7th ICML Workshop on Automated Machine Learning (2020)

Cost-aware Bayesian Optimization

Eric Hans Lee EHL59@CORNELL.EDU
Cornell University, Ithaca, USA
Valerio Perrone VPERRONE@AMAZON.COM
Cédric Archambeau CEDRICA@AMAZON.COM
Matthias Seeger MATTHIS@AMAZON.COM
Amazon, Berlin, Germany

Abstract

Bayesian optimization (BO) is a class of global optimization algorithms ubiquitous in hyperparameter optimization (HPO). BO budgets are typically given in iterations, which implicitly measures convergence in terms of the number of hyperparameter evaluations. In practice, evaluation costs may vary in different regions of the search space. For example, the cost of neural network training increases quadratically with layer size. *Cost-aware BO* measures convergence with alternative cost metrics such as time, for which vanilla BO methods are unsuited. While the standard cost-aware heuristic in the black-box setting is to normalize the acquisition function by the cost, we show that this often underperforms, and adopt a different approach by scheduling cheaper evaluations before expensive ones. We do so through two improved heuristics: a cost-effective initial design and a cost-cooled optimization phase which depreciates a learned cost model as iterations proceed.

1. Introduction

Consider minimizing a black-box function $f(\mathbf{x}) : \Omega \to \mathbb{R}$ over a set $\Omega \subset \mathbb{R}^d$ whose analytical form and gradients are unavailable, and that can only be queried through noisy evaluations. Bayesian optimization (BO) is a well-established class of methods to address this problem, and has been applied with success to hyperparameter optimization (HPO) (Snoek et al., 2012; Shahriari et al., 2016; Frazier, 2018). BO is *sample efficient*, taking fewer steps to converge than competing methods. Evaluations $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ are used to model f, typically with a Gaussian process (GP) (Rasmussen and Williams, 2006), and an acquisition function balances exploration and exploitation to determine the next evaluation. A popular choice is the Expected Improvement (EI) (Mockus et al., 1978).

BO's sample efficiency leads to fast convergence only if evaluations cost the same, an assumption that is often not true in practice. Figure 1 illustrates this by randomly evaluating 5000 hyperparameter configurations for five popular HPO problems. Unsurprisingly, resulting evaluation times vary, often by an order of magnitude or more. Moreover, the bulk of each problem's search space tends to be cheap, suggesting significant cost savings may be achieved by using a cost efficient rather than a sample efficient optimizer.

2. Background and Related Work

Most prior approaches to cost-aware BO occur in the *grey-box* setting, in which additional information about the objective is available. *Multi-fidelity* BO is one such widely studied

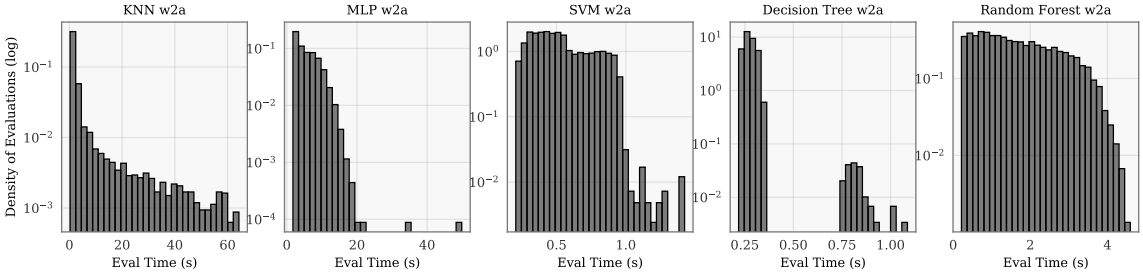

Figure 1: Runtime distribution, log-scaled, of 5000 randomly selected points for the K-nearest-neighbors (KNN), Multi-layer Perceptron (MLP), Support Vector Machine (SVM), Decision Tree (DT), and Random Forest (RF) HPO tasks. Their runtimes vary, often by an order of magnitude or more.

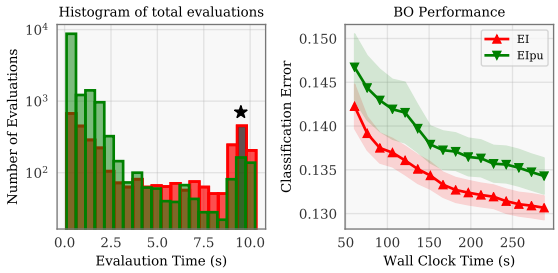

Figure 2: We compare EI and EIpu on KNN. EIpu evaluates many more cheap points than EI, which evaluates more expensive points. The optimum's cost, one of the most expensive points, is a black star. EIpu performs poorly as a result.

approach in which fidelity parameters $s \in [0,1]^m$ are assumed to be a noisy proxy for high-fidelity evaluations (Forrester et al., 2007; Kandasamy et al., 2017; Poloczek et al., 2017; Wu et al., 2019). Multi-fidelity methods are often application-specific. For example, Hyperband (Li et al., 2017; Falkner et al., 2018; Klein et al., 2016, 2017) cheaply trains many neural network configurations for only a few epochs, and then train a selected subset for further epochs. In *multi-task* BO, HPO is run on cheaper training sets before more expensive ones. While these methods demonstrate strong performance, they sacrifice generality and do not directly apply to black-box BO.

Cost-aware BO in the black-box framework is under-explored. The de-facto heuristic in this setting is to normalize the acquisition by the cost, typically predicted with a GP cost model (Snoek et al., 2012). This extends EI to *EI per unit cost (EIpu)*:

$$\text{EIpu}(\mathbf{x}) \coloneqq \frac{\text{EI}(\mathbf{x})}{c(\mathbf{x})}. \tag{1}$$

Snoek et al. (2012) showed that EIpu can boost performance on a variety of HPO problems. In our experiments, EIpu demonstrated underwhelming performance: as we will show, out of twenty HPO problems, EIpu performs worse than EI on nine. EIpu's poor performance on certain problems is explained in Figure 2, in which EIpu (green) is slower than EI (red) at HPO of a K-nearest-neighbor model. The empirical optimum, namely the best point

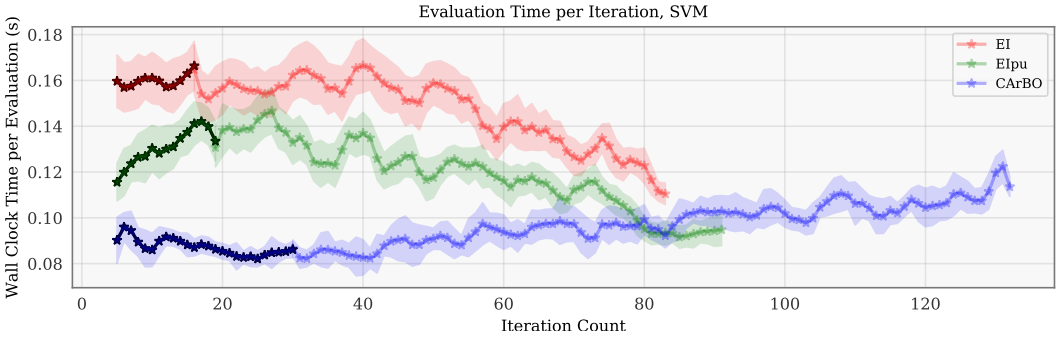

Figure 3: Median evaluation time per iteration using each method's median number of iterations for an SVM HPO problem. We shade the iterations that consume the first $\tau/8$ cost (initial design budget). CArBO clearly starts with many cheap evaluations and gradually evaluates more expensive points.

over all trials (black star), has high cost. As a result, dividing by the cost steers EIpu *away* from the optimum and diminishes its performance. This is evidenced by the evaluation time histograms: EIpu evaluates far more very cheap points compared to EI, which instead evaluates fewer but more expensive points. Due to this bias, EIpu is likely to only display strong results when optima are relatively cheap. This is a problem in the black-box setting as we do not know the global optima's cost a priori.

3. Improved Heuristics in the Cost-Aware Setting

We argue that a better cost-aware strategy can be introduced. Grey-box approaches outperform their black-box counterparts by scheduling cheap points before expensive ones. This *early and cheap, late and expensive* strategy is accomplished by leveraging additional cost information inside the optimization routine. We attempt to do the same in the black-box setting with two novel strategies. The first one is a cost-effective initial design, which aims to maximize coverage of the search space with cheap evaluations. The second one is cost-cooling, which starts the optimization with EIpu and ends it with EI by deprecating the cost model as iterations proceed. We call the combination of the two strategies cost apportioned BO (CArBO). In Figure 3, we contrast CArBO's explicit scheduling with EI and EIpu.

3.1 Cost-effective initial design

BO is typically warm-started with an *initial design*. A design is a set of points selected to learn variation in data, and BO evaluates one before optimization to provide starting data for its GP. Initial designs consume some budget, and must balance information gain with sample efficiency. They must therefore be evenly spaced throughout the domain, and are often low-discrepancy sequences (Kirk, 2012; Ryan and Morgan, 2007).

In cost-aware BO we aim to design a *cost-effective* initial design, which balances information gain with cost efficiency. A cost-effective design fills Ω with more evaluations than a traditional initial design within the same warm-start budget τ_{init}. We devote significant

Algorithm 1 Cost-effective initial design

Input: initial budget τ_{init}, optimization domain Ω.
Cumulative time $ct = 0$, initial design $\mathbf{X}_{init} = \{\}$.
Discretize Ω into $\tilde{\Omega}$.
while $ct < \tau_{init}$ **do**
  while size $\mathbf{X}_{cand} > 1$ **do**
    exclude most expensive point from $\tilde{\Omega}$.
    exclude point closest to $\mathbf{X}_{init}$ from $\tilde{\Omega}$.
  end while
  add remaining point to $\mathbf{X}_{init}$ and evaluate.
  Update ct, cost surrogate.
end while
return $\mathbf{X}_{init}$.

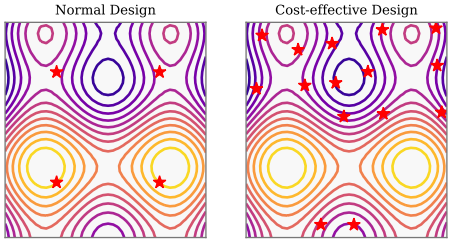

Figure 4: Two initial designs with the same cost, plotted over a contour of the synthetic cost function. *Left:* a grid of four points. *Right:* a cost-effective solution containing 15 points, which covers the search space better than the grid.

time to the initial design (1/8th of the total budget in practice), and select a cost-effective initial design through the following optimization subproblem:

$$\begin{aligned} \underset{\mathbf{X}\in 2^{\Omega}}{\arg\min} \quad & \text{fill}(\mathbf{X}) \ , \ \text{fill}(\mathbf{X}) \coloneqq \sup_{\mathbf{x}\in\Omega} \min_{\mathbf{x}_j\in\mathbf{X}} \|\mathbf{x}_j - \mathbf{x}\|_2. \\ \text{subject to} \quad & \sum_{\mathbf{x}_i\in\mathbf{X}} c(\mathbf{x}_i) < \tau_{init}. \end{aligned} \tag{2}$$

Intuitively, fill($\mathbf{X}$) is the radius of the largest empty sphere one can fit in Ω, and measures the spacing of $\mathbf{X}$ in Ω (Pronzato and Müller, 2012). The smaller a set's fill, the better distributed it is within Ω. The argmin of Eq. (2) is the initial design within τ_{init} cost with the smallest fill. Solving Eq. (2) is challenging. In the discrete setting with constant cost, it is an NP-complete vertex cover problem. Approximations described in Damblin et al. (2013); Pronzato (2017) are greedy and have a worst-case approximation factor of 2. Algorithm 1 is a variation of these and reduces to the standard greedy approach given a constant cost function. Figure 4 shows that a cost-effective design gains far more information than a standard grid, with fifteen points compared to four.

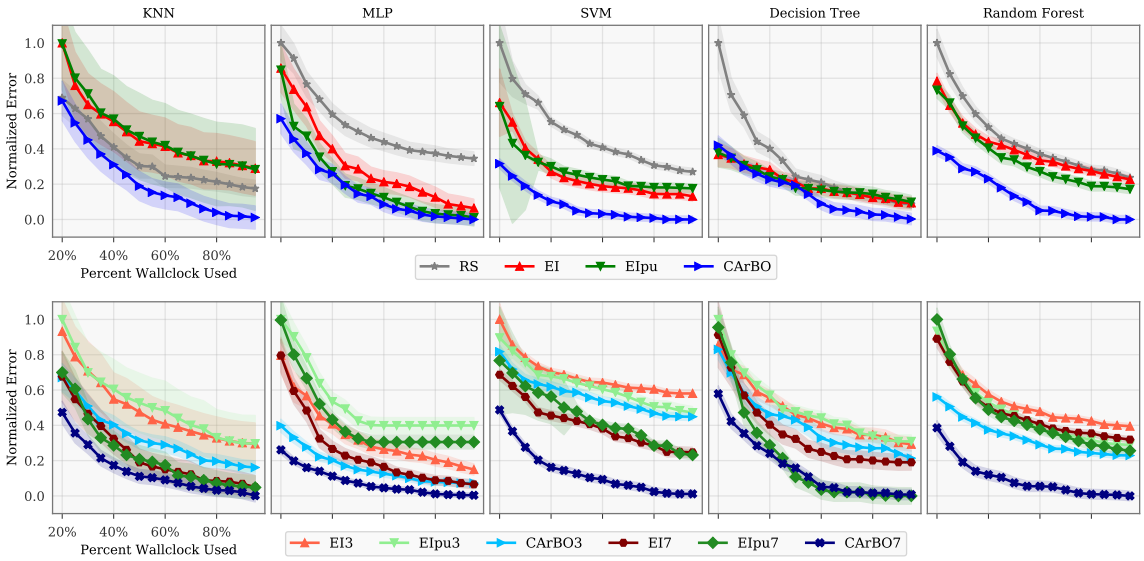

Figure 5: *Top:* Sequential comparison. *Bottom:* Batch comparison. The median is plotted, with one standard deviation shaded above and below. In almost all cases, CArBO converges significantly faster than all competing methods.

3.2 Cost-cooling

Assume that at the kth BO iteration, τ_k of the total budget τ has been used (at $k = 0$, $\tau_k = \tau_{init}$). Cost-cooling, which we call EI-cool when using EI, is defined as:

$$\text{EI-cool}(\mathbf{x}) := \frac{\text{EI}(\mathbf{x})}{c(\mathbf{x})^{\alpha}}\ ,\ \alpha = (\tau - \tau_k)/(\tau - \tau_{init}). \tag{3}$$

Cost $c(\mathbf{x})$ is assumed to be positive and modeled with a warped GP that fits the log cost $\gamma(\mathbf{x})$. The cost is predicted by $c(\mathbf{x}) = \exp(\gamma(\mathbf{x}))$ as in the standard EIpu (Snelson et al., 2004; Snoek et al., 2012). Learning $c(\mathbf{x})$ requires a warm-start, for which we use five points drawn from the search space uniformly at random.

As the parameter α decays from one to zero, EI-cool transitions from EIpu to EI. As a result, cost-cooling de-emphasizes the cost model as the optimization progresses and cheap evaluations are performed before expensive ones. The idea of cost-cooling bears connections to previous work on multi-objective, cost-preference BO (Abdolshah et al., 2019), where cost constraints are loosened to ensure that the entire Pareto frontier is explored.

4. Empirical Evaluation

We investigate cost-aware BO through the problem of tuning five popular algorithms, each trained on four different datasets, yielding twenty total benchmarks. Each HPO problem is a model in scikit-learn (Pedregosa et al., 2011). We train on four classification datasets: *splice*, *a1a*, *a3a*, and *w2a*, all available in the UCI machine learning repository (Dua and Graff, 2017). Each benchmark is replicated 50 times on independent AWS m4.xlarge machines to ensure consistent evaluation times. The search spaces and budgets are in the appendix.

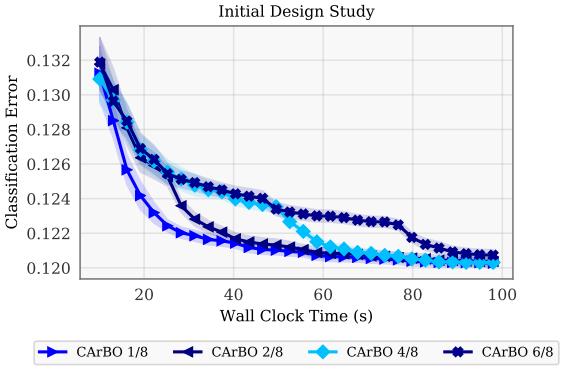

Figure 6: Impact of initial budget.

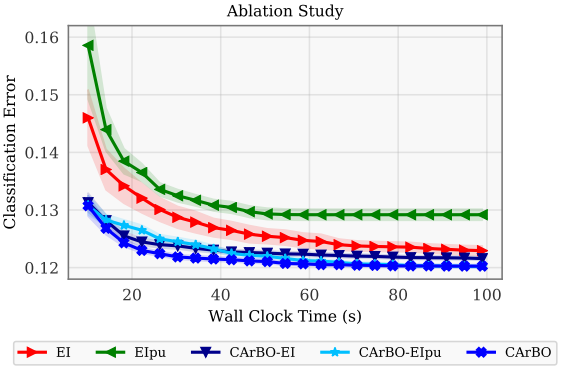

Figure 7: Initial design vs. cost-cooling.

Kernel hyperparameters for both the objective and cost GPs are calculated via maximum marginal likelihood estimation (Rasmussen and Williams, 2006).

We compare CArBO, EI, EIpu, and random search in the sequential case, as well as with batch sizes three, seven, and eleven. In the batch setting, we use the batch fantasizing technique described in Snoek et al. (2015) to get a batch from the sequential optimizer.

To condense the large number of run benchmarks, we plot performance for each HPO problem by averaging the classification error for each model over the four datasets used (Figure 5). We average as follows: first we normalize performance so that the worst optimizer starts optimization at 1.0 and the best optimizer ends at 0.0, then we take the mean over all datasets. We plot sequential results in the first row and batch results in the second. CArBO outperforms both EI and EIpu by a large margin across all batch sizes. A table of the concrete time saved is given in the appendix. Averaged across all benchmarks and batch sizes, CArBO provides a roughly 40 percent performance boost over the next-best method.

Ablation study Finally, we illustrate the CArBO's behavior relative to its internal design choices. In all experiments we fixed CArBO's cost-effective design budget to $\tau_{init} = \tau/8$, with τ being the total budget. Figure 6 varies this budget from 1/8 up to an extreme value of 6/8 of the total. CArBO's performance was relatively unchanged; using 6/8 of the total budget for the initial design degraded performance slightly. Figure 7 shows an ablation study in which we remove each component and re-run optimization, comparing CArBO to CArBO using just EI or EIpu. The initial design contributes the larger performance increase, and CArBO with cost-cooling performs best.

5. Conclusion

When budget is money or time, BO needs to be *cost* efficient, not *evaluation* efficient. We demonstrated that simple initialization and cost-cooling strategies significantly speed up cost-aware BO, outperforming EIpu and EI in both the sequential and batch setting. A number of future directions are open. Adapting CArBO's initial design and cost-cooling to other acquisition functions, such as predictive entropy search (Hernández-Lobato et al., 2014) or max-value entropy search (Wang and Jegelka, 2017), is straightforward. Combining CArBO with multi-fidelity to learn fidelity parameters and their relationship to cost is also

of interest. More principled approaches, such as the non-myopic methods of Lam et al. (2016), are another promising direction but come with significant overhead. Developing principled, cost-aware strategies with lower overhead is a key avenue for future work.

References

Majid Abdolshah, Alistair Shilton, Santu Rana, Sunil Gupta, and Svetha Venkatesh. Cost-aware multi-objective bayesian optimisation. *arXiv preprint arXiv:1909.03600*, 2019.

Guillaume Damblin, Mathieu Couplet, and Bertrand Iooss. Numerical studies of space-filling designs: optimization of latin hypercube samples and subprojection properties. *Journal of Simulation*, 7(4):276–289, 2013.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 10–15 Jul 2018.

Alexander IJ Forrester, András Sóbester, and Andy J Keane. Multi-fidelity optimization via surrogate modelling. *Proceedings of the royal society a: mathematical, physical and engineering sciences*, 463(2088):3251–3269, 2007.

Peter I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

José Miguel Hernández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Proceedings of the 28th Conference on Neural Information Processing Systems*, pages 918–926, 2014.

Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabas Poczos. Multi-fidelity bayesian optimisation with continuous approximations. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1799–1808. JMLR. org, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations*, 2015.

Roger E Kirk. Experimental design. *Handbook of Psychology, Second Edition*, 2, 2012.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*, 2016.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 528–536, 2017.

Remi Lam, Karen Willcox, and David H. Wolpert. Bayesian optimization with a finite budget: An approximate dynamic programming approach. In *Proceedings of the 30th Conference on Neural Information Processing Systems*, pages 883–891, 2016.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129):2, 1978.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Matthias Poloczek, Jialei Wang, and Peter Frazier. Multi-information source optimization. In *Advances in Neural Information Processing Systems*, pages 4288–4298, 2017.

Luc Pronzato. Minimax and maximin space-filling designs: some properties and methods for construction. *Journal de la Societe Franaise de Statistique*, 2017.

Luc Pronzato and Werner G Müller. Design of computer experiments: space filling and beyond. *Statistics and Computing*, 22(3):681–701, 2012.

Carl E. Rasmussen and Christopher K.I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Thomas P Ryan and JP Morgan. Modern experimental design. *Journal of Statistical Theory and Practice*, 1(3-4):501–506, 2007.

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.

Edward Snelson, Zoubin Ghahramani, and Carl E Rasmussen. Warped gaussian processes. In *Advances in neural information processing systems*, pages 337–344, 2004.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 26th Conference on Neural Information and Processing Systems*, pages 2951–2959, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2171–2180, 2015.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3627–3635, 2017.

Jian Wu, Saul Toscano-Palmerin, Peter I Frazier, and Andrew Gordon Wilson. Practical multi-fidelity bayesian optimization for hyperparameter tuning. 2019.

Appendix A. HPO Experiment Search Spaces

- **K-nearest-neighbors (KNN).** We consider a $5d$ search space: dimensionality reduction percentage and type in [1e-6, 1.0] log-scaled and {Gaussian, Random}, respectively, neighbor count in {1, 2, ..., 256}, weight function in {Uniform, Distance}, and distance in {Minkowski, Cityblock, Cosine, Euclidean, L1, L2, Manhattan}.
- **Multi-layer perceptron (MLP).** We consider a $11d$ search space: number of layers in {1, 2, 3, 4}, layer sizes in {10, 11, ..., 150 } log-scaled, activation in {Logistic, Tanh, ReLU}, tolerance in [1e-5, 1e-2] log-scaled, and Adam parameters (Kingma and Ba, 2015): step size in [1e-6, 1.0] log-scaled, initial step size in [1e-6, 1e-2] log-scaled, beta1 and beta2 in [1e-3, 0.99] log-scaled.
- **Support Vector Machine (SVM).** We consider a $6d$ search space: iteration count in {1, 2, ..., 128}, penalty term in {L1, L2, ElasticNet}, penalty ratio in [0, 1], step size in [1e-3, 1e3] log-scaled, initial step size in [1e-4, 1e-1] log-scaled, optimizer in {Constant, Optimal, Invscaling, Adaptive }.
- **Decision tree (DT).** We consider a $3d$ search space: tree depth in {1, 2, ..., 64}, tree split threshold in [0.1, 1.0] log-scaled, and split feature size in [1e-3, 0.5] log-scaled.
- **Random forest (RF).** We consider a $3d$ search space: number of trees in {1, 2, ..., 256}, tree depth in {1, 2, ..., 64}, and tree split threshold in [0.1, 1.0] log-scaled.

Appendix B. Performance Table

We calculate CArBO's total cost savings, defined as the time needed by CArBO to achieve comparable results to the next best optimizer (Table 1). We consider Table 1 the most instructive comparison because it provides quantitative savings instead of a qualitative ranking. We list the median cost savings for each benchmark, as well as net savings over all benchmarks, for each batch size. CArBO achieves large cost savings of roughly 40 percent, averaged over all benchmarks and batch sizes.

Objective	Budget (s)	CArBO	CArBO3	CArBO7	CArBO11
KNN a1a	150	60%	49%	-10%	-11%
KNN a3a	300	52%	58%	22%	28%
KNN splice	10	73%	75%	52%	49%
KNN w2a	400	59%	55%	60%	59%
MLP a1a	100	21%	69%	67%	69%
MLP a3a	160	-9%	50%	61%	56%
MLP splice	50	34%	62%	66%	59%
MLP w2a	200	4%	27%	20%	-7%
SVM a1a	20	22%	42%	53%	39%
SVM a3a	30	67%	66%	65%	52%
SVM splice	4	-1%	50%	67%	67%
SVM w2a	90	74%	78%	22%	72%
DT a1a	2.5	-2%	-7%	17%	-8%
DT a3a	2.5	15%	22%	-22%	35%
DT splice	2	10%	2%	-25%	2%
DT w2a	8	-18%	-41%	95%	96%
RF a1a	30	44%	28%	63%	61%
RF a3a	35	40%	54%	49%	-24%
RF splice	10	16%	33%	27%	33%
RF w2a	80	52%	48%	82%	84%
Net Saving		**32.5**%	**45.1**%	**41.6**%	**40.6**%

Table 1: For each batch size and objective, we calculate the median cost savings as a percentage of budget. Negative numbers indicate that CArBO performed worse than the best optimizer. CArBO performs strongly on the large majority of problems. Furthermore, when it does worse, it only does worse by a small amount.