Bayesian Optimisation in a Parallel Setting

Kim Ward

December 2019

This report looks to examine how the Bayesian Optimisation method of Expected Improvement looked at in Topic Sprint 1 generalises to a setting where multiple simultaneous function calls are allowed.

1 Notation and defaults

We are trying to minimise a deterministic, Lipschitz function $f : X \to \mathbf{R}$ that is expensive to evaluate, where $X \subset \mathbf{R}^n$ is a convex parameter set. We have q parallel cores to run function evaluations on. We assume all function evaluations take approximately the same time as each other and any noise in function evaluations is negligible.

To do this, we first of all pick q points giving a good spread across X and evaluate them all simultaneously, creating a set $S := \{(x, f(x)) : x \in X \text{ has been evaluated}\}$ of basic information and set $f^* = \min_{(x,f(x))\in S} f(x)$. We then choose the next set of q points according to some algorithm, compute, add that information to S, update f^*, and repeat until we have hit our budget for processing power. Our estimator for the minimum of f is our final computed value for f^*.

We assume a Gaussian Process prior for f, i.e. it is possible to write the posterior

$$f(x)|S \sim N(\mu_S(x), \sigma_S(x))$$

Where μ and σ can be calculated.

For the purposes of this report, we will be using the Expected Improvement (EI) acquisition function (and its generalisations) rather than PI, UCB, or any others. The EI acquisition function is a one-dimensional function given by:

$$EI(x) = \mathbf{E}_{f|S}[(f^* - f(x))^+]$$

It turns out to have a closed form in μ_S and σ_S. Our choice of next evaluation point is just the arg max of this function.

2 The problem with deterministics and q-Expected Improvement

With the information S that we have available, a deterministic 1-point algorithm such as Expected Improvement (EI) would give us only once place to next evaluate. Since we're not assuming that evaluations of f are particularly noisy, evaluating the same spot multiple times is of little value, so we can't just run the algorithm q times and be done with it. (In comparison, a stochastic algorithm such as Thompson Sampling would give us a different evaluation point each time we ran the algorithm, so we could run it q times with the same information set S without this problem).

The natural generalisation of EI to a q-dimensional setting is called the q-Expected Improvement (qEI) [reference]. This is a q-dimensional function given by

$$qEI(x_1, ..., x_q) = \mathbf{E}_{f|S}[(f^* - \min_{i \in \{1,...,q\}} f(x_i))^+]$$

Finding the arg max of this function then gives us q places to evaluate simultaneously. Unfortunately, the qEI has the serious disadvantages of not having a closed form (for $q > 2$) and quickly becoming computationally intractable for even moderate values of q.[1]

Instead, we turn to 1-point asynchronous methods. Instead of answering the question "Given S, which q points should I evaluate next?" as qEI does, they instead answer "Given S and the k points currently being evaluated that have not yet returned a result, which point should I evaluate next?". Finding q points to evaluate is then just a case of running the 1-point asynchronous method q times in quick succession.

This gives asynchronous methods another advantage: if evaluations of f take variable amounts of time, there is no need to wait for the whole batch of q evaluations, so machines do not spend time being idle.

All of the asynchronous methods considered in this report are generalisations of EI in the following way: if $k = 0$ i.e. no points are currently being evaluated, the recommended point is the same as would be recommended by EI given the current information set S. (Note that qEI does not have this property).

3 Kriging Believer

Kriging is a term for modelling a function using Gaussian processes. Given our information set S, the function μ_S acts as an estimate for f. The Kriging Believer (KB) algorithm pretends this estimate is the truth, using it as a surrogate for f when it comes to recommending more points to evaluate. [2]

More formally, we construct a belief set S^* as a combination of information and educated guesswork:

$$S^* := S \cup \{(x_i, \mu_S(x_i)) : 1 \leq i \leq k\}$$

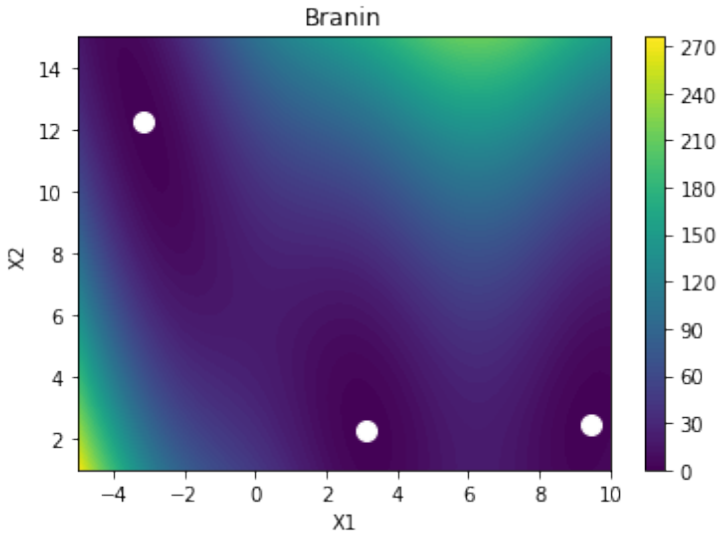

Figure 1: Plot of the Branin function showing the three global minima.

$$\alpha_{KB}(x) = EI_{S^*}(x)$$

Here, the x_i are the points currently being evaluated. Note that if $k = 0$ then $S^* = S$. We then use the EI algorithm with information set S^* (instead of S) to choose the next evaluation point.

KB as an algorithm is easy to understand and implement, and works fairly well in practice. However, if the estimate μ_S is bad, for instance if it is currently missing any hint of where the true minimum of f is, then the KB algorithm will reflect this and therefore fail to improve μ_S as well as other algorithms. KB focuses more on exploitation and can get stuck in its own beliefs because of this.

4 Constant Liar

Constant Liar (CL) is a variant of KB that focuses more on exploration. It again uses EI applied to a belief set S^*, but sets aside truth for convenience by pretending that every point currently being evaluated is some constant l that depends only on S and not the point itself.

$$S^* := S \cup \{(x_i, l) : 1 \leq i \leq k\}$$

$$\alpha_{CL}(x) = EI_{S^*}(x)$$

Possible sensible choices for l are the minimum, maximum, or mean of the evaluations in S. All three of them serve the same purpose, which is to heavily discourage lots of speculative point evaluations in the same area by making the

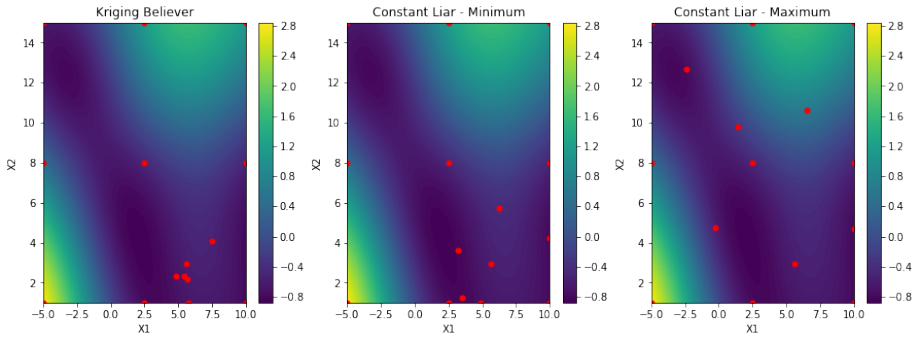

Figure 2: Plot of recommended points for the first step in a parallel Bayesian optimisation over a plot of the acquisition function.

function appear to be flat there. Choosing maximum discourages nearby points more heavily than minimum.

On the two-dimensional Branin function with initial startpoints a 9-point regular grid and a batch size of 6 points to recommend, the KB method recommends points close to each other, whereas CL encourages exploration, recommending points further afield - see Figure 2. Using the maximum instead of the minimum in CL causes even more exploratory recommendations.

5 Local Penalisation

Whereas KB and CL reformulate the Gaussian process using speculative data, Local Penalisation (LP) modifies the Expected Improvement function directly in order to penalise looking in areas "close" to points currently under evaluation, which it calls exclusion zones.

Given each point x_i currently being evaluated, LP constructs a function $\phi_{x_i} : X \to [0, 1]$ of the following form:

- $\phi_{x_i}(x_i) = 0$
- $\phi_{x_i}(x)$ is increasing in $||x - x_i||$
- Outside of the exclusion zone for x_i, $\phi_{x_i}(x)$ is close to 1

Exactly what counts as an exclusion zone is complicated and depends on an estimate of how Lipschitz the function f is - wigglier functions have smaller exclusion zones. Also, points closer to the estimated minimum for f have smaller exclusion zones around them. A proposed form for ϕ_{x_i} is available in [3].

The modified EI to be maximised is then:

$$\alpha_{LP}(x) := EI(x) \prod_{i=1}^{k} \phi_{x_i}(x)$$

Since $EI(x)$ is always positive, the ϕ_{x_i} functions always act as penalties.

One advantage of LP over all other methods here is that it does not need to reformulate the Gaussian process every time a point begins to be evaluated, so if the function evaluations are done in batches rather than asynchronously there is 1 reformulation (rather than q reformulations) per batch. For high-dimensional Gaussian processes this can be a significant time saving, as formulating a Gaussian process is $O(n^3)$ in the dimension n due to needing to solve a system of linear equations.

6 Monte Carlo Acquisition

Whereas KB took on faith the most likely value for points being evaluated, Monte Carlo Acquisition (MCA) does much the same thing from a Bayesian perspective and integrates out the uncertainty as best as possible. We have the acquisition function

$$\alpha_{MCA}(x) := \int_{\mathbf{y}} EI_{S\cup\{(x_1,y_1),..,(x_k,y_k)\}}(x)P(y_1,..,y_k|x_1,..,x_k,S)d\mathbf{y}$$

This integral cannot be directly evaluated, but can be simulated using Monte Carlo methods as the $\mathbf{y}$ form a multivariate Gaussian distribution. [4] While this is computationally more intensive than the other methods considered in this report, the extra uncertainty taken into account can cause a real difference in MCA's recommendations compared to KB's, and for f sufficiently costly the extra computation time needed to use MCA can be negligible.

7 Comparisons

Using the package GPyOpt in Python, I altered the package to implement the KB and CL methods and then made comparisons of the three methods KB, CL-min, LP with a batch size of 5 and the EI method (with a batch size of 1) for a total of 10 evaluations (2 batches for the parallel methods) on the 2-dimensional Branin function. I repeated each experiment 20 times and took averages and standard deviations. The results can be shown in Figure 3.

While the standard EI method performed the best due to the lack of parallelism, all methods behaved well and converged quickly on an optimum. The KB method took longest to complete a good initial exploration.

8 Conclusions

When you have parallel computing resources at your disposal, using a parallel Bayesian Optimisation method is worthwhile. Using the hard-to-compute theoretical generalisation (q-EI) is not necessary to achieve a worthwhile result.

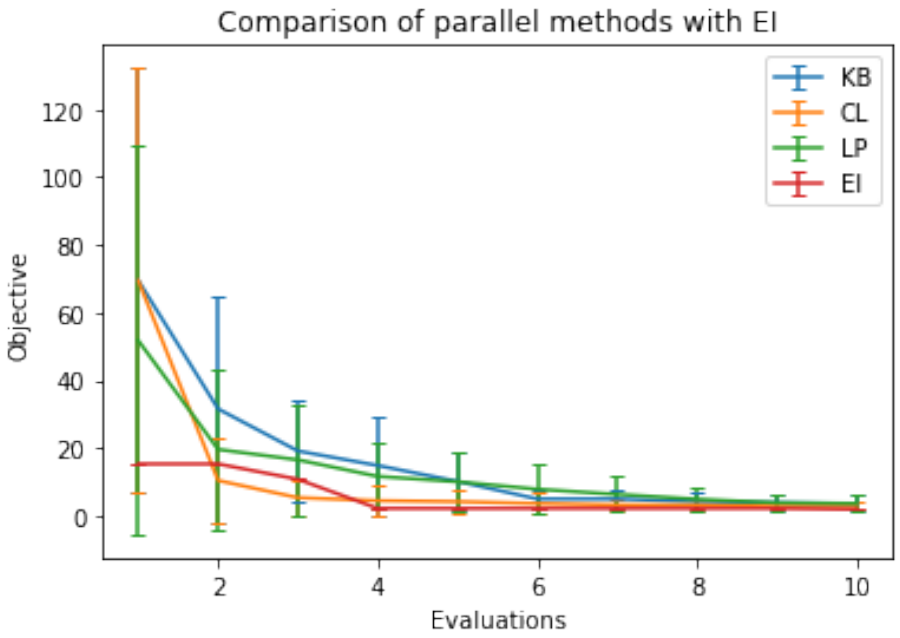

Figure 3: Plot of mean ± standard deviation for KB, CL-min, LP, and EI.

References

[1] `https://arxiv.org/pdf/1602.05149.pdf` *Parallel Bayesian Global Optimization of Expensive Functions*. Wang et. al, 2016 q-EI

[2] `http://www.cs.ubc.ca/labs/beta/EARG/stack/2010_CI_Ginsbourger-ParallelKriging.pdf` *Computational Intelligence In Expensive Optimization Problems*. Ginsbourger et al, 2010 KB and CL methods

[3] `https://arxiv.org/pdf/1505.08052v4.pdf` *Batch Bayesian Optimization via Local Penalization*. Gonzalez. et al, 2015 Local Penalisation

[4] `https://arxiv.org/pdf/1206.2944.pdf` *Practical Bayesian Optimisation of Machine Learning Algorithms*. Snoek et. al, 2012 Monte Carlo Acquisition

[5] `https://sheffieldml.github.io/GPyOpt/` *GPyOpt, a Bayesian Optimisation library for Python*. Code implementing the LP method.

[6] `https://github.com/KimGTWard/MRes` *My GitHub page, where the code used to produce this report can be found.*