative process.

We use the interactive Bayesian optimization model with probit responses on an example gallery application for helping users find a BRDF. For the purposes of this example, we limit ourselves to isotropic materials and ignore wavelength dependent effects in reflection. Our gallery demonstration presents the user with two BRDF images at a time. We start with four predetermined queries to "seed" the parameter space, and after that use the learned model to select gallery images. The GP model is updated after each preference is indicated. We use parameters of real measured materials from the MERL database for seeding the parameter space, but can draw arbitrary parameters after that.

By querying the user with a paired comparison, one can estimate statistics of the valuation function at the query point, but only at considerable expense. Thus, we wish to make sure that the samples we do draw will generate the maximum possible improvement.

Our method for achieving this goal iterates over the following steps:

1. **Present the user with a new set of instances and record preferences from the user**: Augment the training set of paired choices with the new user data.
2. **Infer the valuation function**: Here we use a Thurstone–Mosteller model with Gaussian processes. See §3.1 for details. Note that in this application, the valuation function is the objective of Bayesian optimization. We will use the terms interchangeably.
3. **Optimize the acquisition function of the valuation to obtain the query points for the next gallery**: Methods for selecting a set of instances are described in §3.2.1.

3.2.1 User Study

To evaluate the performance of our application, we have run a simple user study in which the generated images are restricted to a subset of 38 materials from the

algorithm	**trials**	n **(mean ± std)**
random	50	18.40 ± 7.87
$\mathbf{argmax}_\sigma$	50	17.87 ± 8.60
$\mathbf{argmax}_{\mathrm{EI}}$	50	8.56 ± 5.23

Table 1: *Results of the user study on the BRDF gallery.*

MERL database that we deemed to be representative of the appearance space of the measured materials. The user is given the task of finding a single randomly-selected image from that set by indicating preferences. Figure 9 shows a typical user run, where we ask the user to use the preference gallery to find a provided target image. At each step, the user need only indicate the image they think looks most like the target. This would, of course, be an unrealistic scenario if we were to be evaluating the application from an HCI stance, but here we limit our attention to the model, where we are interested here in demonstrating that with human users maximizing valuation is preferable to learning the entire latent function.

Using five subjects, we ran 50 trials each of three different methods of selecting sample pairs[3]. In all cases, one of the pair is the incumbent $\mathrm{argmax}_{\mathbf{x}_i}\, \mu(\mathbf{x}_i), \mathbf{x}_i \in \mathbf{x}_{1:t}$. The second is selected via one of the following methods:

- **random** The second point is sampled uniformly from the parameter domain $\mathcal{A}$.
- $\mathbf{argmax}_\sigma$ The second point is the point of highest uncertainty, $\mathrm{argmax}_{\mathbf{x}}\, \sigma(\mathbf{x})$.
- $\mathbf{argmax}_{\mathrm{EI}}$ The second point is the point of maximum expected improvement, $\mathrm{argmax}_{\mathbf{x}}\, \mathrm{EI}(\mathbf{x})$.

The results are shown in Table 1. n is the number clicks required of the user to find the target image. Clearly $\mathrm{argmax}_{\mathrm{EI}}$ dominates, with a mean n less than half that of the competing algorithms. Interestingly, selecting images using maximum variance does not perform much better than random. We suspect that this is because argmax_σ has a tendency to select images from the corners of the parameter space, which adds limited information to the other images, whereas Latin hypercubes at least guarantees that the selected images fill the space.

4 Bayesian Optimization for Hierarchical Control

In general, problem solving and planning becomes easier when it is broken down into subparts. Variants of functional hierarchies appear consistently in video

[3]An empirical study of various methods on a variety of test functions, and a discussion of why these were selected can be found in [Brochu *et al.*, 2007a].

game AI solutions, from behaviour trees, to hierarchically decomposed agents (teams vs. players), implemented by a multitude of customized hierarchical state machines. The benefits are due to isolating complex decision logic to fairly independent functional units (tasks). The standard game AI development process consists of the programmer implementing a large number of behaviours (in as many ways as there are published video games), and hooking them up to a more manageable number of tuneable parameters. We present a class of algorithms that attempt to bridge the gap between game development, and general reinforcement learning. They reduce the amount of hand-tuning traditionally encountered during game development, while still maintaining the full flexibility of manually hard-coding a policy when necessary.

The Hierarchical Reinforcement Learning [Barto and Mahadevan, 2003] field models repeated decision making by structuring the policy into tasks (actions) composed of subtasks that extend through time (temporal abstraction) and are specific to a subset of the total world state space (state abstraction). Many algorithms have recently been developed, and are described further in Section 4.1. The use of Bayesian optimization for control has previously been proposed by Murray-Smith and Sbarbaro [2002], and (apparently independently) by Frean and Boyle [2008], who used it for a control problem of balancing two poles on a cart. This work did not involve a nonhierarchical setting, however.

The exploration policies typically employed in HRL research tend to be slow in practice, even after the benefits of state abstraction and reward shaping. We demonstrate an integration of the MAXQ hierarchical task learner with Bayesian active exploration that significantly speeds up the learning process, applied to hybrid discrete and continuous state and action spaces. Section 4.2 describes an extended Taxi domain, running under The Open Racing Car Simulator [Wymann *et al.*, 2009], a 3D game engine that implements complex vehicle dynamics complete with manual and automatic transmission, engine, clutch, tire, suspension and aerodynamic models.

4.1 Hierarchical Reinforcement Learning

Manually coding hierarchical policies is the mainstay of video game AI development. The requirements for automated HRL to be a viable solution are it must be easy to customize task-specific implementations, state abstractions, reward models, termination criteria and it must support continuous state and action spaces. Out of the solutions investigated, MAXQ [Dietterich, 2000] met all our requirements, and was the easiest to understand and get positive results quickly. The other solutions investigated include HAR and RAR [Ghavamzadeh, 2005] which extend MAXQ to the case of average rewards (rather than discounted rewards). The implementation of RAR is mostly the same as MAXQ, and in our experiments gave the same results. Hierarchies of Abstract Machines (HAM) [Parr, 1998] and ALisp [Andre, 2003] are an exciting new development that has been recently applied to a Real-Time-Strategy (RTS) game [Marthi *et al.*, 2005]. ALisp introduces programmable reinforcement learning policies that allows the programmer to specify choice points for the algorithm to optimize. Although the formulation is very nice and would match game AI development processes, the underlying solver based on HAMs flattens the task hierarchy by including the program's memory and call-stack into a new joint-state space, and solves this new MDP instead. It is less clear how to extend and implement per-task customized learning with this formulation. Even if this difficulty is surmounted, as evidenced by the last line in the concluding remarks of [Marthi *et al.*, 2005], there is an imperative need for designing faster algorithms in HRL. This paper aims to address this need.

In our solution, we still require a programmer or designer to specify the task hierarchy. In most cases breaking a plan into sub-plans is much easier than coding the decision logic. With the policy space constrained by the task hierarchy, termination and state abstraction functions, the rate of learning is greatly improved, and the amount of memory required to store the solution reduces. The benefits of HRL are very dependant however on the quality of these specifications, and requires the higher-level reasoning of a programmer or designer. An automatic solution to this problem would be an agent that can learn how to program, and anything less than that will have limited applicability.

We can use Bayesian optimization to learn the relevant aspects of value functions by focusing on the most relevant parts of the parameter space. In the work on this section, we use refer to the objective as the *value* function, to be consistent with the HRL literature.

4.1.1 Semi-MDPs

Each task in an HRL hierarchy is a semi-Markov Decision Process [Sutton *et al.*, 1999], that models repeated decision making in a stochastic environment, where the actions can take more than one time step. Formally, an SMDP is defined as a tuple: $\{S, A, P(s', N|s, a), R(s, a)\}$ where S is the set of state variables, A is a set of actions, $P(s', N|s, a)$ is the transition probability of arriving to state s' in N time steps after taking action a in s, and $R(s, a)$ is the reward received. The solution of this process is a policy $\pi^*(s) \in A$, that selects the action with the highest expected discounted reward in each state. The function $V^*(s)$ is the value of state s when following the optimal policy. Equivalently, the $Q^*(s, a)$ function stores the value of taking action a in state s and following the optimal policy thereafter. These quantities follow the classical Bellman recursions:

$$\begin{aligned} V^*(s) &= \max_{a \in A} \left[R(s,a) + \gamma \sum_{s',N} P(s', N|s, a) \gamma^N V^*(s') \right] \\ Q^*(s,a) &= R(s,a) + \gamma \sum_{s',N} P(s', N|s, a) \gamma^N V^*(s') \qquad (8) \end{aligned}$$

4.1.2 Hierarchical Value Function Decomposition

A task i in MAXQ [Dietterich, 2000] is defined as a tuple: $\{A_i, T_i(s), Z_i(s), \pi_i(s)\}$ where s is the current world state, A_i is a set of subtasks, $T_i(s) \in \{true, false\}$

is a termination predicate, $Z_i(s)$ is a state abstraction function that returns a subset of the state relevant to the current subtask, and $\pi_i(s) \in A_i$ is the policy learned by the agent (or used to explore during learning). Each task is effectively a separate, decomposed SMDP that has allowed us to integrate active learning for discrete map navigation with continuous low-level vehicle control. This is accomplished by decomposing the Q function into two parts:

$$\begin{aligned} a &= \pi_i(s) \\ Q^\pi(i,s,a) &= V^\pi(a,s) + C^\pi(i,s,a) \\ C^\pi(i,s,a) &= \sum_{s',N} P_i^\pi(s',N|s,a)\gamma^N Q^\pi(i,s',\pi_i(s')) \\ V^\pi(i,s) &= \begin{cases} Q^\pi(i,s,\pi_i(s)) & \text{if composite} \\ \sum_{s'} P(s'|s,i)R(s'|s,i) & \text{if primitive} \end{cases} \end{aligned} \tag{9}$$

Here, γ is the discount factor, i is the current task, and a is a child action given that we are following policy π_i. The Q function is decomposed into two parts: the value of V^π being the expected one step reward, plus C^π which is the expected completion reward for i after a completes. V is defined recursively, as the expected value of its child actions, or the expected reward itself if i is a primitive (atomic) action. The MAXQ learning routine is a simple modification of the typical Q-learning algorithm. In task i, we execute subtask a, observe the new state s' and reward r. If a is primitive, we update $V(s,a)$, otherwise we update $C(i,s,a)$, with learning rate $\alpha \in (0,1)$:

$$\begin{aligned} V(a,s) &= (1-\alpha) \times V(a,s) + \alpha \times r \\ C(i,s,a) &= (1-\alpha) \times C(i,s,a) + \alpha \times \max_{a'} Q(i,s',a') \end{aligned} \tag{10}$$

An important consideration in HRL is whether the policy calculated is hierarchically or recursively optimal. Recursive optimality, satisfied by MAXQ and RAR, means that each subtask is locally optimal, given the optimal policies of the descendants. This may result in a suboptimal overall policy because the effects of tasks executed outside of the current task's scope are ignored. For example if there are two exits from a room, a recursively optimal policy would pick the closest exit, regardless of the final destination. A hierarchically optimal policy (computed by the HAR [Ghavamzadeh, 2005] and HAM [Andre, 2003] three-part value decompositions) would pick the exit to minimize total travelling time, given the destination. A recursively optimal learning algorithm however generalizes subtasks easier since they only depend on the local state, ignoring what would happen after the current task finishes. So both types of optimality are of value in different degrees for different cases. The MAXQ formulation gives a programmer or designer the ability to selectively enable hierarchical optimality by including the relevant state features as parameters to a task. However, it may be difficult to identify the relevant features, as they would be highly application specific.

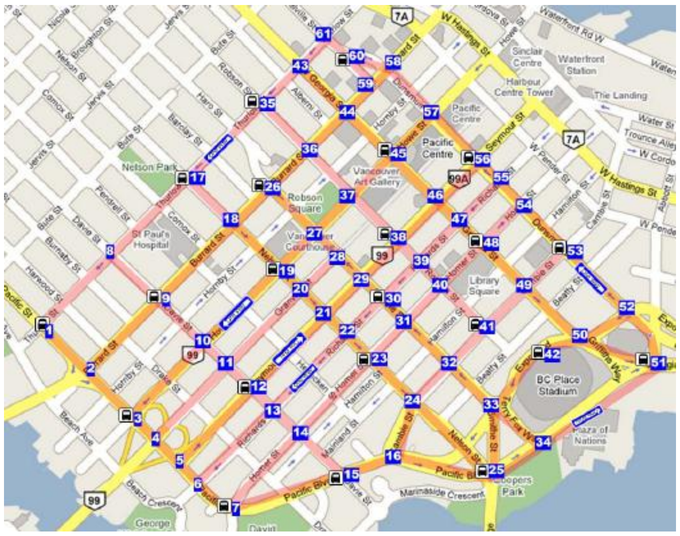


Figure 10: ***City Experiment*** *uses a simplified map (orange overlay) roughly based on downtown Vancouver, and used by the TORCS simulator. Each waypoint is labeled, and pickup and dropoff locations are marked by the Taxi icons. One way streets are accounted for in the waypoint adjacency matrix. Source image care of Google Maps.*

4.2 Application: The Vancouver Taxi Domain

Our domain is a city map roughly based on a portion of downtown Vancouver, British Columbia, illustrated in Figure 10. The data structure is a topological map (a set of intersection nodes and adjacency matrix) with 61 nodes and 22 possible passenger pickup and drop-off locations. The total navigable area of the map is roughly 28 kilometers.

The state model includes both discrete variables used in the top layers of the task hierarchy, as well as continuous variables used by the *Follow* task that tracks a trajectory, and are described in Table 3. The original taxi domain [Dietterich, 2000] is a 5x5 grid, with 4 possible pickup and dropoff destinations, and 6 actions (pickup, dropoff, and navigating North, South, East, West).

Table 2 makes a rough comparison between the size of our extended application and the original taxi domain. Ignoring the continuous trajectory states (including the *Stopped* flag) and assuming the taxi hops from one intersection to an adjacent one in a single time step results in a fully discrete problem. A flat learning solution scales poorly, not only in terms of world samples required, but also in the size of the computed policy (if represented in a discrete table). The extended task hierarchy illustrated in Figure 11 requires just a little bit more memory than the small 5x5 taxi domain.

Table 2: Comparing Domain Size

Domain	Size of final policy
5x5 Taxi Flat	$\sim 12,200$ *bytes*
Vancity Flat	$\sim 1,417,152$ *bytes*
Vancity Hierarchical	$\sim 18,668$ *bytes*

Table 3: States and Task Parameters

Name	Range/Units	Description
$TaxiLoc$	{0,1,..61}	current taxi waypoint #, or 0 if in transit between waypoints
$PassLoc$	{0,1,..22}	passenger waypoint #, or 0 if in taxi
$PassDest$	{1,2,..22}	passenger destination waypoint #
$LegalLoad$	$\{true, false\}$	true if taxi is empty and at passenger, or loaded and at target
Stopped	$\{true, false\}$	indicates whether the taxi is at a complete stop
T	{1,2,..22}	passenger location or destination parameter passed into *Navigate*
WP	{1,2,..22}	waypoint parameter adjacent to *TaxiLoc* passed to *Follow*
Y_{err}	meters	lateral error between desired point on the trajectory and vehicle
V_y	meters/second	lateral velocity (to detect drift)
V_{err}	meters/second	error between desired and real speed
Ω_{err}	radians	error between trajectory angle and vehicle yaw

4.2.1 State Abstraction, Termination and Rewards

Figure 11 compares the original task hierarchy, with our extended version that includes continuous trajectory following and a hard-coded *Park* task. The state abstraction function filters out irrelevant states while computing the hash key for looking up and updating values of $V(s,a)$ and $C(i,s,a)$, where s is the current state, i is the current task, and a is the child task. The *Follow* task has been previously trained with the Active Policy optimizer from section 4.3.1 and the policy parameters fixed before learning the higher level tasks. Algorithms RAR and MAXQ are applied to all the tasks above and including *Navigate*, which also uses the Active Path learning algorithm from section 4.3.2. Here is a summary of each task, including its reward model, termination predicate T_i, and state abstraction function Z_i:

Root - this task selects between *Get* and *Put* to pickup and deliver the passenger. It requires no learning because the termination criteria of the subtasks fully determine when they should be invoked. $T_{Root} = (PassLock = PassDest)$ and $Z_{Root} = \{\}$.

Get - getting the passenger involves navigating through the city, parking the car and picking up the passenger. In this task, the *LegalLoad* state is true when the taxi is at the passenger's location. Receives a reward of 750 when the passenger is picked up, $T_{Get} = ((PassLoc = 0)\ \mathit{or}\ (PassLoc = PassDest))$, and $Z_{Get} = \{\}$.

Put - similar to *Get*, also receives reward of 750 when passenger is successfully delivered. The passenger destination *PassDest* is passed to the *Navigate* task. The abstracted *LegalLoad* state is true when the taxi is at the passenger's destination location. $T_{Put} = ((PassLoc > 0)\ \mathit{or}\ (PassLoc = PassDest))$ and $Z_{Put} = \{\}$.

Pickup - this is a primitive action, with a reward of 0 if successful, and -2500 if a pickup is invalid (if the taxi is not stopped, or if *LegalLoad* is false). $Z_{Pickup} = \{LegalLoad, Stopped\}$.

Dropoff - this is a primitive action, with a reward of 1500 if successful, and -2500 if a dropoff is invalid. $Z_{Dropoff} = \{LegalLoad, Stopped\}$.

Navigate - this task learns the sequence of intersections from the current *TaxiLoc* to a target destination *T*. By parameterizing the value function of this task, we can apply Active Path learning as described in Section 4.3.2. $T_{Navigate} = (TaxiLoc = T)$ and $Z_{Navigate} = \{T, TaxiLoc\}$.

Follow - this is the previously trained continuous trajectory following task that takes as input an adjacent waypoint *WP*, and generates continuous steering and throttle values to follow the straight-line trajectory from *TaxiLoc* to *WP*. $T_{Follow} = (TaxiLoc = WP)$ and $Z_{Follow} = \{WP, \Omega_{err}, V_{err}, Y_{err}, V_y\}$.

Park - this is a hard-coded task which simply puts on the brakes ($steer = 0$, $throttle = -1$).

Drive - this performs one time step of the physics simulation, with the given steer and throttle inputs. The default reward per time step of driving is -0.75.

4.3 Bayesian Optimization for Hierarchical Policies

The objective of Bayesian optimization is to learn properties of the value function or policy with as few samples as possible. In direct policy search, where this idea has been explored previously [Martinez–Cantin *et al.*, 2007], the evaluation of the expected returns using Monte Carlo simulations is very costly. One, therefore, needs to find a peak of this function with as few policy iterations as possible. As shown here, the same problem arises when we want to learn an approximation of the value function only over the relevant regions of the state space. Bayesian optimization provides an exploration-exploitation mechanism for finding these relevant regions and fitting the value function where needed.

When carrying out direct policy search [Ng and Jordan, 2000], the Bayesian optimization approach has several advantages over the policy gradients method [Baxter and Bartlett, 2001]: it is derivative free, it is less prone to be caught in

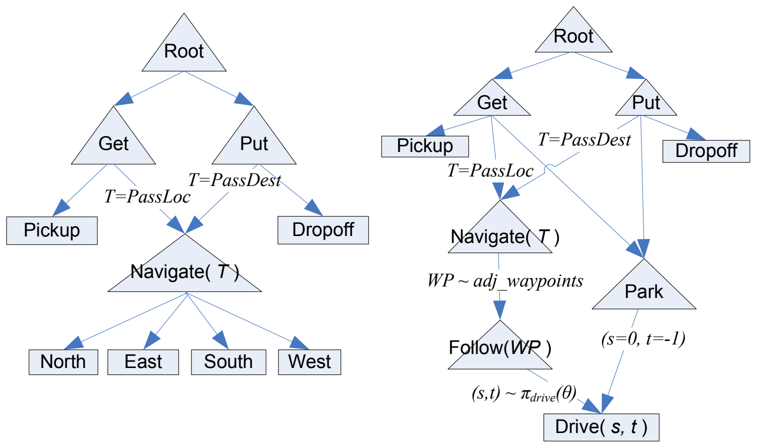


Figure 11: ***Task Hierarchies.*** *Each composite task is a separate SMDP whose policy is optimal given the optimal policies of its subtasks (recursive optimality). Triangles are composite tasks, and rectangle are primitive actions. The hierarchy on the right simplifies learning by reusing policies for navigating form waypoint to waypoint, and the Navigation task only needs to learn the sequence of waypoints to get to the destination. For the continuous case, the discrete actions N/S/E/W are replaced by one continuous* Drive(steer, throttle) *task, with driving parameters generated by the parameterized policy contained in the* Follow *task.*

the first local minimum, and it is explicitly designed to minimize the number of expensive value function evaluations.

4.3.1 Active Policy Optimization

Algorithm 2 Bayesian Active Learning with GPs

```
1: $N = 0$
2: Update the expected improvement function over $D_{1:N}$.
3: Choose $\mathbf{x}_{N+1} = \text{argmax}_x EI(\mathbf{x})$.
4: Evaluate $V_{N+1} = V(\mathbf{x}_{N+1})$ and halt if a stopping criterion is met.
5: Augment the data $D_{1:N+1} = \{D_{1:N}, (\mathbf{x}_{N+1}, V_{N+1})\}$.
6: $N = N + 1$ and go to step 2.
```

The lowest level *Drive* task uses the parameterized function illustrated in Figure 12 to generate continuous steer and throttle values, within the range of -1 to 1. The $|\mathbf{x}| = 15$ parameters (weights) are trained using the Bayesian active policy learning Algorithm 2. We first generate and evaluate a set of 30 Latin hypercube samples of $\mathbf{x}$ and store them and corresponding values vector V in the data matrix $\mathcal{D}$. The value of a trajectory is the negative accumulated error between the car's position and velocity, and the desired position and velocity. The policy evaluation consists of averaging 10 episodes along the same trajec-

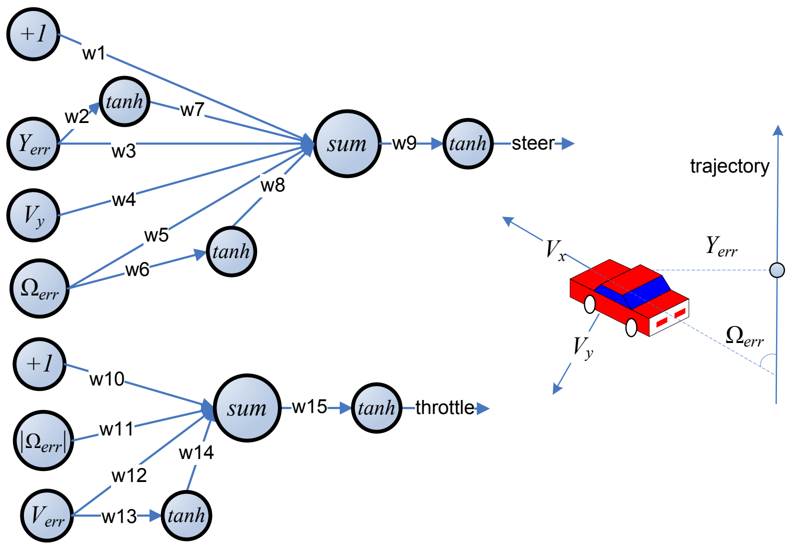


Figure 12: ***Trajectory-following policy:*** *this parameterized policy, inspired by Ng* et al *[2003] minimizes the error between the vehicle's heading and velocity while following a trajectory. The positional errors X_{err} and Y_{err} are in trajectory coordinates, Ω_{err} refers to the difference between the current heading and the trajectory tangent, and V_{err} is the difference between the real and desired velocities.*

tory but with different, evenly spaced starting angles, where the car needs to accelerate from rest, go to the first waypoint, perform a u-turn, and arrive back to the starting location. In a noisier environment, more samples would be necessary to properly evaluate a policy. The TORCS simulator is deterministic, and a small amount of noise arises from unmodeled tire slipping and random bumpiness of the road. The 10 different starting angles were sufficient for evaluating a policy in our experiments. Subsequently, we perform the iteration described in Algorithm 2 to search for the best instantiation of the parameters.

4.3.2 Active Value Function Learning

The *Navigate* task learns path finding from any intersection in the topological map to any of the destinations. Although this task operates on a discrete set of waypoints, the underlying map coordinates are continuous, and we can again apply active exploration with GPs.

Unlike the previous algorithm that searches for a set of optimal parameters, Algorithm 3 learns the value function at a finite set of states, by actively generating exploratory actions; it is designed to fit within a MAXQ task hierarchy. The 4-dimensional value function $V(\mathbf{x})$ in this case is parameterized by two 2D map coordinates $\mathbf{x} = \{x_C, y_C, x_T, y_T\}$, and stores the sum of discounted rewards while travelling from the current intersection $|C| = 61$ to the target $|T| = 22$. The 1342 sampled instances of $\mathbf{x}_{1:1342}$ and corresponding V vector are stored

Algorithm 3 Active Path Learning with GPs

```
1: function $NavigateTaskLearner(Navigate\ i, State\ s)$
2: let $trajectory$=() - list of all states visited in $i$
3: let $intersections$=() - intersection states visited in $i$
4: let $visits = 0$ - # of visits at an intersection in $i$
5: while $Terminated_i(s)$ is false do
6:    choose adjacent intersection $WP$ using $\epsilon$-greedy or Active exploration.
7:    let $childSeq = Follow(WP, s)$
8:    append $childSeq$ onto the front of $trajectory$
9:    observe result state $s'$
10:   $N$ = length( $childSeq$ )
11:   $R = \sum_{j=1}^{N} \gamma^{N-j} \times r_j$ be the total discounted reward received from $s$ to $s'$
12:   $V_s' = V(TaxiLoc_{s'}, Target_i)$ { guaranteed <= 0}
13:   $V_s = V(TaxiLoc_s, Target_i)$ { guaranteed <= 0}
14:   if $Terminated_i(s')$ is true then
15:      $V_s \leftarrow (1-\alpha) \times V_s + \alpha \times R$
16:      for all $j = 1$ to length(intersections) do
17:         $\{s', N', R'\}$ = intersections($j$)
18:         $R \leftarrow R' + \gamma^{N'} \times R$
19:         $V_s' \leftarrow V(TaxiLoc_{s'}, Target_i)$
20:         $V_s' \leftarrow (1-\alpha) \times V_s' + \alpha \times R$
21:      end for
22:   else
23:      append $\{s, N, R\}$ onto the front of $intersections$
24:      $visits(TaxiLoc_s) \leftarrow visits(TaxiLoc_s) + 1$
25:      $penalty \leftarrow V_s \times visits(TaxiLoc_s)$ {prevent loops}
26:      $V_s \leftarrow (1-\alpha) \times V_s + \alpha(penalty + R + \gamma^N \times V_s')$
27:   end if
28:   $s = s'$
29: end while
30: return $trajectory$
```

in the data matrix $\mathcal{D}$; it is initialized with $V(x_T, y_T, x_T, y_T) = 0$ for all target destinations T, which enables the GP to create a useful response surface without actually having observed anything yet.

In the $\epsilon-$greedy experiments, a random intersection is chosen with chance 0.1, and the greedy one with chance 0.9. For the active exploration case, we fit a GP over the data matrix $\mathcal{D}$, and pick the adjacent intersection that maximizes the expected improvement function. We parameterize this function with an annealing parameter that decays over time such that initially we place more importance on exploring.

The true value will not be known until the *Navigate* task reaches its destination and terminates, but we still need to mark visited intersections to avoid indefinite looping. Lines 23–26 compute an estimated value for $V(s)$ by summing the immediate discounted reward of executing $Follow(WP, s)$ with the discounted, previously recorded value of the new state $V(s')$, and a heuristic penalty factor to avoid looping. Once we reach the destination of this task, $Target_i$, we have the necessary information to propagate the discounted reward to all the intersections along the trajectory, in lines 15–21.

4.4 Simulations

The nature of the domain requires that we run policy optimization first to train the *Follow* task. This is reasonable, since the agent cannot be expected to learn map navigation before learning to drive the car. Figure 13 compares the results of three different values for the GP kernel k, when running the active policy optimization algorithm from §4.3.1. The desired velocity is $60km/hr$, a time step lasts 0.25 seconds, and the trajectory reward $R = -\sum_t \left[1 \times \tilde{Y}^2_{err} + 0.8 \times \tilde{V}^2_{err} + 1 \times \tilde{\Omega}^2_{err} + 0.7 \times \tilde{\mathbf{a}}'\tilde{\mathbf{a}}\right]$ is the negative weighted sum of normalized squared error values between the vehicle and the desired trajectory, including $\mathbf{a} = [\mathbf{steer}, \mathbf{throttle}]$ to penalize for abrupt actions. After ~ 50 more parameter samples (after the initial 30 random samples), the learner has already found a useable policy.

Subsequently, the best parameters are fixed inside the *Follow* task, and we run the full hierarchical learners, with results in Figure 14. We averaged the results from 10 runs of RAR, MAXQ, and the value learning Algorithm 3 applied only to the *Navigate* task (with the rest of the hierarchy using MAXQ). All the experiments use the hierarchical task model presented in Section 4.2.1. Each reward time step lasts 0.3 seconds, so the fastest learner, V_{TM} GP with $\epsilon = 0.2$ drove for ~ 4 hours real-time at ~ 60 km/hr before finding a good approximation of the $V_{Navigate}$ value function. Refer to Figure 15 for an intuition of how fitting the GP over the samples values transfers observations to adjacent areas of the state space. This effect is controlled through the GP kernel parameter k. While the application is specific to navigating a topological map, the algorithm is general and can be applied to any continuous state spaces of reasonable dimensionality.

5 Discussion and advice to practitioners

Bayesian optimization is a powerful tool for machine learning, where the problem is often not acquiring data, but acquiring labels. In many ways, it is like conventional active learning, but instead of acquiring training data for classification or regression, it allows us to develop frameworks to efficiently solve novel kinds of learning problems such as those discussed in Sections 3 and 4. It proves us with an efficient way to learn the solutions to problems, and to collect data, all within a Bayesian framework.

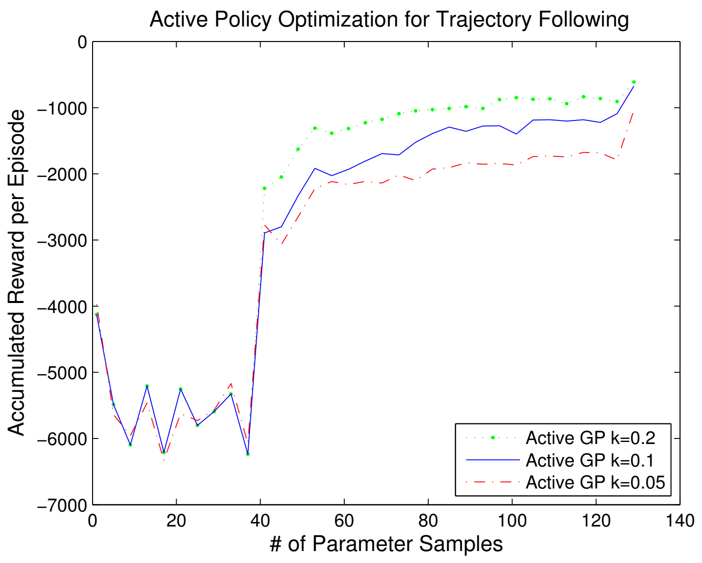


Figure 13: ***Active Policy Optimizer:*** *searching for the* 15 *policy parameters, and comparing different values for the GP kernel size* k*. We used the Expected Improvement function 3, and the three experiments are initialized with the the same set of* 30 *Latin hypercube samples. A total of* 20 *experimental runs were averaged for this plot.*

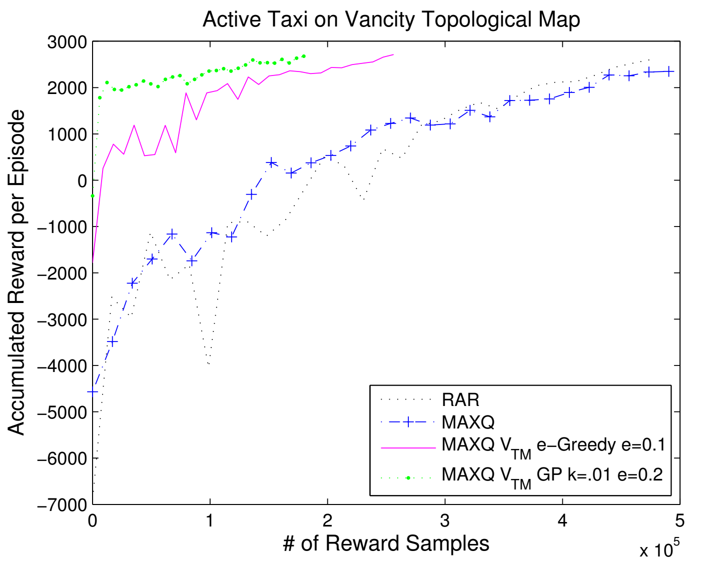
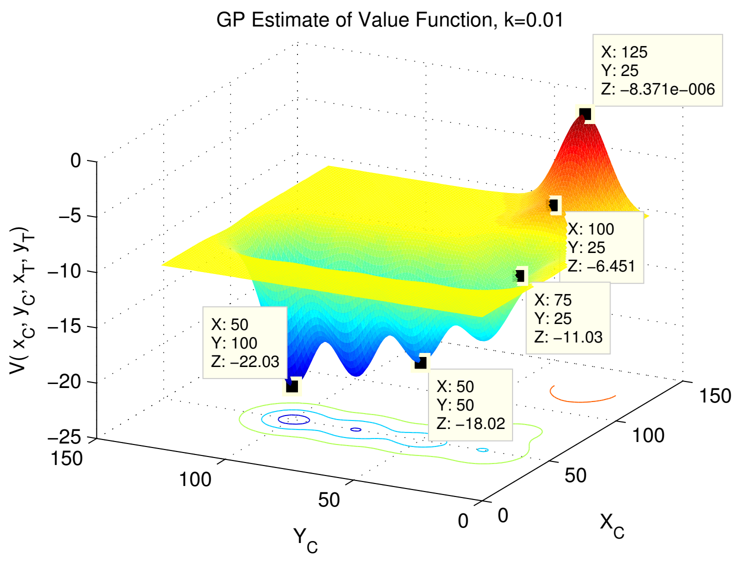
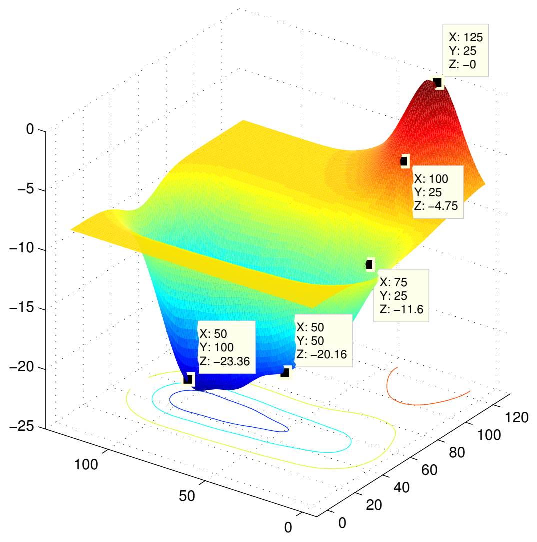


Figure 14: ***Parameterized*** V_{TM} ***vs. RAR and MAXQ:*** *These experiments compare the original Recursive Average Reward (RAR) and MAXQ (discounted reward) algorithms against the parameterized* $V_{TM}(TaxiLoc, \mathrm{WP})$ *path learner.*



(a) V_{TM} *GP* $k = 0.01$



(b) V_{TM} *GP* $k = 0.02$

Figure 15: ***GP Response Surface.*** *A small kernel value narrows the 'footprint' of an observation, whereas a larger k interpolates to the surrounding state space.*

However, Bayesian optimization is also a fairly recent addition to the machine learning community, and not yet extensively studied on user applications. Here, we wish to describe some of the shortcomings we have experienced in our work with Bayesian optimization, both as caveats and as opportunities for other researchers.

A particular issue is that the design of the prior is absolutely critical to efficient Bayesian optimization. Gaussian processes are not always the best or easiest solution, but even when they are, great care must be taken in the design of the kernel. In many cases, though, little is known about the objective function, and, of course, it is expensive to sample from (or we wouldn't need to use Bayesian optimization in the first place). The practical result is that in the absence of (expensive) data, either strong assumptions are made without certainty that they hold, or a weak prior must be used. It is also often unclear how to handle the trade-off between exploration and exploitation in the acquisition function. Too much exploration, and many iterations can go by without improvement. Too much exploitation leads to local maximization.

These problems are exacerbated as dimensionality is increased—more dimensions means more samples are required to cover the space, and more parameters and hyperparameters may need to be tuned, as well. In order to deal with this problem effectively, it may be necessary to do automatic feature selection, or assume independence and optimize each dimension individually.

Another limitation of Bayesian optimization is that the acquisition is currently both myopic and permits only a single sample per iteration. Looking forward to some horizon would be extremely valuable for reinforcement learning problems, as well as in trying to optimize within a known budget of future observations. Recent work [Garnett *et al.*, 2010b, Azimi *et al.*, 2011] has indicated very promising directions for this work to follow. Being able to efficiently select entire sets of samples to be labelled at each iteration would be a boon to design galleries and other batch-incremental problems.

Finally, there are many extensions that will need to be made to Bayesian optimization for particular applications—feature selection, time-varying models, censored data, heteroskedasticity, nonstationarity, non-Gaussian noise, *etc.* In many cases, these can be dealt with as extensions to the prior—in the case of Gaussian processes, for example, a rich body of literature exists in which such extensions have been proposed. However, these extensions need to take into account the adaptive and iterative nature of the optimization problem, which can vary from trivial to impossible.

Clearly, there is a lot of work to be done in Bayesian optimization, but we feel that the doors it opens make it worthwhile. It is our hope that as Bayesian optimization proves itself useful in the machine learning domain, the community will embrace the fascinating new problems and applications it opens up.

References

[Andre, 2003] D. Andre. *Programmable Reinforcement Learning Agents.* PhD thesis, University of California at Berkley, 2003.

[Audet *et al.*, 2000] C. Audet, J. Jr, Dennis, D. W. Moore, A. Booker, and P. D. Frank. Surrogate-model-based method for constrained optimization. In *AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2000.

[Azimi *et al.*, 2011] J. Azimi, A. Fern, and X. Z. Fern. Batch Bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems 24*, 2011.

[Barto and Mahadevan, 2003] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[Bartz-Beielstein *et al.*, 2005] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *Proc. CEC-05*, 2005.

[Baxter and Bartlett, 2001] J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[Bertsekas and Tsitsiklis, 1996] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[Betrò, 1991] B. Betrò. Bayesian methods in global optimization. *J. Global Optimization*, 1:1–14, 1991.

[Boyle, 2007] P. Boyle. *Gaussian Processes for Regression and Optimisation.* PhD thesis, Victoria University of Wellington, Wellington, New Zealand, 2007.

[Brochu *et al.*, 2007a] E. Brochu, N. de Freitas, and A. Ghosh. Active preference learning with discrete choice data. In *Advances in Neural Information Processing Systems 21*, 2007.

[Brochu *et al.*, 2007b] E. Brochu, A. Ghosh, and N. de Freitas. Preference galleries for material design. In *ACM SIGGRAPH 2007 Posters*, page 105, 2007.

[Brochu *et al.*, 2010a] E. Brochu, T. Brochu, and N. de Freitas. A Bayesian interactive optimization approach to procedural animation design. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2010.

[Brochu *et al.*, 2010b] E. Brochu, M. Hoffman, and N. de Freitas. Hedging strategies for Bayesian optimization. eprint arXiv:1009.5419, arXiv.org, September 2010.

[Busby, 2009] D. Busby. Hierarchical adaptive experimental design for Gaussian process emulators. *Reliability Engineering and System Safety*, 94(7):1183–1193, July 2009.

[Chu and Ghahramani, 2005a] W. Chu and Z. Ghahramani. Extensions of Gaussian processes for ranking: semi-supervised and active learning. In *Learning to Rank workshop at NIPS-18*, 2005.

[Chu and Ghahramani, 2005b] W. Chu and Z. Ghahramani. Preference learning with Gaussian processes. In *Proc. 22nd International Conf. on Machine Learning*, 2005.

[Cora, 2008] V. M. Cora. Model-based active learning in hierarchical policies. Master's thesis, University of British Columbia, Vancouver, Canada, April 2008.

[Cox and John, 1992] D. D. Cox and S. John. A statistical method for global optimization. In *Proc. IEEE Conference on Systems, Man and Cybernetics*, volume 2, pages 1241–1246, 1992.

[Cox and John, 1997] D. D. Cox and S. John. SDO: A statistical method for global optimization. In M. N. Alexandrov and M. Y. Hussaini, editors, *Multidisciplinary Design Optimization: State of the Art*, pages 315–329. SIAM, 1997.

[Dietterich, 2000] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[Diggle and Ribeiro, 2007] P. J. Diggle and P. J. Ribeiro. *Model-Based Geostatistics*. Springer Series in Statistics. Springer, 2007.

[Diggle *et al.*, 1998] P. J. Diggle, J. A. Tawn, and R. A. Moyeed. Model-based geostatistics. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 47(3):299–350, 1998.

[Elder, 1992] J. F. Elder, IV. Global R^d optimization when probes are expensive: The GROPE algorithm. In *Proc. IEEE International Conference on Systems, Man and Cybernetics*, 1992.

[Élő, 1978] Á. Élő. *The Rating of Chess Players: Past and Present.* Arco Publishing, New York, 1978.

[Frean and Boyle, 2008] M. Frean and P. Boyle. Using Gaussian processes to optimize expensive functions. In W. Wobcke and M. Zhang, editors, *AI 2008: Advances in Artificial Intelligence*, volume 5360 of *Lecture Notes in Computer Science*, pages 258–267. Springer Berlin / Heidelberg, 2008.

[Garnett *et al.*, 2010a] R. Garnett, M. Osborne, S. Reece, A. Rogers, and S. Roberts. Sequential Bayesian prediction in the presence of changepoints and faults. *The Computer Journal*, 2010.

[Garnett *et al.*, 2010b] R. Garnett, M. Osborne, and S. Roberts. Bayesian optimization for sensor set selection. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 209–219. ACM, 2010.

[Genton, 2001] M. G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, 2001.

[Ghavamzadeh, 2005] M. Ghavamzadeh. *Hierarchical Reinforcement Learning in Continuous State and Multi-agent Environments.* PhD thesis, University of Massachusetts Amherst, 2005.

[Ginsbourger *et al.*, 2008] D. Ginsbourger, R. Le Riche, and L. Carraro. A Multi-points Criterion for Deterministic Parallel Global Optimization based on Gaussian Processes. 2008.

[Goldberg *et al.*, 1998] P. W. Goldberg, C. K. I. Williamsn, and C. M. Bishop. Regression with input-dependent noise: A Gaussian process treatment. In *Advances in Neural Information Processing Systems 10*, 1998.

[Herbrich and Graepel, 2006] R. Herbrich and T. Graepel. Trueskill: A Bayesian skill rating system. Technical Report MSR-TR-2006-80, Microsoft Research, June 2006.

[Hinton and Salakhutdinov, 2006] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[Holmes and Held, 2006] C. Holmes and L. Held. Bayesian auxiliary variable models for binary and multinomial regression. *Bayesian Analysis*, 1(1):145–168, 2006.

[Huang *et al.*, 2006] D. Huang, T. T. Allen, W. I. Notz, and N. Zheng. Global optimization of stochastic black-box systems via sequential Kriging meta-models. *J. Global Optimization*, 34(3):441–466, March 2006.

[Hutter *et al.*, 2009] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy. An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proc. GECCO'09*, 2009.

[Hutter, 2009] F. Hutter. *Automating the Configuration of Algorithms for Solving Hard Computational Problems.* PhD thesis, University of British Columbia, Vancouver, Canada, August 2009.

[Jones *et al.*, 1993] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *J. Optimization Theory and Apps*, 79(1):157–181, 1993.

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

[Jones, 2001] D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *J. Global Optimization*, 21:345–383, 2001.

[Kahneman and Tversky, 1979] D. Kahneman and A. Tversky. Prospect theory: an analysis of decision making under risk. *Econometrica*, 47:263–291, 1979.

[Kapoor *et al.*, 2007] A. Kapoor, K. Grauman, R. Urtasun, and T. Sarrell. Active learning with Gaussian processes for object categorization. In *Proc. International Conference on Computer Vision (ICCV)*, 2007.

[Kendall, 1975] M. Kendall. *Rank Correlation Methods.* Griffin Ltd, 1975.

[Kingsley, 2006] D. C. Kingsley. Preference uncertainty, preference refinement and paired comparison choice experiments. Working Paper 06-06, Center for Economic Analysis, University of Colorado at Boulder, 2006. Dept. of Economics, University of Colorado.

[Krause *et al.*, 2008] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *J. Machine Learning Research*, 9:235–284, 2008.

[Krige, 1951] D. G. Krige. A statistical approach to some basic mine valuation problems on the Witwatersrand. *J. the Chemical, Metallurgical and Mining Soc. of South Africa*, 52(6), 1951.

[Kushner and Yin, 1997] H. J. Kushner and G. G. Yin. *Stochastic Approximation Algorithms and Applications.* Springer-Verlag, 1997.

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

[Lewis and Gale, 1994] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *Proc. ACM SIGIR Conference on Research and Development in Information Retreival*, 1994.

[Liberti and Maculan, 2006] L. Liberti and N. Maculan, editors. *Global Optimization: From Theory to Implementation.* Springer Nonconvex Optimization and Its Applications. Springer, 2006.

[Lizotte *et al.*, 2007] D. Lizotte, T. Wang, M. Bowling, and D. Schuurmans. Automatic gait optimization with Gaussian process regression. In *IJCAI*, 2007.

[Lizotte, 2008] D. Lizotte. *Practical Bayesian Optimization.* PhD thesis, University of Alberta, Edmonton, Alberta, Canada, 2008.

[Locatelli, 1997] M. Locatelli. Bayesian algorithms for one-dimensional global optimization. *J. Global Optimization*, 1997.

[Mackay, 1992] D. J. C. Mackay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[Marthi *et al.*, 2005] B. Marthi, D. Latham, S. Russell, and C. Guestrin. Concurrent hierarchical reinforcement learning. *In Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005.

[Martinez–Cantin *et al.*, 2006] R. Martinez–Cantin, N. de Freitas, and J. Castellanos. Analysis of particle methods for simultaneous robot localization and mapping and a new algorithm: Marginal-SLAM. In *Proc. IEEE International Conference on Robots and Automation*, 2006.

[Martinez–Cantin *et al.*, 2007] R. Martinez–Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos. Active policy learning for robot planning and exploration under uncertainty. *Robotics: Science and Systems (RSS)*, 2007.

[Martinez–Cantin *et al.*, 2009] R. Martinez–Cantin, N. de Freitas, E. Brochu, J. Castellanos, and A. Doucet. A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Autonomous Robots*, 27(2):93–103, 2009.

[Matérn, 1960] B. Matérn. *Spatial Variation.* Springer-Verlag, 2nd (1986) edition, 1960.

[Matheron, 1971] G. Matheron. The theory of regionalized variables and its applications. *Cahier du Centre de Morphologie Mathematique, Ecoles des Mines*, 1971.

[McFadden, 1980] D. McFadden. Econometric models for probabilistic choice among products. *Journal of Business*, 53(3):13–29, 1980.

[McFadden, 2001] D. McFadden. Economic choices. *The American Economic Review*, 91:351–378, 2001.

[Močkus *et al.*, 1978] J. Močkus, V. Tiesis, and A. Žilinskas. *Toward Global Optimization*, volume 2, chapter The Application of Bayesian Methods for Seeking the Extremum, pages 117–128. Elsevier, 1978.

[Močkus, 1982] J. Močkus. The Bayesian approach to global optimization. In R. Drenick and F. Kozin, editors, *System Modeling and Optimization*, volume 38, pages 473–481. Springer Berlin / Heidelberg, 1982.

[Močkus, 1994] J. Močkus. Application of Bayesian approach to numerical methods of global and stochastic optimization. *J. Global Optimization*, 4(4):347 – 365, 1994.

[Mongeau *et al.*, 1998] M. Mongeau, H. Karsenty, V. Rouzé, and J.-B. Hiriart-Urruty. Comparison of public-domain software for black-box global optimization. Technical Report LAO 98-01, Universite Paul Sabatier, Toulouse, France, 1998.

[Mosteller, 1951] F. Mosteller. Remarks on the method of paired comparisons: I. the least squares solution assuming equal standard deviations and equal correlations. *Psychometrika*, 16:3–9, 1951.

[Murray-Smith and Girard, 2001] R. Murray-Smith and A. Girard. Gaussian process priors with ARMA noise models. In *Irish Signals and Systems Conference*, 2001.

[Murray-Smith and Sbarbaro, 2002] R. Murray-Smith and D. Sbarbaro. Nonlinear adaptive control using non-parametric Gaussian process prior models. In *15th IFAC World Congress on Automatic Control*. Citeseer, 2002.

[Ng and Jordan, 2000] A. Y. Ng and M. I. Jordan. Pegasus: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence (UAI2000)*, 2000.

[O'Hagan, 1978] A. O'Hagan. On curve fitting and optimal design for regression. *Journal of the Royal Statistical Society B*, 40:1–42, 1978.

[Osborne *et al.*, 2010] M. Osborne, R. Garnett, and S. Roberts. Active data selection for sensor networks with faults and changepoints. In *IEEE International Conference on Advanced Information Networking and Applications*, 2010.

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

[Parr, 1998] R. E. Parr. *Hierarchical control and learning for markov decision processes*. PhD thesis, 1998. Chair-Stuart Russell.

[Payne *et al.*, 1993] J. W. Payne, J. R. Bettman, and E. J. Johnson. *The Adaptive Decision Maker*. Cambridge University Press, 1993.

[Poyiadjis *et al.*, 2005] G. Poyiadjis, A. Doucet, and S. S. Singh. Particle methods for optimal filter derivative: Application to parameter estimation. In *IEEE ICASSP*, pages 925–928, 2005.

[Press *et al.*, 2007] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[Rasmussen and Williams, 2006] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, Massachusetts, 2006.

[Sacks *et al.*, 1989] J. Sacks, W. J. Welch, T. J. Welch, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423, 1989.

[Santner *et al.*, 2003] T. J. Santner, B. Williams, and W. Notz. *The Design and Analysis of Computer Experiments*. Springer, 2003.

[Sasena, 2002] M. J. Sasena. *Flexibility and Efficiency Enhancement for Constrained Global Design Optimization with Kriging Approximations*. PhD thesis, University of Michigan, 2002.

[Schonlau, 1997] M. Schonlau. *Computer Experiments and Global Optimization*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1997.

[Settles, 2010] B. Settles. Active learning literature survey. Computer Science Technical Report 1648, University of Wisconsin-Madison, January 2010.

[Siegel and Castellan, 1988] S. Siegel and N. J. Castellan. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, 1988.

[Srinivas *et al.*, 2010] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. International Conference on Machine Learning (ICML)*, 2010.

[Stein, 1999] M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Series in Statistics. Springer, 1999.

[Stern, 1990] H. Stern. A continuum of paired comparison models. *Biometrika*, 77:265–273, 1990.

[Streltsov and Vakili, 1999] S. Streltsov and P. Vakili. A non-myopic utility function for statistical global optimization algorithms. *J. Global Optimization*, 14:283–298, 1999.

[Stuckman, 1988] B. Stuckman. A global search method for optimizing nonlinear systems. *IEEE Transactions on Systems, Man and Cybernetics*, 18(6):965–977, 1988.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Sutton *et al.*, 1999] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[Thurstone, 1927] L. Thurstone. A law of comparative judgement. *Psychological Review*, 34:273–286, 1927.

[Törn and Žilinskas, 1989] A. Törn and A. Žilinskas. *Global Optimization*. Springer-Verlag, 1989.

[Tversky and Kahneman, 1992] A. Tversky and D. Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *J. Risk and Uncertainty*, 5:297–323, 1992.

[Vasquez and Bect, 2008] E. Vasquez and J. Bect. On the convergence of the expected improvement algorithm. Technical Report arXiv:0712.3744v2, arXiv.org, Feb 2008.

[Williams *et al.*, 2000] B. J. Williams, T. J. Santner, and W. I. Notz. Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica*, 10:1133–1152, 2000.

[Wymann *et al.*, 2009] B. Wymann, C. Dimitrakakis, and C. Alexopoulos. The open racing car simulator (http://torcs.sourceforge.net/), 2009.

[Younes, 1989] L. Younes. Parameter estimation for imperfectly observed Gibbsian fields. *Prob. Theory and Rel. fields*, 82:625–645, 1989.

[Zhigljavsky and Žilinskas, 2008] A. Zhigljavsky and A. Žilinskas. *Stochastic Global Optimization*. Springer Optimization and Its Applications. Springer, 2008.

[Žilinskas and Žilinskas, 2002] A. Žilinskas and J. Žilinskas. Global optimization based on a statistical model and simplical partitioning. *Computers and Mathematics with Applications*, 44:957–967, 2002.

[Žilinskas, 1980] A. Žilinskas. *Lecture Notes in Control and Information Sciences*, chapter On the Use of Statistical Models of Multimodal Functions for the Construction of Optimization Algorithms. Number 23. Springer-Verlag, 1980.