Towards efficient Bayesian Optimization for Big Data

Aaron Klein[1] **Simon Bartels**[2] **Stefan Falkner**[1]
Philipp Hennig[2] **Frank Hutter**[1]

[1]Department of Computer Science
University of Freiburg, Germany
{kleinaa,sfalkner,fh}@cs.uni-freiburg.de

[2]Department of Empirical Inference
Max Planck Institute for Intelligent Systems
{simon.bartels, phennig}@tuebingen.mpg.de

Abstract

We present a new Bayesian optimization method, environmental entropy search (EnvES), suited for optimizing the hyperparameters of machine learning algorithms on large datasets. EnvES executes fast algorithm runs on subsets of the data and probabilistically extrapolates their performance to reason about performance on the entire dataset. It considers the dataset size as an additional degree of freedom to choose freely at each step of the optimization, and sets it adaptively to trade off expected information gain about the location of the best configuration vs. expected time spent. We empirically evaluate EnvES for optimizing the hyperparameters of a support vector machine, showing that extrapolating performance from small to large datasets can yield a considerable speedup over standard Bayesian optimization methods.

1 Introduction

Bayesian optimization has proven to be a successful tool for hyperparameter optimization of machine learning algorithms [1, 2, 3], as it is well suited for noisy and expensive functions $f : \mathbb{R}^d \to \mathbb{R}$ where no gradient information is available. To find the best point $\boldsymbol{x}_\star = \operatorname{argmin} f(x)$, it iterates the following steps: (1) learn a model $p(f)$ that describes the underlying function f; (2) use $p(f)$ to define an acquisition function $a : \mathbb{R}^d \to \mathbb{R}$ that quantifies how promising it is to evaluate a given point and (3) select the most promising point by optimizing this acquisition function a [4].

Despite being more sample-efficient than other global optimization methods, Bayesian optimization still typically requires tens to hundreds of function evaluations to identify strong solutions. For hyperparameter optimization, each such evaluation includes the training and validation of the machine learning algorithm at hand, which is particularly expensive on large datasets. Scaling up Bayesian optimization to big datasets is therefore a substantial challenge, the solution to which would allow automated machine learning to tackle much larger datasets.

Recently, Nickson et al. [5] considered this problem and addressed it by estimating a configuration's performance by evaluating it based on several training runs on small random subsets of fixed, manually chosen size. To transfer knowledge between similar tasks, Swersky et al. [6] used multi-task Gaussian processes (GPs). Their method is able to exploit information gained by previous optimization runs for a new task that is similar to previously seen ones. This approach allows one to optimize on a small, manually-chosen subset of datapoints first to obtain a good prior for optimizing on a larger dataset more quickly.

In this work, we present a novel Bayesian optimization method that is able to model the performance of hyperparameter settings over smoothly increasing subsets of the data. Following Williams et al. [7], we contrast the typical inputs of Bayesian optimization (here: hyperparameters) to *environmental* inputs, which can be set freely during optimization, but which are fixed at evaluation time (here: dataset size). We consider the dataset size an environmental variable that we can choose freely during each optimization step, but which is set to the entire dataset size at evaluation time (since we aim to find good hyperparameter settings for the entire dataset). Due to its prominent use of environmental variables, we dub our approach environmental entropy search (EnvES). This approach resembles the multi-task Bayesian optimization approach by Swersky et al. [6], but differs from this by considering an additional *continuous* input rather than a categorical one and can exploit available prior knowledge about the scaling of cost and performance with dataset size.

2 Method

We adapt the standard way of modeling the objective function f by adding an additional input $f : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}$ where we denote $\boldsymbol{x} \in [0,1]^d$ as our hyperparameter vector and $s \in [s_{min}, s_{max}]$ specifies the dataset size. We assume we can query the function everywhere in the input space but only make noisy observations $y_i \sim \mathcal{N}(f(\boldsymbol{x}_i, s_i), \sigma^2)$. As it is common in Bayesian optimization, we model our posterior believe $p(f|D_n)$ of f after observing some data $D_n = (\boldsymbol{x}_1, s_1, y_1), \ldots, (\boldsymbol{x}_n, s_n, y_n))$ with a GP.

Popular acquisition functions, such as expected improvement (EI) [8] or probability of improvement [9, 10], prefer configurations with better performance. In our scenario, this usually means larger dataset sizes as the quality of a machine learning model commonly increases with the amount of data used for training. Therefore, we use *entropy search* (ES) [11, 12] which aims to decrease the entropy

$$H\left[p_{min}\right] = \int p_{min}(\boldsymbol{x}) \cdot \log\left(p_{min}(\boldsymbol{x})\right) \, \mathrm{d}\boldsymbol{x} \tag{1}$$

of the posterior distribution over the global minimum $\boldsymbol{x}_\star$

$$p_{min}(\boldsymbol{x}) = p\left[\boldsymbol{x} = \underset{\boldsymbol{x}' \in \mathbb{X}}{\operatorname{argmin}}(g(\boldsymbol{x}'))\right] . \tag{2}$$

As we are only interested in finding $\boldsymbol{x}_\star$ for the full dataset size, we use $g(\boldsymbol{x}) = f(\boldsymbol{x}, s = s_{max})$ as the model to compute the entropy in this subspace only.

Observations at large dataset sizes naturally have a high estimated information gain, so in order to encourage evaluations of smaller dataset sizes we also have to take into account the time for the observation. We can achieve this by modeling a cost function $c : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^+$ that measures the time it takes to evaluate $\boldsymbol{x}$ on a dataset of size s. Instead of modeling $c(\boldsymbol{x}, s)$ directly, we follow previous work [1, 6] and use a GP to model $\log(c)$ to ensure positive runtime predictions.

Taking everything together, our acquisition function represents the change in entropy per unit time spent for the evaluation:

$$a(\boldsymbol{x}, s) = \frac{H\left[p_{min}(\boldsymbol{x}|D_n)\right] - \mathbb{E}_{p(f(\boldsymbol{x},s)|D_n,\boldsymbol{x},s)}\left[H\left[p_{min}(\boldsymbol{x}|D_n \cap \{(\boldsymbol{x}, s, f(\boldsymbol{x}, s))\})\right]\right]}{c(\boldsymbol{x}, s)} . \tag{3}$$

As an implementation note, we follow Hennig et al. [11] who discretize the continuous input space by irregularly chosen representer points sampled from EI, but after sampling, we project all representer points to the maximum dataset size. After discretization, we use Expectation Propagation in order to approximate p_{min}. However, we note that different approximations are possible [13] and our approach works orthogonally to those approximations.

To efficiently model the objective and cost functions, any prior knowledge should be incorporated into the GPs. As we expect a rather simple dependency for the dataset size compared to the dependency on hyperparameters, we use kernels of the following form for modeling performance and cost, respectively:

$$\begin{aligned} K_f\left((\boldsymbol{x}, s), (\boldsymbol{x}', s')\right) &= K_{\text{Matèrn5/2}}\left(\boldsymbol{x}, \boldsymbol{x}'\right) \cdot \left(\phi_f(s)^T \cdot \Sigma_{\phi_f} \cdot \phi_f(s')\right) \\ K_c\left((\boldsymbol{x}, s), (\boldsymbol{x}', s')\right) &= K_{\text{Matèrn5/2}}\left(\boldsymbol{x}, \boldsymbol{x}'\right) \cdot \left(\phi_c(s)^T \cdot \Sigma_{\phi_c} \cdot \phi_c(s')\right) \end{aligned} \tag{4}$$

While the Matèrn kernel can be commonly found in Bayesian optimization [1], we model the dataset size dependency with a dot product kernel using only a few basis functions ϕ_f and ϕ_c. This is motivated by the intuition that we know the computational complexity of many families of algorithms, and that the runtime usually follows the same functional form across different hyperparameter configurations. This approach yields robust predictions even in regions of the input space not well explored. Similarly, the objective function, i.e., test error or accuracy, often follows a pattern that can be exploited with the right basis functions. In the next section, we discuss the particular choices for our experiments. Additionally, we use MCMC sampling to estimate the hyperparameters, namely the noise level σ^2, the covariance amplitude θ, the d lengthscales of the Matèrn kernel, and the covariance matrices Σ_{ϕ_f} and Σ_{ϕ_c}.

To fully leverage the speedup from evaluating smaller datasets, we choose a relatively large initial design with comparably small (and thus cheap) dataset sizes in order to improve the prediction for dependencies on s. More specifically, we draw N random points in $\mathbb{X}$ and evaluate them on $s \in \left\{\frac{s_{max}}{4}, \frac{s_{max}}{8}, \frac{s_{max}}{16}, \frac{s_{max}}{32}\right\}$, which, assuming linear or superlinear scaling of the cost in s, is cheaper than evaluating $N/2$ configurations at $s = s_{max}$.

3 Experiments

To evaluate our algorithm we optimized an SVM's kernel parameter γ and regularisation parameter C. Both parameters were bounded on a log-scale between -10 and 10. On a given dataset, we set the lower bound s_{min} to be 10 times the number of classes, and s_{max} equal to the number of training data points. In each step we shuffled the data to prevent always evaluating on the same subset. We used five different benchmark datasets (between 1000 and 5000 data points) obtained from [14]. Please note that these datasets are not meant to represent Big Data, but merely serve as a computationally cheap proof of concept.

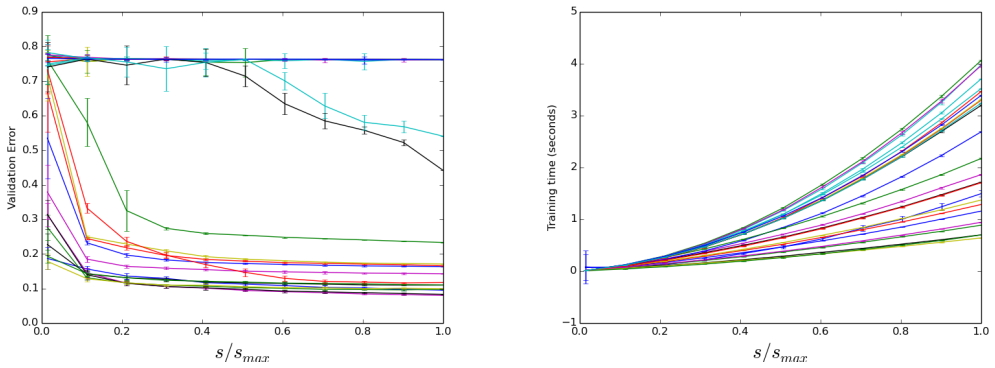

Figure 1: Training time and validation error of 25 random SVM configurations for one of the datasets. The cost for all configurations increases with s whereas the validation error decreases or stays constant. For most configurations around $\sim 20\%$ of the training data suffices to predict the performance on the whole dataset.

To gain intuition how validation error and cost vary across different s, we randomly sampled 25 configuration for several datasets and evaluated them on different dataset sizes. Figure 1 shows a representative example (plots for the other 4 datasets are qualitatively similar). Clearly, many configurations already performed well with a relatively small subset of the data, while some other configurations did not achieve good performance even with more data. Also, training time increases significantly with the number of data points.

Based on these curves we chose $\phi_f(s) = (1, (1-s)^2)^T$ as basis function to model the objective function in dimension s. With this kernel, we bias the GP to model the prediction error as a monotonically decreasing function that has its minimum at $s = s_{max}$. For the cost function we used $\phi_c(s) = (1, s)^T$ as basis function which, combined with the fact that we predict $log(c)$, captures all function of the form $c \sim s^\alpha$ with any exponent α.

To evaluate the predictive power of the used kernels, we trained models for both, validation error and cost, on the points from our initial design. Figure 2 shows their predictions as a function of s

on an unseen test point. While the actual values predicted are rather uncertain, the models already learned the general trend that the runtime grows and the validation error decreases with s. This is true even though the training data only included points with values of s up to $\frac{s_{max}}{2}$, indicated by the black vertical line in Figure 2.

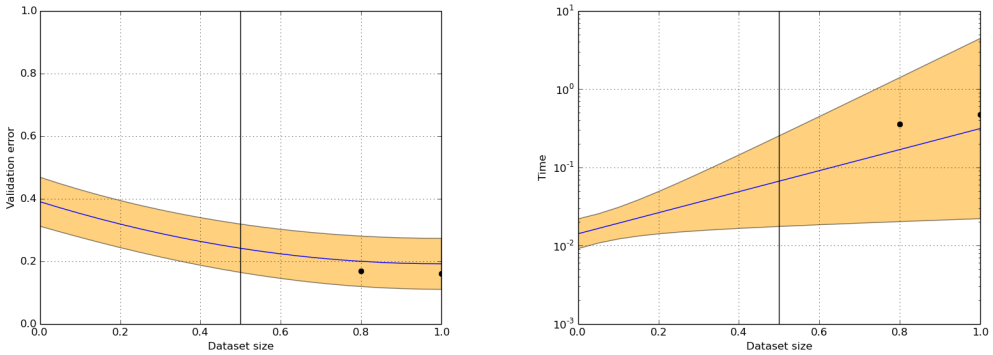

Figure 2: Extrapolation of the models for a random test point for the objective functions and the cost functions after trained on the initial design. Both models have only seen $N = 40$ points up to $\frac{s}{2}$ (indicated by the vertical line). The black dots indicate the true function values of the test point evaluated on this s.

Finally, we compared our entropy search variant to standard ES and EI by optimizing the above mentioned SVMs. To evaluate each method, in each iteration we estimated its incumbent by optimizing the posterior mean plus one standard deviation. Then, in an offline evaluation, we evaluated the true performance of that incumbent by retraining a model with it on the full training data and evaluating it on the validation dataset. For the initial design we used $N = 40$ random configurations, evaluating 10 different ones for each subset size $s \in \left\{\frac{s_{max}}{4}, \frac{s_{max}}{8}, \frac{s_{max}}{16}, \frac{s_{max}}{32}\right\}$. As Figure 3 shows, our methods achieved dramatic speedups, finding a good solution roughly 100 times faster than both EI and ES (note the logarithmic x-axis). However, compared to EI, both entropy search variants failed to converge completely to the global optimum. As entropy search is designed to learn a distribution instead of collecting low function values, it does not necessarily evaluate many points around the expected minimum. Due to noise inside the model, this often leads to a limited accuracy for the exact value of $\boldsymbol{x}_\star$. This is a known shortcoming of ES when the model performs poorly, and improving this is subject of future work.

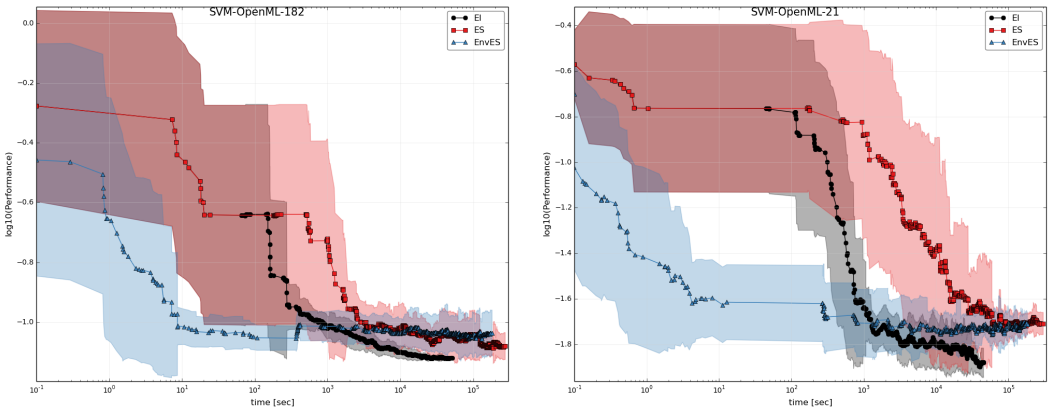

Figure 3: Comparison of our environmental entropy search method (EnvES) against expected improvement (EI) and standard entropy search (ES) on the SVM task on two different datasets. Note the logarithmic x-axis.

4 Conclusions

We presented a new Bayesian optimization method that models the quality of hyperparameter configurations of machine learning algorithms as a function of the dataset size they run on. In this way our method achieves good (albeit not perfect) performance roughly 100 times faster than expected improvement or standard entropy search, even on the rather small datasets presented here. In future work, we will apply our methods to much larger datasets using scalable learning methods, such as deep neural networks. Furthermore, we plan to also consider other potential environmental inputs that could be adjusted during optimization, such as the number of positive or negative datapoints for imbalanced data, the number of classes in image classification, or the number of epochs used for stochastic gradient descent training.

References

[1] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NIPS'12)*, pages 2960–2968, 2012.

[2] F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. Coello, editor, *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, Lecture Notes in Computer Science, pages 507–523. Springer-Verlag, 2011.

[3] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NIPS'11)*, pages 2546–2554, 2011.

[4] E. Brochu, V. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *Computing Research Repository (CoRR)*, abs/1012.2599, 2010.

[5] T. Nickson, M. A Osborne, S. Reece, and S. Roberts. Automated machine learning on big data using stochastic algorithm tuning. *Computing Research Repository (CoRR)*, abs/1407.7969, 2014.

[6] K. Swersky, J. Snoek, and R. Adams. Multi-task bayesian optimization. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Proceedings of the 27th International Conference on Advances in Neural Information Processing Systems (NIPS'13)*, 2013.

[7] B. Williams, T. Santner, and W. Notz. Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica*, 2000.

[8] J. Mockus, V. Tiesis, and A. Zilinskas. The application of bayesian methods for seeking the extremum. *Towards Global Optimization*, 2, 1978.

[9] D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[10] H. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Fluids Engineering*, 86(1):97–106, 1964.

[11] Philipp Hennig and Christian J. Schuler. Entropy search for information-efficient global optimization. *jmlr*, 98888(1):1809–1837, 2012.

[12] J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. elsevier science direct. 2009.

[13] J. Hernández-Lobato, M. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In Z. Ghahramani, M. Welling, C. Cortes, N.d. Lawrence, and K.q. Weinberger, editors, *Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NIPS'14)*, 2014.

[14] J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.