UNIVERSITY *of* TORONTO

Opportunity Cost in Bayesian Optimization

Jasper Snoek, Hugo Larochelle & Ryan Prescott Adams

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 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.

Opportunity Cost in Bayesian Optimization

- Myopic EI is efficient in the number of function evaluations required (Bull, 2011)
    - What if there are input dependent costs associated with evaluating the function?

- With a fixed budget we can consider the *opportunity cost* of evaluating a set of inputs
    - Evaluating joint EI is prohibitively expensive (Ginsbourger & Riche, 2010)
    - We develop a monte carlo approximation and a simple greedy algorithm that significantly improve on standard EI in this scenario

Expected Improvement with a Deadline

- We consider the case where running time is input dependent and we have a deadline
    - Many machine learning models exhibit this behavior – e.g. neural nets and SVMs

Expected Improvement per Second

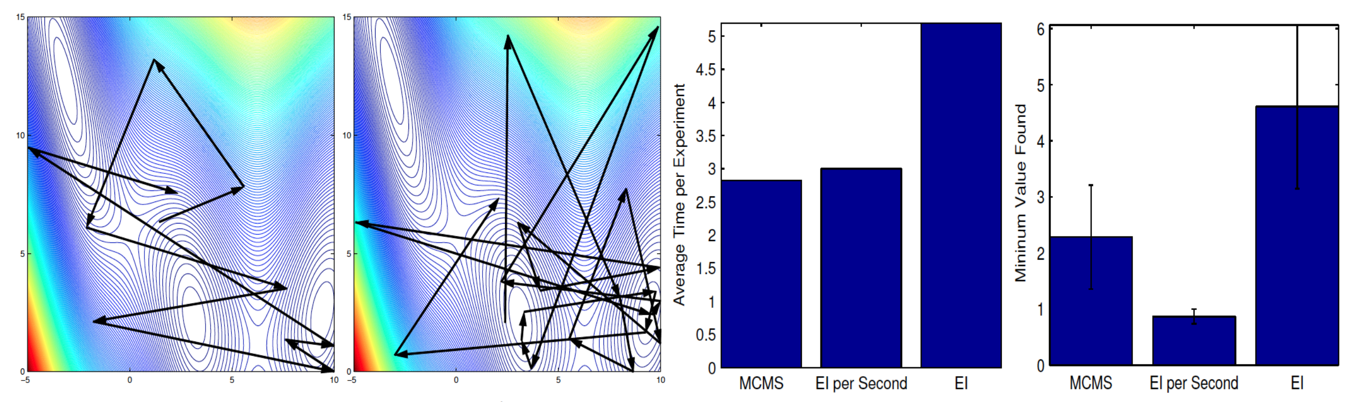
- A myopic baseline greedy approach where the next point to be chosen is the one for which the expected improvement per second is highest

Monte Carlo Multi-Step Myopic EI (MCMS)

- Rather than evaluate joint EI of each possible subset of experiments fitting within the deadline, we develop a Monte Carlo approximation to running multiple steps of myopic EI optimization

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

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



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

Figure 1: Each algorithm was used to optimize the Branin-Hoo function with fantasized time costs and a deadline of 50 seconds. (a) shows an example sequence of points evaluated within the deadline by standard EI and (b) MCMS EI. 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. (c) shows the average time taken per experiment by each algorithm and (d) shows the average minimum found within the deadline.

Simple Branin-Hoo Example

- The Branin-Hoo function is common Bayesian optimization benchmark
    - defined over $0 \leq x_1 \leq 15$ and $-5 \leq x_2 \leq 10$

- We assume that half of the search space ($x_2 < 2.5$) takes ten times longer (10 seconds) to evaluate than the other half (1 second) and we have a deadline of 50 seconds
    - It's easy to see that EI will not behave optimally in this setting
    - We assume the time is known (noiseless) and compare the behavior of EI vs our algorithms

- Results are shown in Figure 1
    - Clearly, one can find a better optimum in less time when taking the time into account

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, these are optimized using cross-validation
    - In a multiple kernel learning setting, however, exhaustive grid search becomes computationally infeasible

- We use EI to optimize the parameters of a support vector machine with multiple kernels (the sum of an rbf and linear kernel)
    - 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

- Results are presented in Table 1 and Figure 2
    - Under a deadline, one can find a better solution in the same time when taking the algorithm running time into account

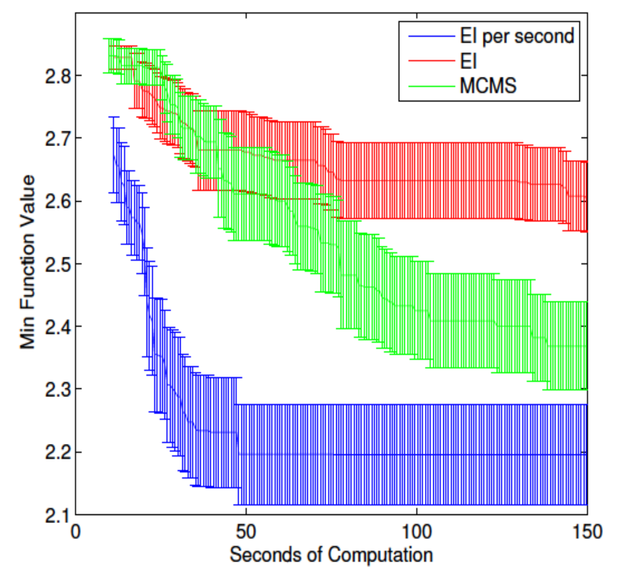


Figure 2: Example optimization curves for optimization performed on the w1a data demonstrating the minumum function values found by each algorithm as a function of time. In this case, the optimization deadline was chosen to be 150 seconds.

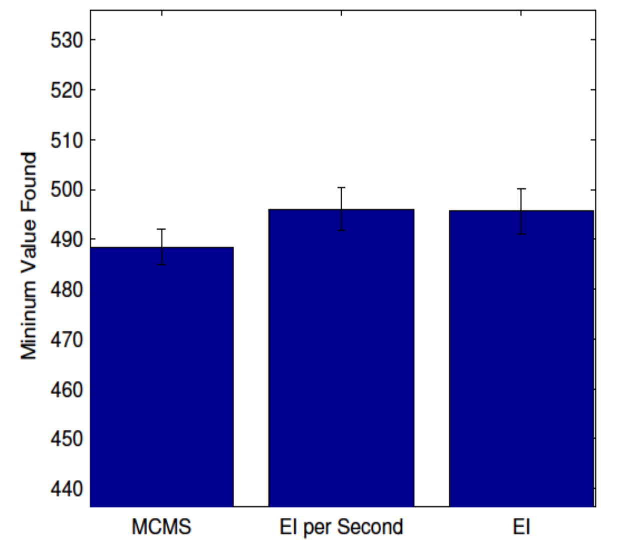


Figure 3: A comparison of the minimum error achieved before the deadline by the various algorithm on the neural network problem. The error values are averaged over twenty evaluations and the errorbars are standard error. In this case MCMS outperforms both algorithms.

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.

Training a Neural Network

- We use EI 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
    - This is challenging for a non-expert
    - We compare to EI and our algorithms
    - The parameters required to tune are the number of hidden units to use, the number of epochs of unsupervised RBM 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

- Results are presented in Figure 3
    - MCMS outperforms EI and EI per second

UNIVERSITÉ DE SHERBROOKE

HARVARD INTELLIGENT PROBABILISTIC SYSTEMS