Opportunity Cost in Bayesian Optimization

Jasper Snoek
University of Toronto
jasper@cs.toronto.edu

Hugo Larochelle
University of Sherbrooke
hugo.larochelle@usherbrooke.ca

Ryan Prescott Adams
Harvard University
rpa@seas.harvard.edu

Abstract

A major advantage of Bayesian optimization is that it generally requires fewer function evaluations than optimization methods that do not exploit the intrinsic uncertainty associated with the task. The ability to perform well with fewer evaluations of the target function makes the Bayesian approach to optimization particularly compelling when that target distribution is expensive to evaluate. The notion of *expense*, however, depends on the problem and may even depend on the location in the search space. For example, we may be under a time deadline and the experiments we wish to run may have varying duration, as when training neural networks or finding the hyperparameters of support vector machines. In this paper we develop a new idea for selecting experiments in this setting, that builds in information about the opportunity cost of some experiments over others. Specifically, we consider Bayesian optimization where 1) there are limited resources, 2) function evaluations vary in resource cost across the search space, and 3) the costs are unknown and must be learned.

1 Introduction

The optimization of machine learning models frequently involves careful tuning of hyperparameters. Unfortunately, however, this tuning is often a "black art", requiring expert experience, rules of thumb, or sometimes brute-force search. Bayesian optimization would seem to provide an elegant approach to this "meta-learning" problem, as the hyperparameter-specific learning may be an expensive procedure in time and other resources. Algorithms optimizing expected improvement (Mockus et al., 1978) and the Gaussian process upper confidence bound (Srinivas et al., 2010) are appealing in this setting as they have been shown to be efficient in the number of function evaluations required to find the global optimum of many multimodal black-box functions (Bull, 2011).

An important real-world caveat in hyperparameter learning and other problems, however, is that the cost of function evaluations may vary over the space. For example, in a neural network, the learning algorithm will be trained to convergence, but how long this takes may change with the learning rate, regularization strength, and number of hidden units. Even without considering duration, the advent of cloud computing makes it possible to quantify economically the cost of requiring large-memory machines for learning, changing the actual cost in dollars of an experiment with a different number of hidden units. Finally, network communication between processes and in loading data may also have direct costs, or may determine whether an experiment can be conducted using, e.g., a GPU.

Our framework for thinking about varying expenses under resource constraints is to model the *opportunity cost* associated with candidate experiments. We would like our Bayesian optimization algorithm to consider running a larger number of cheap experiments, when the cost-ignorant algorithm would otherwise only be able to run a few expensive ones. We examine two different ideas

along these lines. When there is only a single constrained resource (e.g., time to a deadline), we have found that a simple myopic variant of expected improvement performs well. As we will discuss, however, this approach does not generalize well when there are multiple resource constraints. For this more general case, we develop a new algorithm that attempts to directly compute an approximation to the opportunity cost.

As our goal is to develop effective meta-learning algorithms, we also address the case where the resource costs are unknown *a priori*. In keeping with the overall philosophy of Bayesian optimization, we learn these costs and represent their associated uncertainty, enabling us to make choices that incorporate our own ignorance appropriately. We use Gaussian process priors to model these cost functions.

2 Expected Improvement

Bayesian optimization refers to a Bayesian motivated method for seeking the extrema of expensive functions (Brochu et al., 2010). Intuitively, the Bayesian approach suggests that given a prior belief over the form of a function and a finite number of observations, one can make an educated guess about where the optima of such a function should be. Much of the Bayesian approach was introduced and developed by Mockus et al. (1978); Mockus (1994, 1989). In particular, Mockus et al. (1978) argued that when searching for the optima of a function, rather than assume this corresponds to the optima of the estimated function (i.e. the predictive mean in the Gaussian case) one should search where the *expected improvement* is greatest. Schonlau et al. (1998) derive an analytic form for the case where a single next point is selected while an approximate approach is suggested for the less myopic next n-point alternative.

2.1 Expected Improvement with a Deadline

Ginsbourger and Riche (2010) explored the scenario of optimization using the expected improvement criterion when the number of available function evaluations is fixed and known. They demonstrate that choosing the next experiment to run based on the standard myopic expected improvement algorithm is inferior to choosing it based on the joint expected improvement of the available function evaluations. Computing the joint EI, however, is intractable in general. Therefore, we explore a greedy algorithm and a monte carlo approximation.

Expected Improvement per Second We develop a baseline greedy approach where we modify the acquisition function such that the next point to be chosen is the one for which the expected improvement per second is highest. That is we greedily choose the experiment that will give us the most efficiency in terms of improvement over time.

Monte Carlo Multi-Step Myopic EI (MCMS) Rather than evaluate the joint EI of each possible subset of experiments that fits within the time horizon, which is intractable in general, we develop a Monte Carlo approximation to running multiple steps of myopic EI optimization. That is, we sample paths conditionally on each candidate point, recursively sampling function values for each experiment, computing EI and adding the top EI candidate until the total estimated time taken to evaluate the path exceeds the time deadline. The minimum function value observed along the path is taken to be the improvement achieved by following that path. The expected improvement is then taken to be the average improvement over multiple sampled paths. This algorithm is outlined in Algorithm 1.

3 Experiments

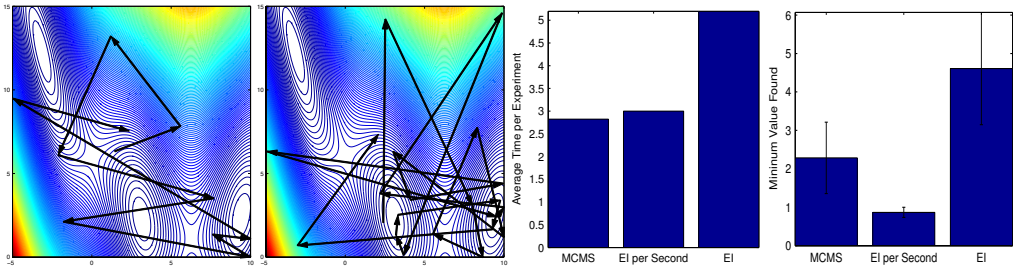
3.1 Simple Branin-Hoo Example

To illustrate the advantage of our approach, we first present a simple toy example. The Branin-Hoo function is a common benchmark for Bayesian optimization techniques (Jones, 2001) that is defined over $x \in \mathbb{R}^2$ where $0 \leq x_1 \leq 15$ and $-5 \leq x_2 \leq 15$. Rather than assume that each function evaluation is equally expensive, we assume that half of the search space is exactly ten times more expensive to evaluate than the other half, i.e. function evaluations in the expensive half, $x_2 < 2.5$,

Algorithm 1 A Monte Carlo Algorithm for computing the next experiment to run given a bounded amount of time:

$maxtot \leftarrow 0$
for $m = 1 \rightarrow M$ **do**
  $bestm \leftarrow max(y)$ [Initialize to current maximum.]
  $timeleft \leftarrow T$ [Initialize to total time left.]
  $z_{cur} \leftarrow z_j$ [Starting point is the next-step candidate.]
  while $timeleft > 0$ **do**
    Sample $y_{cur} = f(z_{cur})$ [Sample the function from the GP, given history.]
    if $y_{cur} > bestm$ **then**
      $bestm = y_{cur}$ [Update the maximum.]
    end if
    $timeleft = timeleft - g(z_{cur})$ [Subtract the runtime of this expt.]
    Update the GP posterior for this path.
    $z_{cur} = \arg\max_z(EI(z))$ [Choose next point with myopic EI.]
  end while
  $maxtot = maxtot + bestm$ [Accumulate for later sample average.]
end for
$\Psi(z_j) = maxtot/M$ [Divide to get average.]
Return $max(\Psi)$

take ten fantasy seconds rather than one. We further assume that there is a deadline of 50 seconds to complete the optimization. The halves are split in such a way that equally good optima exist in either half. Thus, with knowledge of the time it would take to evaluate each candidate point, an algorithm that is aware of the time required to evaluate each point should prefer running cheaper experiments and reach a better optimum than a time agnostic algorithm. Each algorithm was evaluated on this toy scenario and the results are presented in Figure 1. Notice that the MCMS algorithm is capable of running many more experiments in the same amount of time, and therefore is able to find a better optimum than standard EI.



(a) (b) (c) (d)

Figure 1: An example sequence of points evaluated within the 50 second deadline by (a) standard EI and (b) MCMS EI. Note that inputs less than 2.5 on the horizontal axis are ten times more expensive to evaluate. Thus, the time-aware algorithms spend most of their evaluations in the cheaper half. Each algorithm was used to optimize the Branin-Hoo function with fantasized time costs with a deadline of 50 seconds. (c) shows the average time taken per experiment by each algorithm and (d) shows the average minimum found within the deadline.

3.2 Multiple Kernel Learning with Support Vector Machines

A natural application for Bayesian optimization is the optimization of hyperparameters within multiple kernel support vector machines. Traditionally, the hyperparameters of support vector machines are optimized using cross-validation. For a small number of parameters a grid search over parameter values is feasible. However, in a multiple kernel learning setting, where potentially many kernel hyperparameters must be set, exhaustive grid search becomes computationally infeasible. In this example, expected improvement is used to to optimize the parameters of a support vector machine with multiple kernels. In this case, we use the sum of an rbf and a linear kernel, each scaled by a

Data Set	MCMS	EI/S	EI	Deadline(s)
Ionosphere	10.4 ± 0.74	9.40 ± 1.27	13.7 ± 0.96	10
w1a	2.36 ± 0.14	2.20 ± 0.08	2.61 ± 0.06	150
Australian	30.8 ± 1.50	30.0 ± 0.44	32.0 ± 0.04	250
Sonar	12.8 ± 0.19	13.0 ± 0.18	13.8 ± 0.21	2

Table 1: Minimum cross-validation error values in percent found by the optimization routine for various Bayesian optimization algorithms performing hyperparameter optimization on multiple kernel svms applied to standard UCI datasets.

scale parameter. The parameters to be optimized are kernel scale parameters, rbf kernel width, the slack penalty, C, and the convergence tolerance parameter. Experiments were conducted on four binary classification datasets from the UCI data repository, where we use EI to find the hyperparameter settings that minimize ten-fold cross validation error. Results are presented in Table 1 and curves demonstrating the average minumum function value found by each algorthm vs the amount of time are presented in Figure 2.

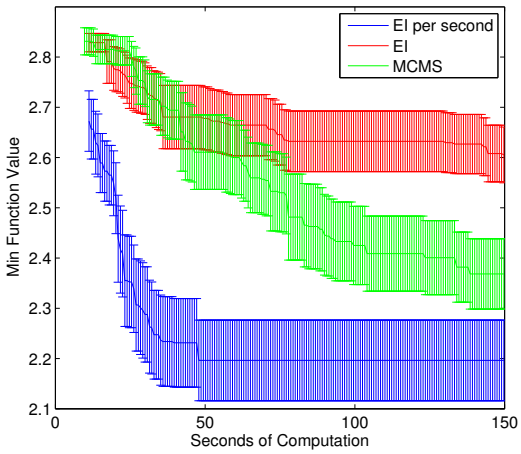


Figure 2: Example optimization curves for optimization performed on the w1a data demonstrating the minumum function values found by each algorithm as a function of time. In this case, the optimization deadline was chosen to be 150 seconds.

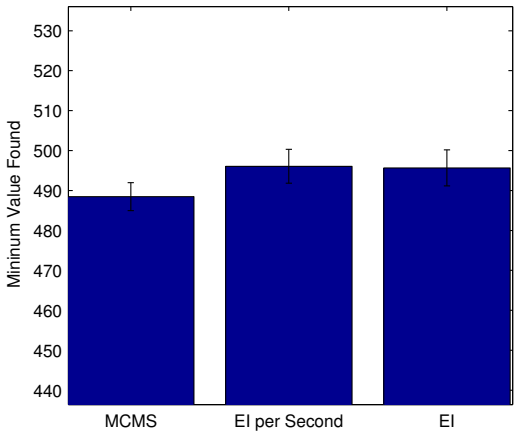


Figure 3: A comparison of the minimum error achieved before the deadline by the various algorithm on the neural network problem. The error values are averaged over twenty evaluations and the errorbars are standard error. In this case MCMS outperforms both algorithms.

3.3 Training a Neural Network

As a final experiment, we use Bayesian optimization to optimize the hyperparameters of a neural network. This scenario is borrowed from an undergraduate course assignment at the University of Toronto, where students are given neural network code and are asked to tune the hyperparameters to reach a validation classification error of 500. Thus, the idea is that this is a challenge for someone with introductory knowledge of neural networks. The parameters required to tune are the number of hidden units to use, the number of epochs of unsupervised pretraining, a weight decay for pretraining, the learning rate for stochastic gradient descent, the number of epochs of backpropagation, and the weight decay during backpropagation. All algorithms were run on this problem 20 times, with a deadline chosen such that the algorithms could run approximately between 10 to 40 evaluations, and the results are presented in Figure 3.

References

J Mockus, V Tiesis, and A Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2:117–129, 1978.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. International Conference on Machine Learning (ICML)*, 2010.

Adam D. Bull. Convergence rates of efficient global optimization algorithms. *JMLR*, (3-4):2879–2904, 2011.

Eric Brochu, Vlad M Cora, and Nando De Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *Arxiv preprint arXiv10122599*, page 49, 2010.

Jonas Mockus. Application of Bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347–365, 1994.

Jonas Mockus. *Bayesian Approach to Global Optimization*. Kluwer, Dordrecht, Netherlands, 1989.

M Schonlau, W J Welch, and D R Jones. Global versus local search in constrained optimization of computer models. *Lecture NotesMonograph Series*, 34:11–25, 1998.

David Ginsbourger and Rodolphe Riche. Towards gaussian process-based optimization with finite time horizon. In *Advances in Model-Oriented Design and Analysis*, Contributions to Statistics, pages 89–96. Physica-Verlag HD, 2010.

D.R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.