Budgeted batch Bayesian optimization

Budgeted Batch Bayesian Optimization

Vu Nguyen, Santu Rana, Sunil K Gupta, Cheng Li and Svetha Venkatesh
Deakin University, Australia
{v.nguyen,santu.rana,sunil.gupta,cheng.l,svetha.venkatesh}@deakin.edu.au

***Abstract*—Parameter settings profoundly impact the performance of machine learning algorithms and laboratory experiments. The classical trial-error methods are exponentially expensive in large parameter spaces, and Bayesian optimization (BO) offers an elegant alternative for global optimization of black box functions. In situations where the functions can be evaluated at multiple points simultaneously, batch Bayesian optimization is used. Current batch BO approaches are restrictive in fixing the number of evaluations per batch, and this can be wasteful when the number of specified evaluations is larger than the number of real maxima in the underlying acquisition function. We present the budgeted batch Bayesian optimization (B3O) for hyper-parameter tuning and experimental design - we identify the appropriate batch size for each iteration in an elegant way. In particular, we use the infinite Gaussian mixture model (IGMM) for automatically identifying the number of peaks in the underlying acquisition functions. We solve the intractability of estimating the IGMM directly from the acquisition function by formulating the batch generalized slice sampling to efficiently draw samples from the acquisition function. We perform extensive experiments for benchmark functions and two real world applications - machine learning hyper-parameter tuning and experimental design for alloy hardening. We show empirically that the proposed B3O outperforms the existing fixed batch BO approaches in finding the optimum whilst requiring a fewer number of evaluations, thus saving cost and time.**

***Index Terms*—batch Bayesian optimization, parallel global optimization, hyper-parameter tuning, experimental design.**

I. Introduction

Global optimization is fundamental to diverse real-world problems where parameter settings and design choices are pivotal - as an example, in algorithm performance (deep learning networks [3]) or quality of the products (chemical processes or engineering design [20]). This requires us to find the global maximum of a unknown and expensive objective function. The challenge is to find the maximum of such expensive objective functions in few sequential queries to minimize time and cost. Bayesian optimization (BO) offers an elegant solution to optimize such expensive, black-box functions f. Instead of optimizing the real function, BO utilizes a cheaper to evaluate surrogate function, called the acquisition function, to suggest the next point by balancing exploitation (knowledge of what has been observed) and exploration (where a function has not been investigated). Then, after evaluating the objective function at the suggested point, a Gaussian process is updated and thus a profile of the mean and uncertainty of the exploration space is gradually built up.

Most selection strategies in BO are sequential, wherein only one experiment is tested at a time [11], [5] - the experiment selection at each iteration is optimized by using the available observed information. However, such methods are "experiment inefficient" - for examples, many alloy samples can be placed in an oven simultaneously (for testing the alloy quality), or several machine learning algorithms can be run in parallel using multiple cores. This motivates batch Bayesian optimization algorithms that select multiple experiments, a batch of q experiments, at each iteration for evaluation.

Existing batch BO approaches are mostly greedy, sequentially visiting all the maxima of the acquisition function. This is not optimal as after the "real" maxima in the function are found, noisy points will get added simply to complete the batch. Moreover, the existing batch approaches rely on a predefined batch size. Fixed batch size approaches can restrict the flexibility in finding the global solution and require unnecessary evaluations or miss evaluation of important points. If we over-specify the batch size, we waste resources in evaluating redundant points. In contrast, if we under-specify the batch size, we may miss important points that could potentially be the optimal solution of f.

The best solution is to fill each batch *flexibly*, but the challenge is to know exactly *how many maxima are there in each batch*. Our key contribution is to solve this problem - by estimating the number of maxima in the acquisition function at each round. The solution comes from our intuition that the multiple peaks in the acquisition function can be approximated by a mixture of Gaussians - the means of the Gaussian correspond to the underlying peaks. However, since the number of underlying peaks is unknown, we utilize the infinite Gaussian mixture model (IGMM) [14] to learn the unknown number of peaks. Since fitting the IGMM directly to the acquisition function is intractable, we present an efficient batch generalized slice sampler approach to draw samples from acquisition function which are then used to learn the IGMM. Although the IGMM and the slice sampler have existed for more than a decade, the idea of making use of these techniques for batch Bayesian optimization is novel.

In the experiments, we first validate our methods in 8 functions and compare it against 9 baselines. We further demonstrate the algorithm on hyperparameter tuning for machine learning algorithms and a real world experimental design in which we formulate the aging process required for producing the desired hardness in an Aluminum-Scandium alloy. We show that our method produces the highest hardness using the fewest number of evaluations. Our result is significant as searching for the best aging process is costly and making this process efficient results in cost and time savings.

II. Preliminary

A. Bayesian optimization

We assume that f is a black-box function, that is, its form is unknown and further it is expensive to evaluate. Formally, let $f : \mathcal{X} \to \mathcal{R}$ be a well behaved function defined on a compact subset $\mathcal{X} \subseteq \mathcal{R}^D$. Our goal is solving the optimization problem

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}}\, f(\boldsymbol{x}). \tag{1}$$

Bayesian optimization (BO) makes a series of evaluations $\boldsymbol{x}_1, ..., \boldsymbol{x}_T$ of f such that the maximum of f is found in the fewest iterations [5], [18], [17]. BO reasons about f by building a Gaussian process through evaluations [15]. This flexible distribution allows us to associate a normally distributed random variable at every point in the continuous input space. Formally, a GP is given by $f(x) \sim GP(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$, where m is the mean, and $k(\mathbf{x}, \mathbf{x}')$ contains the covariance of any two observations. Using the standard calculation, we get the predictive distribution of the new observation that also follows a Gaussian distribution [15] - its mean and variance are given by: $\mu(\boldsymbol{x}_{N+1}) = \mathbf{k}_* \mathbf{K}^{-1} \mathbf{y}$ and $\sigma^2(\boldsymbol{x}_{N+1}) = k_{**} - \mathbf{k}_* \mathbf{K}^{-1} \mathbf{k}_*^T$.

Acquisition functions: The acquisition function acts as a surrogate that determines which point should be selected next $\boldsymbol{x}^*_{t+1} = \operatorname{argmax}\alpha_t(\boldsymbol{x})$. In this auxiliary maximization problem, the objective is known and easy to evaluate and can be easily carried out with standard numerical techniques such as multi-start or DIRECT [12]. We consider two common acquisition functions of expected improvement (EI) and upper confidence bound (UCB). The EI chooses the next point with highest expected improvement over the current best outcome, i.e. the maximizer of $\alpha_{EI}(\boldsymbol{x}; \mathcal{D}) = (\mu(\boldsymbol{x}) - \tau)\Phi(z) + \sigma(\boldsymbol{x})\phi(z)$ where τ is the current best value, $z = \frac{\mu(\boldsymbol{x})-\tau}{\sigma(\boldsymbol{x})}$. The UCB [19] is given by $\alpha_{UCB}(\boldsymbol{x} \mid D) = \mu(x) + \kappa\sigma(x)$.

B. Batch Bayesian optimization

We focus on evaluating f using a batch of points. As discussed earlier, such scenarios appear, for instance, in the optimization of computer models where several machines (or cores) are available to run experiments in parallel, or in wet-lab experiments when the time of testing one experimental design is the same as testing a batch. Formally, we maximize the acquisition function as

$$\boldsymbol{X}_t = [\boldsymbol{x}_{t1}, \boldsymbol{x}_{t2}, ..., \boldsymbol{x}_{tn_t}] = \underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}}\alpha_t(\boldsymbol{x}) \tag{2}$$

where n_t is the batch size at iteration t.

C. Existing batch Bayesian optimization methods

Since the optimization in Eq. 2 is intractable, batch BO techniques avoid this computational burden by resorting to different strategies. Batch BO simulation matching [1] aims to select a batch of size q, which includes points 'close to' the best point. Constant Liar (CL) is a heuristic q-EI algorithm [9], which uses a greedy approach to iteratively construct a batch of q points. At each iteration, the heuristic uses the sequential algorithm to find a point that maximizes the expected improvement and to move to the next maximum by suppressing this point. This is done by inserting the outcome at this point as a constant value. The GP posterior is updated through the "fake" outcome which then is used to compute the acquisition function. This process is repeated till the batch is filled.

Another direction [7], [8] in batch Bayesian optimization exploits an interesting fact about GPs: the predictive variance of GPs depends only on the feature $\boldsymbol{x}$, but not the outcome values $\boldsymbol{y}$. The GP-BUCB algorithm [8] extends the sequential UCB to a batch setting by first selecting the next point, updating the predictive variance which in turn alters the acquisition function, and then selecting the next point.

More recently, Local Penalization (LP) [10] presents a heuristic approach for batch BO by iteratively penalizing the current peak in the acquisition function to find the next peak. LP depends on the estimation of Lipschitz constant to flexibly penalize the peaks. However, in general scenarios, the Lipschitz constant L is unknown. Furthermore, the use of a unique value of L assumes that the function is Lipschitz homoscedastic. Gonzalez et al [10] has demonstrated that LP outperforms a wide range of baselines in batch Bayesian optimization. We thus consider LP as the most competitive baseline in batch BO.

Although the batch algorithms can speedup the selection of experiments, there are two main drawbacks. First, most of the proposed batch BO [9], [1], [2], [7], [8], [10] involve a greedy algorithm, which chooses individual points until the batch is filled. This is often detrimental as the requirement to fill a batch enforces the selection of not only the true maxima but also noisy peaks. Second, most approaches use a predefined and fixed batch size q for all iterations. It may be inefficient if we fix the batch size for Bayesian optimization because the number of peaks in the acquisition function is unknown and constantly changing when the GP posterior is updated. In contrast, if we under-specify the number of peaks, we could miss important points that may be the optimal solution of f. Hence, it is necessary to identify the appropriate, if not exact, number of peaks for evaluation while preserving the ability of finding the optimal value of f.

III. Budgeted Batch Bayesian Optimization

We propose the novel batch Bayesian optimization method that learns the suitable batch size for each iteration. We term our approach *budgeted batch Bayesian optimization (B3O)*. The proposed method is economic in terms of number of optimal evaluations whilst preserving the performance.

A. Approximating acquisition function with IGMM

The acquisition function α is often multi-modal with the unknown number of peaks. It is intuitive to assume that the acquisition function can be approximated by a mixture of Gaussians (cf. Fig. 1) wherein the peaks in the acquisition function are equivalent to the mean locations of the underlying Gaussians. Because the number of peaks are unknown, we borrow the elegance of Bayesian nonparametrics to identify

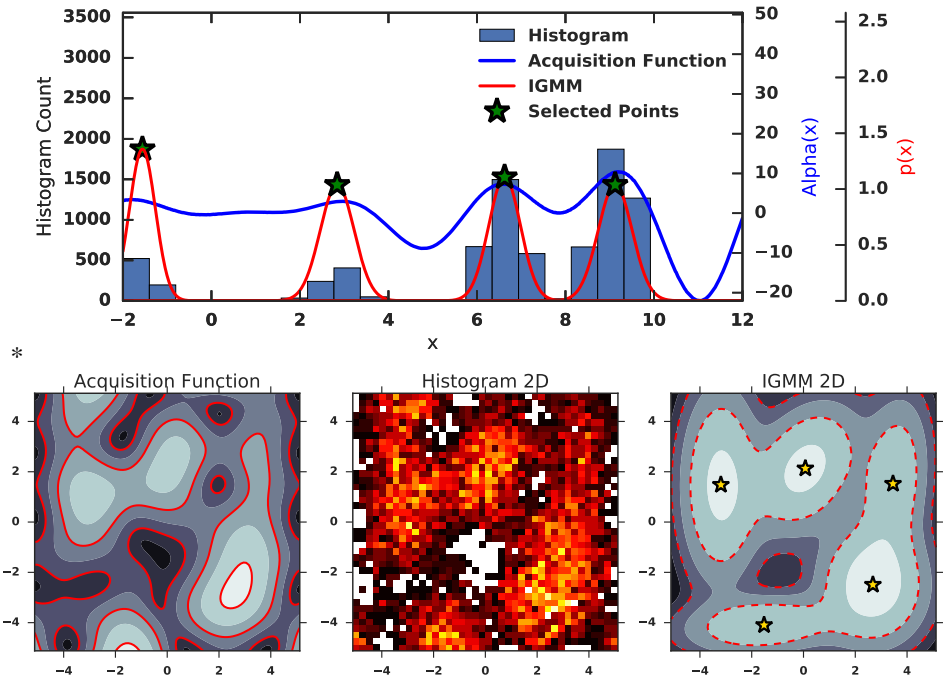

Figure 1: Budgeted Batch Bayesian Optimization. Top: i.i.d samples are drawn (see the histogram) from the acquisition function α (blue line) using batch generalized slice sampling. These generated samples after fitted with IGMM (red line) and estimated means are selected (green star). Bottom: Illustration in 2D of B3O.

the unknown number of Gaussian components. We use the infinite Gaussian mixture model (IGMM) [14] which induces the Dirichlet process prior over possibly infinite number of Gaussian components.

Our primary goal in utilizing IGMM is to approximately find the mean μ_k(s) of Gaussian distributions as the unknown peaks from the acquisition function. However, directly estimating IGMM from acquisition function is intractable. Therefore, we use the intermediate step to draw samples under the acquisition function and then fit the IGMM.

B. Batch generalized slice sampler (BGSS)

We now present the sampling technique to draw samples under the acquisition function for fitting to the IGMM. We note that sampling process, by its nature, generates more samples from high probability regions. This implies that most of the samples come from the peaks. Thus, even with small number of samples, the sampling process approximates the peaks and is then efficient. There are many existing sampling methods to draw samples from the probability distributions. However, to sample under the D-dimensional acquisition function which is not strictly probability distribution, we utilize the accept-reject sampling [6] where we keep samples in the region under the density function and ignore samples if they are outside.

In particular, we extend the standard slice sampler to the *generalized slice sampler* that draws samples from D-dimensional acquisition function α which is not a proper distribution and can be negative. To sample for s, we can sample jointly for (s, u), and then ignore u. To draw u uniformly over the region below curve of the acquisition function, we get the $\alpha_{\min} = \min \alpha(\mathcal{I}_t)$ obtained by one of the non-convex optimization toolbox (e.g., DIRECT [12]). Therefore, we can define the joint density

$$p(s,u) = \begin{cases} \frac{1}{Z} & \text{if } \alpha_{\min} < u < \alpha(s;\mathcal{I}_t) \\ 0 & \text{otherwise} \end{cases}$$

where $Z = \int \alpha(s;\mathcal{I}_t)\, ds$. The marginal density for x is then $p(s) = \int_{\alpha_{\min}}^{\alpha(s;\mathcal{I}_t)} \frac{1}{Z} du = \frac{\alpha(s;\mathcal{I}_t) - \alpha_{\min}}{Z}$.

Algorithm 1 Algorithm for Budgeted Batch BO.

Input: $\mathcal{D}_0 = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n_0}$, #iter T, acquisition function α

1: **for** $t = 1$ to T **do**
2: Fit a GP and acquisition function α from the data $\mathcal{D}_{t-1}$.
3: Draw auxiliary samples $\boldsymbol{s} \sim \alpha(x)$.
4: Fit the Infinite GMM using $\boldsymbol{s}$ to obtain the means μ_k(s)
5: Obtain $\boldsymbol{X}_t \leftarrow [\mu_1, ..., \mu_k]$ and evaluate $\boldsymbol{y}_t = f(\boldsymbol{X}_t)$
6: Augment the data $\mathcal{D}_t = \mathcal{D}_{t-1} \cup (\boldsymbol{X}_t, \boldsymbol{y}_t)$
7: **end for**

Output: $\mathcal{D}_T$

To overcome the problem in high-dimensional functions, we extend the generalized slice sampler in batch setting where a bunch of samples are drawn i.i.d at different places. Since our goal is to draw a collection of i.i.d. samples $s_{1,2,...N}$

from the acquisition function α. We define a collection of joint density distribution $p(s_1, u_1), ...p(s_M, u)$ and perform the generalized slice sampling for each joint distribution independently. In the experiment, we set M as 200. In Sec IV-D, we further study the computational efficiency of the *batch generalized slice sampling* (BGSS) for drawing samples from high dimensional acquisition functions (e.g., $D = 50$).

C. Variational inference for infinite Gaussian mixture model

IGMM is the nonparametric mixture model where the prior distribution over the mixing proportion is a Dirichlet process. In this paper, we follow [4] to derive the variational inference for IGMM [14] since the variational approach is generally faster than the Gibbs sampler. After fitting the IGMM, we obtain the mean locations $\mu_{1....K}$ as the points for evaluations in the batch BO setting. We note that K is unknown and identified automatically in this Bayesian nonparametric setting.

D. Algorithm

Although the technique of variational inference for IGMM and the slice sampling are existed for years, the idea of connecting these things at the right place for batch BO is novel. We summarize the steps for B3O in Algorithm 1. At the iteration t, we will find a batch including n_t points where the number of point n_t varies at each iteration.

IV. Experiments

We evaluate the B3O algorithm using benchmark functions and real-world experiments. We show that the B3O outperforms the baselines in the best-found-value whilst requiring fewer evaluations.

Experimental Settings: We denote the number of iterations by T, and the number of evaluations by N. The number of initial points n_0 for all methods is set as $3 \times D$ where D is the dimension. For fixed-batch approaches, the batch size at each iteration is set as $n_t = 3$ for functions with $D < 5$ or $n_t = D$ for functions with $D \geq 5$. In contrast, we do not set the batch size for B3O in advance. The performance of the algorithms is compared for a fixed number of iterations $T = 10 \times D$, and the total number of evaluated points is $N = \sum_{t=0}^{T} n_t$.

In all the experiments, the squared exponential (SE) covariance kernel given as $k(x, x') = \exp\left(-\gamma||x - x'||^2\right)$ is used in the underlying GP, where $\gamma = 0.1 \times D$. For methods using the UCB, κ is fixed to 2 (following the setting used in [10]), which allows us to compare the different batch designs using the same acquisition function. The results are taken over 20 replicates with different initial values. All implementations are in Python and simulations are done on Windows machine Core i7 Ram 24GB.

Although B3O is designed to work with any kind of acquisition function, in this paper we use B3O with UCB since we empirically notice that the UCB generally works better than EI for B3O. *All the source codes and data are available for reproducibility at the link*[1].

[1]https://github.com/ntienvu/ICDM2016_B3O

A. Comparison - best found value

We examine the performance of B3O in finding the optimum of the chosen benchmark functions. We demonstrate that our method can find better optimal values (minimum) for a fixed amount of iterations T. We report the best-found-value w.r.t. iterations in Fig. 2. Our method achieves significantly better values than the baselines in 5 over 8 functions. In other three cases, B3O still provides relatively good values. The fixed batch approaches (e.g., LP) may suffer the under-specification (the number of specified batch size n_t is smaller than the number of real peaks) at some iterations and thus negatively affect the final performance.

B. Comparison - number of evaluations

The next criteria for comparison is the total number of evaluations, $N = \sum_{t=0}^{T} n_t$. The requirement of BO is to keep the number of evaluations to find the optimal value as low as possible, as these evaluations can be costly.

It is not natural to fix the number of evaluations per batch n_t, since the number of peaks is unknown and importantly will be changed after new evaluations are added. Therefore, we may waste time and resources if we over-evaluate the number of points than we need. In particular, after all the actual peaks are detected, if we set the batch size n_t large, we will get the noisy points which are possibly close to the already detected ones due to the effect of penalizing the peaks [10]. Hence, these noisy points are not useful for evaluation. In contrast, under-evaluating the number of necessary points also brings detrimental effects of losing the optimal points.

B3O automatically identifies the suitable number of peaks from the acquisition function in each iteration. This is efficient without suffering any performance loss. Given a fixed number of iterations T, we summarize the total number of evaluations in Fig. 3 - our proposed approach significantly reduces the number of evaluations as compared to the fixed-batch baselines, especially for high dimensional settings, e.g., Alpine2 $10D$ and gSobol $10D$.

C. Analysis of computational time

Next we compare the computational time at each iteration for the different batch approaches. CL [9] and BUCB [8] take time for re-estimation of the GP when the fake observations are added and noticeably slower when the number of data points N is large. The LP [10] consumes a considerable amount of time to estimate the Lipschitz constant. In addition, LP computes and maintains the penalized cost around the visited points. The cost for penalizing and optimizing will grow for high dimensions and/or large number of observed points.

B3O is generally competitive to LP [10] in terms of computation. We run faster for low dimensional functions (e.g., less than 6 dimensions). However, LP tends to run faster than B3O for 10D functions. We note that in high dimensional functions, the bottle neck in LP is estimating the Lipschitz constant whilst for B3O the batch generalized slice sampling is the bottle neck due to the nature of the sampling algorithm.

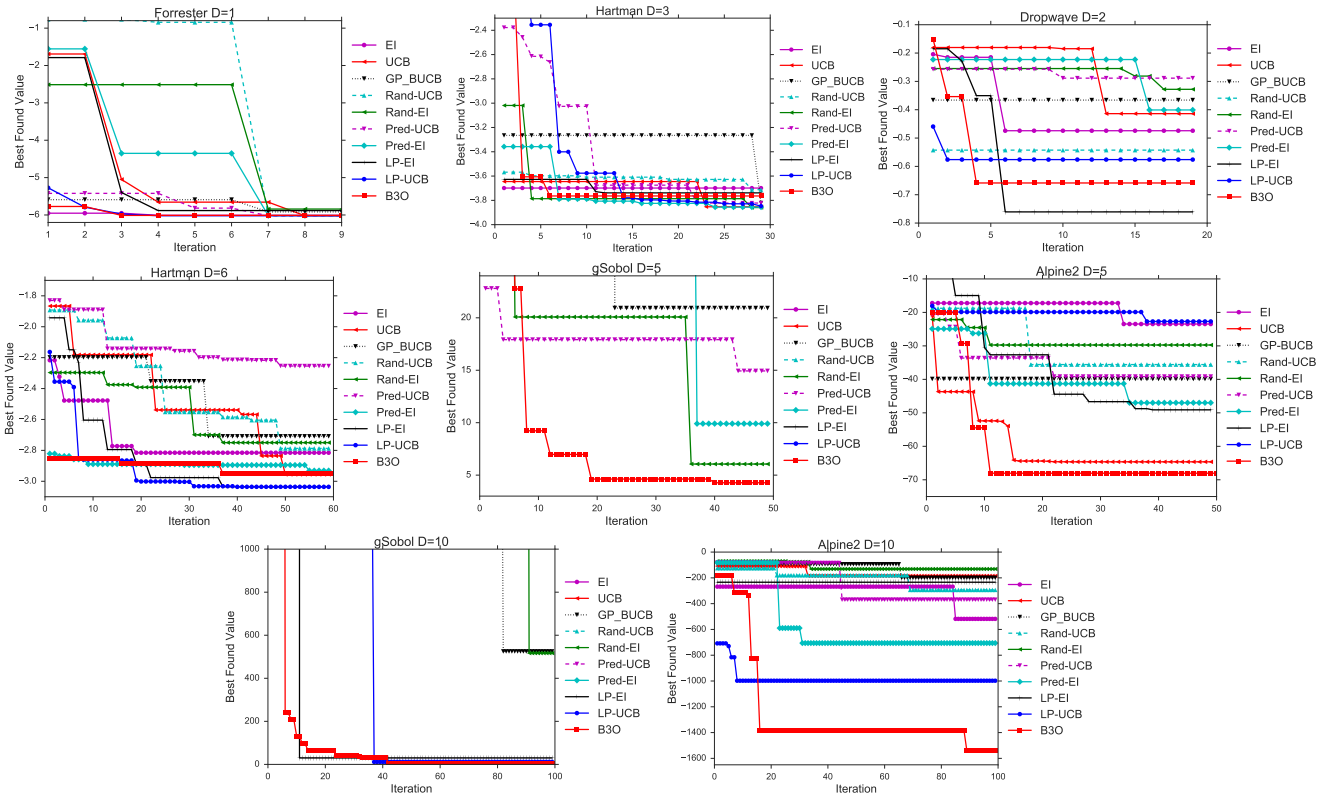

Figure 2: Best found value w.r.t. fixed number of iterations. B3O is indicated in red. EI and UCB are the sequential algorithms whilst the others are batch BO. B3O achieves the best performance for Forrester 1D, gSobol 5D,10D and Alpine2 5D, 10D.

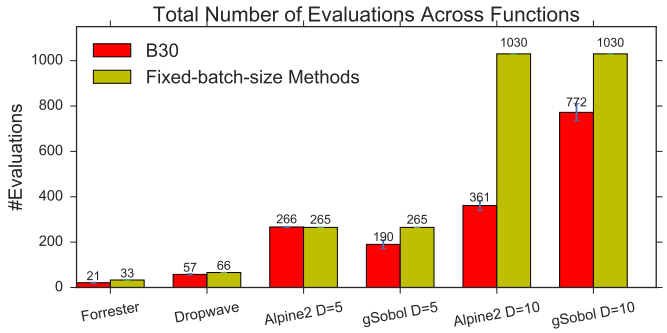

Figure 3: The ideal methods should have less total number of evaluations N for saving time and resources.

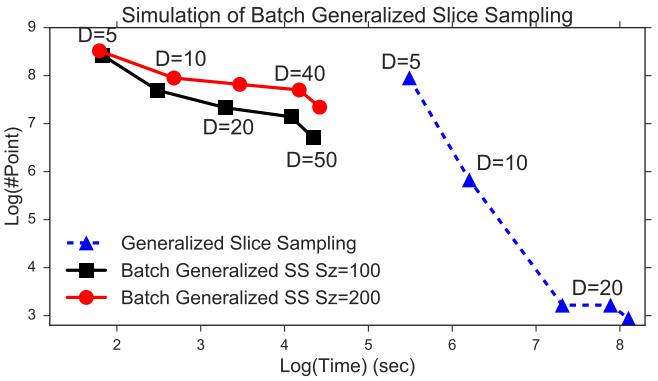

Figure 4: Simulation of BGSS (logarithmic scales used).

Dimension	1D	2D	3D	5D	6D	10D
GP-BUCB	23.9	81.1	154.7	573	1585	23,384
CL-EI	17.9	30.4	79.8	277.9	810	21,413
CL-UCB	19.9	37.3	95.5	95.5	1093	15,401
LP-EI	17.0	14.3	25.7	183.2	96.4	7,748
LP-UCB	10.3	10.9	53.2	96.0	235.8	**4,334**
B3O	**2.04**	**7.34**	**21.6**	**58.34**	**81.85**	7,726

Table I: Optimization time (second unit) for each iteration.

D. Analysis of batch generalized slice sampling (BGSS)

We investigate the efficiency of the BGSS presented in Section III-B for drawing samples under the acquisition function. We vary the observation dimension D and build the acquisition function α_{UCB}. Then, we design two BGSS settings of size 100 and 200 and compare with the generalized slice sampling. We record the computational time and the number of accepted samples in Fig. 4. BGSS significantly outperforms the standard slice sampling in terms of computation and efficiency (higher number of accepted points). These accepted points are lying under the curve and generally surrounding the peak. Therefore, these samples are beneficial to fit the IGMM since we are particularly interested in finding the peaks.

Settings	SVR, D=3	MLP, D=7	Alloy, D=4
Task	Hyper-parameter Tuning		Experimental Design
Evaluation	RMSE	Accuracy(%)	Hardness
EI	1.935(0.01)	98.29(0.00)	84.86(2.0)
UCB	1.937(0.01)	97.61(0.01)	85.48(1.8)
GP-BUCB	1.932(0.00)	98.45(0.00)	88.23(0.2)
Rand-EI	1.933(0.00)	98.41(0.00)	87.76(0.8)
Rand-UCB	1.936(0.00)	98.44(0.00)	87.87(0.9)
CL-EI	1.936(0.00)	98.44(0.00)	86.95(1.6)
CL-UCB	1.937(0.00)	98.44(0.00)	86.98(0.6)
LP-EI	1.932(0.00)	98.45(0.00)	87.71(0.7)
LP-UCB	1.936(0.00)	98.38(0.00)	86.40(1.9)
B3O	**1.928(0.00)**	**98.47(0.00)**	**88.30(0.3)**

Table II: Results for machine learning algorithms hyper-parameter tuning and alloy hardening design.

Settings	SVR, D=3	MLP, D=7	Alloy, D=4
Fixed batch approaches	39	91	42
B3O	**37**	**62**	**36**

Table III: Number of required evaluations for real applications.

E. Tuning hyperparameters for machine learning algorithms

Hyper-parameter settings greatly impact prediction accuracy of machine learning algorithms. We employ the batch Bayesian optimization to find the optimal hyper-parameters for support vector regression (SVR) and multi layer perceptron (MLP). The SVR model is evaluated using Abalone dataset, with three parameters: regularizer parameter, ϵ-insensitive loss and scale parameter for RBF kernel function. For the deep learning model, we use three layers in the model which results in 7 parameters and evaluate it on the MNIST dataset.

We compare B3O with baselines in Table II. Our model performs better than all baselines in terms of RMSE for SVR and has the highest accuracy of 98.47% for MLP. We run 10 iterations for all methods in which the number of initial points is 9 and the batch size n_t is 3 for the fixed batch BO. The number of evaluations in B3O is the lowest compared to other methods in this setting. Our model identifies the unknown number of peaks in the acquisition function, then evaluate at these points. Other fixed-batch approaches may evaluate points which are unnecessary. Thus, B3O requires less number of evaluation than the baselines (cf. Fig. III).

F. Bayesian optimization for experimental design

We consider the alloy hardening process consisting of two stages: nucleation and precipitation coarsening[16]. We aim to maximize the hardness for Aluminum-scandium alloys by designing the appropriate times and temperatures for the two stages in the process. We use B3O for the experimental design problem above and show that our method achieves the highest hardness using the smallest number of experiments (see Table II). Given the times and temperatures setting, we evaluate the hardness using the standard kinematic KWN model used by metallurgists[13] .

V. Conclusion

We have introduced a novel approach for batch Bayesian optimization. The proposed approach of B3O can identify the suitable batch size at each iteration that most existing approaches in batch BO are unable to do. We have presented the batch generalized slice sampling for drawing samples under the acquisition function. We perform extensive experiments on finding the optimal solution for 8 synthetics functions and evaluate the performance further on 3 real-world tasks. The experimental results highlight the ability of B3O in finding the optimum whilst requiring fewer evaluations than the baselines.

References

[1] J. Azimi, A. Fern, and X. Z. Fern. Batch bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems*, pages 109–117, 2010.

[2] J. Azimi, A. Jalali, and X. Zhang-fern. Hybrid batch bayesian optimization. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1215–1222, 2012.

[3] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[4] D.M. Blei and M.I. Jordan. Variational inference for dirichlet process mixtures. *Bayesian Analysis*, 1(1):121–143, 2006.

[5] 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 arXiv:1012.2599*, 2010.

[6] George Casella, Christian P Robert, and Martin T Wells. Generalized accept-reject sampling schemes. *Lecture Notes-Monograph Series*, pages 342–347, 2004.

[7] Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel gaussian process optimization with upper confidence bound and pure exploration. In *Machine Learning and Knowledge Discovery in Databases*, pages 225–240. Springer, 2013.

[8] Thomas Desautels, Andreas Krause, and Joel W Burdick. Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *The Journal of Machine Learning Research*, 15(1):3873–3923, 2014.

[9] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. A multi-points criterion for deterministic parallel global optimization based on gaussian processes. 2008.

[10] Javier González, Zhenwen Dai, Philipp Hennig, and Neil D Lawrence. Batch bayesian optimization via local penalization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 648–657, 2016.

[11] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

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

[13] R Kampmann and R Wagner. Kinetics of precipitation in metastable binary alloys-theory and applications to cu-1.9 at% ti and ni-14 at% al. In *Decomposition of Alloys: The Early Stages, Proceedings of the 2 nd Acta-Scripta Metallurgica Conference*, pages 91–103, 1983.

[14] Carl Edward Rasmussen. The infinite gaussian mixture model. In *NIPS*, volume 12, pages 554–560, 1999.

[15] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[16] JD Robson, MJ Jones, and PB Prangnell. Extension of the n-model to predict competing homogeneous and heterogeneous precipitation in al-sc alloys. *Acta Materialia*, 51(5):1453–1468, 2003.

[17] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[18] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[19] N. Srinivas, A. Krause, M. Seeger, and S. M. Kakade. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1015–1022, 2010.

[20] G Gary Wang and Songqing Shan. Review of metamodeling techniques in support of engineering design optimization. *Journal of Mechanical design*, 129(4):370–380, 2007.