Modular mechanisms for Bayesian optimization

Matthew W. Hoffman
mwh30@cam.ac.uk
University of Oxford, United Kingdom

Bobak Shahriari
bshahr@cs.ubc.ca
University of British Columbia, Canada

Abstract

The design of methods for Bayesian optimization involves a great number of choices that are often implicit in the overall algorithm design. In this work we argue for a modular approach to Bayesian optimization and present a Python implementation, `pybo`, that allows us to easily vary these choices. In particular this includes selection of the acquisition function, kernel, and hyperpriors as well as less-discussed components such as the recommendation and initialization strategies. Ultimately this approach provides us a straightforward mechanism to examine the effect of each choice both individually and in combination.

1 Introduction

Bayesian optimization is fast becoming a significant sub-topic in machine learning, particularly when applied to the problem of hyperparameter tuning. However, as more methods for Bayesian optimization are developed it becomes increasingly important to properly compare these techniques and analyze their various strengths and weaknesses. See, for example, the recent work of Eggensperger et al. which examines the performance of three packages widely-used for hyperparameter optimization [4]. One downside of this approach, though, is the way in which this earlier work treats each method as a single, monolithic optimization technique.

Modern approaches for Bayesian optimization are instead built on a number of different modular components. Minimally these include

- any initial allocation of points;
- the choice of infill or acquisition function;
- the method used to optimize the acquisition function;
- the choice of posterior model—for Gaussian process (GP) models this includes the choice of kernel and likelihood models; and
- the method of final recommendation (i.e. the point deemed optimal).

These choices are also often parameterized, which greatly expands the number of possible components. Additionally, for GP posteriors it is well known that a single setting of hyperparameters is prone to overfitting given small amounts of data [17]. This introduces additional components corresponding to (i) the choice of hyperpriors and (ii) the method used for hyperparameter marginalization. It is also often the case that no single selection of these components is optimal even within the same run [9], although we will not elaborate on this problem here.

Instead, in this work we propose a modular approach for Bayesian optimization which we have implemented as a Python library which we call `pybo`. In `pybo` each of the components outlined above can easily be modified or replaced in order to experiment with different combinations. This in turn allows us to examine the effect on performance as sets of components are varied.

Algorithm 1 Modular Bayesian Optimization

Input: a black-box function $\tilde{f} : \mathcal{X} \to \mathcal{Y}$

1: $\{\mathbf{x}_1, \ldots, \mathbf{x}_i\} \leftarrow \textsc{InitSamples}(\mathcal{X})$ *initial queries which supplement the policy*
2: $\mathcal{M} \leftarrow \textsc{InitModel}(\{\mathbf{x}_1, \ldots, \mathbf{x}_i\})$ *the prior/posterior model*
3: **for** $n = i + 1, \ldots$ **do**
4: $\quad \alpha_n \leftarrow \textsc{GetPolicy}(\mathcal{M}, n)$ *the acquisition function*
5: $\quad \mathbf{x}_n \leftarrow \textsc{Optimize}(\alpha_n)$ *optimize the acquisition function*
6: $\quad y_n \leftarrow \tilde{f}(\mathbf{x}_n)$
7: $\quad \mathcal{M} \leftarrow \textsc{AddData}(\mathcal{M}, \mathbf{x}_n, y_n)$ *update the model given new data, possibly marginalizing over any hyperparameters*
8: **end for**
9: **return** $\textsc{Recommend}(\mathcal{M})$ *return the recommended optimizer*

2 Modular Bayesian optimization

In the setting of Bayesian optimization we are interested in solving optimization problems of the form $\mathbf{x}_\star = \arg\max_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$ via queries to some noisy, black-box function $\tilde{f}$. In this section we outline our modular approach to this problem. General pseudocode for this approach is given by Algorithm 1 which takes as input $\tilde{f}$ and returns a recommendation for $\mathbf{x}_\star$.

In this work we assume a probabilistic approach to global optimization utilizing a model—initialized on Line 2—which sequentially constructs a Bayesian posterior of the function f. The model, denoted $\mathcal{M}$ in the pseudocode, must provide a consistent interface to enable its use in a wide variety of acquisition/recommendation strategies. These include

- $\textsc{GetQuantile}(\mathcal{M}, \mathbf{x}, q)$ for evaluating the marginal posterior quantiles at a given input;
- $\textsc{GetMoments}(\mathcal{M}, \mathbf{x})$ for evaluating the marginal posterior moments at a given input;
- $\textsc{Sample}(\mathcal{M})$ for sampling a function from the posterior—for non-parametric models this can be performed either via lazy-evaluation or a closed-form approximation [see 8, 16];
- $\textsc{AddData}(\mathcal{M}, \mathbf{x}, y)$ is used after each new observation.

In `pybo` we assume a GP model consisting of a likelihood, kernel, mean, and parameters for each associated sub-component, although alternative models could easily be incorporated. Sparse GP approximations [15] are also implemented for efficiency. Hyperparameters can be fixed or a *meta model* can be used which offers the same external interface described above, but marginalizes a parameterized model given a set of hyperpriors. Meta models in `pybo` are currently implemented for MCMC and SMC using this interface, but it also easily allows for arbitrary methods of integration such as the marginal GP [6]. Kernels can be given as arbitrary sums/products of base kernel objects (e.g. Matern, squared-exponential). Finally, here we assume Gaussian likelihood, although the facility exists to replace these with non-Gaussian models (e.g. Student's t, logistic).

Before initializing the model it is also often desirable to perform a separate initial sampling of the state-space, as in Line 1 of the pseudocode. For example, the EGO algorithm of Jones et al. recommends an initial design using Latin hypercube sampling [12]. Other approaches include random sampling (uniform or otherwise) or quasi-random sampling via Sobol or Halton sequences [3].

Given an initial model we can then proceed to generate a policy for selecting the next query location. This is done in line 4 of Algorithm 1 which constructs an index policy or acquisition function given the current posterior and the number of past queries. The dependence of the policy on n is important for approaches such as GP-UCB [18] or many models that are expected to run indefinitely and aim for low regret. Other possible implementations of this component include the probability of improvement (PI) [13], expected improvement (EI) [14, 12], Thompson sampling [19], or mixtures of different acquisitions [9, 16]. See also [16] for an efficient implementation of Thompson sampling in continuous models.

Given an acquisition function α_n the next query location is selected as the result of $\textsc{Optimize}(\alpha_n)$. Implementations of this component in `pybo` include global optimization techniques such as DIRECT [11] or multi-restart versions of local-optimization techniques such as L-BFGS-B [2]. Alter-

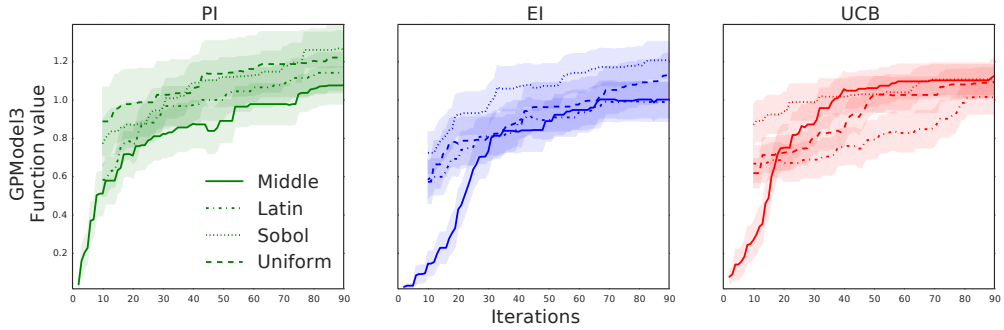

Figure 1: Effect of initialization on optimizing sample functions.

natively this component may initialize a datastructure to be used between repeated invocations of the optimization method—e.g. a dynamically growing grid [17] or tree [20].

Finally, after N iterations a recommendation must be returned. A common practice is to return the best point observed, however in order to accomodate noisy outputs we can also return those with the highest expected value. Alternatively, rather than restricting ourselves to the observed inputs we can consider maximizing the posterior mean over the entire input space. Additionally, while in this work we only consider halting after a fixed horizon N, stopping criterion based on the confidence of the recommendation can also be easily incorporated. See for example the discussion in [10] based on work of [5].

3 Experiments

In this section we vary a subset of the components used in Bayesian optimization and view the effect this variation has on performance. The experiments given in this section provide just a limited overview of the types of experiments that can be performed with `pybo`. Varying these components can be done by passing keyword arguments to a solution method called `solve_bayesopt` as follows

```
gp = ExactGP(likelihood=Gaussian(sigma),
             kernel=SE(ell, sf2))
xstar, _ = solve_bayesopt(f, bounds, model=gp, init='latin',
                          policy='ucb', solver='lbfgs',
                          recommender='latent')
```

where `gp` defines a prior model and implementing the model interface defined earlier. Here the hyperparameters are assumed fixed, but marginalization can also be performed using additional `inference` and `priors` keywords. Note, however, that reasonable defaults exist for all of these components such that `solve_bayesopt(f, bounds)` can be called.

We first consider the effect of initialization. The initializations considered include (quasi-) random samplings of the space and an approach which samples a single point in the midpoint of the bounded space. In Figure 1 this behavior is shown across 3 different acqusition functions, while optimizing functions sampled from a known GP prior. Here we see that for the for the PI and EI acquisition functions a Sobol sequence initialization tends to give the best performance, however for GPUCB the midpoint initialization is also competitive.

Next, we consider the effect of kernel choice on two global optimization test functions, namely Branin and Gramacy [see 1, 7]. Here we consider 4 ARD kernels: three from the Matérn family and the squared-exponential. Results are shown in Figure 2. Note that MCMC was used to integrate over the hyperparameters of these kernels, thus controlling the effect of the ARD parameters. Here we see that for the smoother Branin function the Matern5 and squared-exponential perform best, however on the Gramacy function the comparatively spikier Matérn1 exhibits better performance.

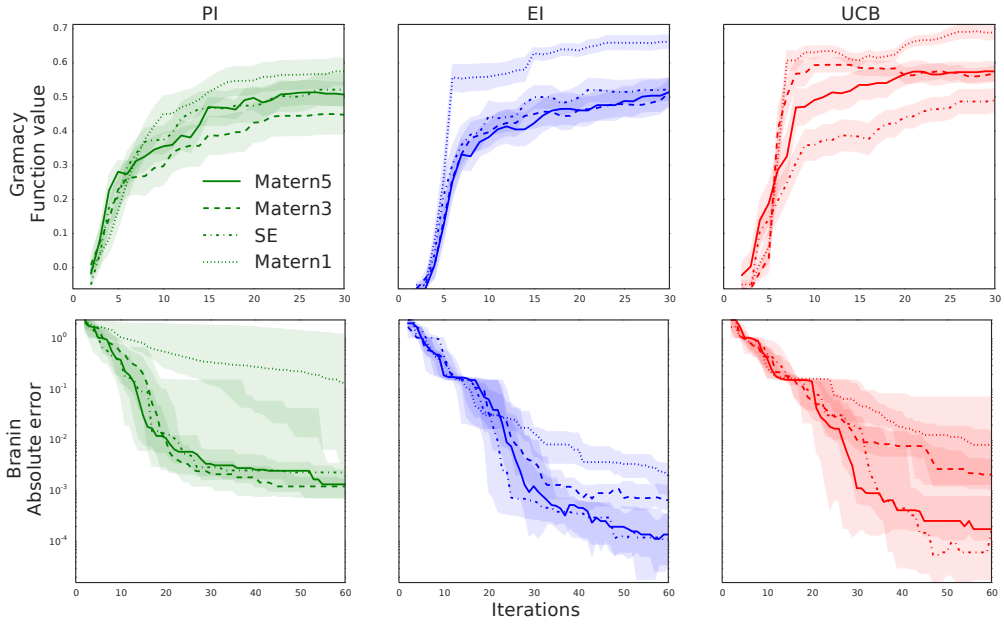

Figure 2: Effect of kernel choice on the Gramacy (top) and Branin (bottom) test functions.

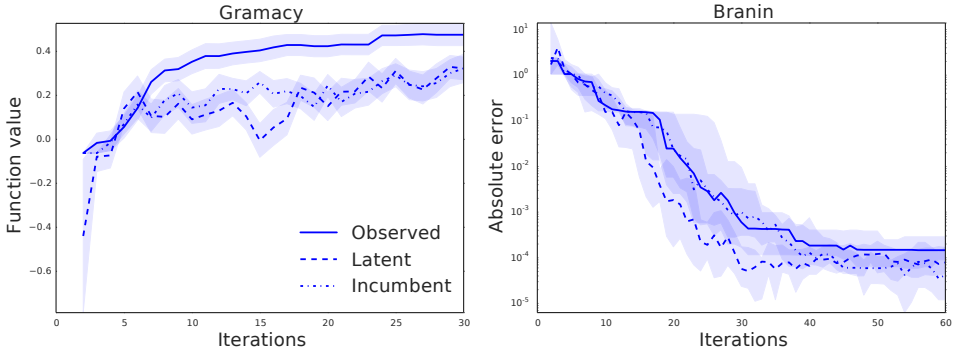

Figure 3: Effect of recommendation strategy on the Gramacy (left) and Branin (right) test functions.

Finally, we consider the effect of recommendation strategy on the same functions. The different strategies correspond to whether the latent posterior mean is optimized, or if either the highest incumbent or observation is returned. Figure 3 shows results with the EI acquisition function. Surprisingly, here we see that on the more multi-modal problem (Gramacy) returning the highest observation gives better performance, whereas better results are obtained by optimizing the latent function on Branin.

4 Conclusion

In this work we describe a modular approach for Bayesian optimization, and `pybo`, a Python package which facilitates comparing arbitrary combinations of these modular components. We also briefly examine a number of variations on its use as a mechanism for evaluating algorithmic approaches for probabilistic global optimization. Further work with this approach involves a more detailed study of the various combinations of sub-components for Bayesian optimization, which are necessarily outside the scope of this work. Finally, we see modular approaches of this kind as a step towards more fully automating this procedure—once we can evaluate the efficacy of different combinations we can begin the process of algorithmically selecting between these components.

References

[1] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Technical Report UBC TR-2009-23 and arXiv:1012.2599v1, Dept. of Computer Science, University of British Columbia, 2009.

[2] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[3] J. Dick, F. Y. Kuo, and I. H. Sloan. High-dimensional integration: The quasi-Monte Carlo way. *Acta Numerica*, 22:133–288, 2013.

[4] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *the NIPS Workshop on Bayesian Optimization*, 2013.

[5] V. Gabillon, M. Ghavamzadeh, and A. Lazaric. Best arm identification: A unified approach to fixed budget and fixed confidence. In *Neural Information Processing Systems*, 2012.

[6] R. Garnett, M. A. Osborne, and P. Hennig. Active learning of linear embeddings for Gaussian processes. In *Uncertainty in Artificial Intelligence*, 2014.

[7] R. B. Gramacy and H. K. Lee. Cases for the nugget in modeling computer experiments. *Statistics and Computing*, 22(3):713–722, 2012.

[8] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Neural Information Processing Systems*, 2014.

[9] M. W. Hoffman, E. Brochu, and N. de Freitas. Portfolio allocation for Bayesian optimization. In *Uncertainty in Artificial Intelligence*, pages 327–336, 2011.

[10] M. W. Hoffman, B. Shahriari, and N. de Freitas. On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning. In *the International Conference on Artificial Intelligence and Statistics*, pages 365–374, 2014.

[11] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

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

[13] H. Kushner. A new method of locating the maximum of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86, 1964.

[14] J. Močkus, V. Tiesis, and A. Žilinskas. The application of Bayesian methods for seeking the extremum. In L. Dixon and G. Szego, editors, *Toward Global Optimization*, volume 2. Elsevier, 1978.

[15] J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959, 2005.

[16] B. Shahriari, Z. Wang, M. W. Hoffman, A. Bouchard-Côté, and N. de Freitas. An entropy search portfolio for bayesian optimization. arXiv:1406.4625, 2014.

[17] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, pages 2960–2968, 2012.

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

[19] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

[20] Z. Wang, B. Shakibi, L. Jin, and N. de Freitas. Bayesian multi-scale optimistic optimization. In *the International Conference on Artificial Intelligence and Statistics*, 2014.