Electronic Journal of Statistics
Vol. 11 (2017) 4945–4968
ISSN: 1935-7524
https://doi.org/10.1214/17-EJS1335SI

Fast Bayesian hyperparameter optimization on large datasets*

Aaron Klein and Stefan Falkner

Department of Computer Science
University of Freiburg
e-mail: kleinaa@cs.uni-freiburg.de
e-mail: sfalkner@cs.uni-freiburg.de

Simon Bartels and Philipp Hennig

Max Planck Institute for Intelligent Systems
Spemannstr. 34, Tübingen, Germany
e-mail: simon.bartels@tuebingen.mpq.de
e-mail: phennig@tuebingen.mpg.de

and

Frank Hutter

Department of Computer Science
University of Freiburg
e-mail: fh@cs.uni-freiburg.de

Abstract: Bayesian optimization has become a successful tool for optimizing the hyperparameters of machine learning algorithms, such as support vector machines or deep neural networks. Despite its success, for large datasets, training and validating a single configuration often takes hours, days, or even weeks, which limits the achievable performance. To accelerate hyperparameter optimization, we propose a generative model for the validation error as a function of training set size, which is learned during the optimization process and allows exploration of preliminary configurations on small subsets, by extrapolating to the full dataset. We construct a Bayesian optimization procedure, dubbed FABOLAS, which models loss and training time as a function of dataset size and automatically trades off high information gain about the global optimum against computational cost. Experiments optimizing support vector machines and deep neural networks show that FABOLAS often finds high-quality solutions 10 to 100 times faster than other state-of-the-art Bayesian optimization methods or the recently proposed bandit strategy Hyperband.

Received June 2017.

1. Introduction

The performance of many machine learning algorithms hinges on certain hyperparameters. For example, the prediction error of non-linear support vector

*This paper is an extended version of our AISTATS 2017 conference paper Klein et al. (2017a)

machines depends on regularization and kernel hyperparameters C and γ; and modern neural networks are sensitive to a wide range of hyperparameters, including learning rates, momentum terms, number of units per layer, dropout rates, weight decay, etc. (Montavon et al., 2012). The poor scaling of naïve methods like grid search with dimensionality has driven interest in more sophisticated hyperparameter optimization methods over the past years (Bergstra et al., 2011; Hutter et al., 2011; Bergstra and Bengio, 2012; Snoek et al., 2012; Bardenet et al., 2014; Bergstra et al., 2014; Swersky et al., 2013, 2014; Snoek et al., 2014, 2015). *Bayesian optimization* has emerged as an efficient framework, achieving impressive successes. For example, in several studies, it found better instantiations of convolutional network hyperparameters than domain experts, repeatedly improving the top score on the CIFAR-10 (Krizhevsky, 2009) benchmark without data augmentation (Snoek et al., 2012; Domhan et al., 2015; Snoek et al., 2015).

In the traditional setting of Bayesian hyperparameter optimization, the loss of a machine learning algorithm with hyperparameters $\boldsymbol{x} \in \mathbb{X}$ is treated as the "black-box" problem of finding $\arg\min_{\boldsymbol{x}\in\mathbb{X}} f(\boldsymbol{x})$, where the only mode of interaction with the objective f is to evaluate it for inputs $\boldsymbol{x} \in \mathbb{X}$. If individual evaluations of f on the entire dataset require days or weeks, only very few evaluations are possible, limiting the quality of the best found value. Human experts instead often study performance on subsets of the data first, to become familiar with its characteristics before gradually increasing the subset size (Bottou, 2012; Montavon et al., 2012). This approach can still outperform contemporary Bayesian optimization methods.

Motivated by the experts' strategy, here we leverage dataset size as an additional degree of freedom enriching the representation of the optimization problem. We treat the size of a randomly subsampled dataset N_{sub} as an additional input to the blackbox function, and allow the optimizer to actively choose it at each function evaluation. This allows Bayesian optimization to mimic and improve upon human experts when exploring the hyperparameter space. In the end, N_{sub} is not a hyperparameter itself, but the goal remains a good performance on the full dataset, i.e. $N_{sub} = N$.

While in this paper we focus on hyperparameter optimization for large datasets, in principle, our method could also be applied to other scenarios where cheap but potentially biased and noisy approximations of the actual objective function are available, such as, for instance, in the work by Kandasamy et al. (2016), which introduces a Bayesian optimization variant that can optimize expensive functions by exploiting cheaper fidelities. Our method's only assumption is that one can define a proper basis function to describe the similarity between the objective function and its approximations. Another interesting application would be to likelihood-free inference, where Bayesian optimization has been successfully applied before (Gutmann and Corander, 2016).

Hyperparameter optimization for large datasets has been explored by other authors before. Our approach is similar to Multi-Task Bayesian optimization by Swersky et al. (2013), where knowledge is transferred between a finite number of correlated tasks. If these tasks represent manually-chosen subset-sizes, this

method also tries to find the best configuration for the full dataset by evaluating smaller, cheaper subsets. However, the discrete nature of tasks in that approach requires evaluations on the entire dataset to learn the necessary correlations. Instead, our approach exploits the regularity of performance across dataset size, enabling generalization to the full dataset without evaluating it directly.

Other approaches for hyperparameter optimization on large datasets include work by Nickson et al. (2014), who estimated a configuration's performance on a large dataset by evaluating several training runs on small, random subsets of fixed, manually-chosen sizes. Krueger et al. (2015) showed that, in practical applications, small subsets can suffice to estimate a configuration's quality, and proposed a cross-validation scheme that sequentially tests a fixed set of configurations on a growing subset of the data, discarding poorly-performing configurations early.

Li et al. (2017) proposed a multi-arm bandit strategy, called Hyperband, which dynamically allocates more and more resources to randomly sampled configurations based on their performance on subsets of the data. Hyperband assures that only well-performing configurations are trained on the full dataset while discarding bad ones early. Despite its simplicity, in their experiments the method was able to outperform well-established Bayesian optimization algorithms.

The remainder of the paper is structured as follow: In §2, we review Bayesian optimization, in particular the Entropy Search algorithm (Hennig and Schuler, 2012) on which our method is based. In §3 we show that subsets of the training data are often sufficient to reason about the performance of a hyperparameter configuration. In §4 we then present previous approaches such as Multi-task Bayesian optimization and Hyperband. In §5, we introduce our new Bayesian optimization method FABOLAS for hyperparameter optimization on large datasets. In each iteration, FABOLAS chooses the configuration x and dataset size N_{sub} predicted to yield most information about the loss-minimizing configuration on the full dataset per *unit time spent*. Finally, in §6, a broad range of experiments with support vector machines and various deep neural networks show that FABOLAS often identifies good hyperparameter settings 10 to 100 times faster than state-of-the-art Bayesian optimization methods acting on the full dataset, as well as Hyperband.

2. Bayesian optimization

Given a black-box function $f : \mathbb{X} \to \mathbb{R}$, Bayesian optimization[1] aims to find an input $\boldsymbol{x}_\star \in \arg\min_{\boldsymbol{x}\in\mathbb{X}} f(\boldsymbol{x})$ that globally minimizes f. It requires a prior $p(f)$ over the objective function f and an acquisition function $a_{p(f)} : \mathbb{X} \to \mathbb{R}$ quantifying the *utility* of an evaluation at any $\boldsymbol{x}$. Popular choices for the objective model are Gaussian processes (Snoek et al., 2012) (see Section 2.1), random forests (Hutter et al., 2011) or Bayesian neural networks (Snoek et al., 2015; Springenberg et al., 2016).

[1] Comprehensive tutorials are presented by Brochu et al. (2010) and Shahriari et al. (2016).

With these ingredients, the following three steps are iterated (Brochu et al., 2010): (1) find the most promising $\boldsymbol{x}_{n+1} \in \arg\max a_p(\boldsymbol{x})$ by numerical optimization; (2) evaluate the expensive and often noisy function $y_{n+1} \sim f(\boldsymbol{x}_{n+1}) + \mathcal{N}(0, \sigma^2)$ and add the resulting data point $(\boldsymbol{x}_{n+1}, y_{n+1})$ to the set of observations $\mathcal{D}_n = (\boldsymbol{x}_j, y_j)_{j=1\ldots n}$; and (3) update $p(f \mid \mathcal{D}_{n+1})$ and $a_{p(f|\mathcal{D}_{n+1})}$. Algorithm 1 shows pseudo code for Bayesian optimization. Typically, evaluations of the acquisition function a are cheap compared to evaluations of f such that the optimization effort is negligible.

Algorithm 1 Bayesian Optimization

1: Initialize data $\mathcal{D}_0$ using an initial design.
2: **for** $t = 1, 2, \ldots$ **do**
3: Fit probabilistic model for $f(\boldsymbol{x})$ on data $\mathcal{D}_{t-1}$
4: Choose $\boldsymbol{x}_t$ by maximizing the acquisition function $a_p(\boldsymbol{x})$
5: Evaluate $y_t \sim f(\boldsymbol{x}_t) + \mathcal{N}(0, \sigma^2)$, and augment the data: $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(\boldsymbol{x}_t, y_t)\}$
6: Choose incumbent $\hat{\boldsymbol{x}}_t \leftarrow \arg\min\{y_1, \ldots y_t\}$
7: **end for**

2.1. Gaussian processes

Gaussian processes (GP) are a prominent choice for $p(f)$, thanks to their descriptive power and analytic tractability (e.g. Rasmussen and Williams, 2006). Formally, a GP is a collection of random variables, such that every finite subset of them follows a multivariate normal distribution. A GP is identified by a mean function m (often set to $m(\boldsymbol{x}) = 0\ \forall \boldsymbol{x} \in \mathbb{X}$), and a positive definite covariance function (kernel) $k(\boldsymbol{x}, \boldsymbol{x}')$. Given observations $\mathcal{D}_n = (\boldsymbol{x}_j, y_j)_{j=1\ldots n} = (\boldsymbol{X}, \boldsymbol{y})$ with joint Gaussian likelihood $p(\boldsymbol{y} \mid \boldsymbol{X}, f(\boldsymbol{X}))$, the posterior $p(f|\mathcal{D}_n)$ follows another GP, with mean and covariance functions of tractable, analytic form.

The covariance function determines how observations influence the prediction. For the hyperparameters we wish to optimize, we adopt the Matérn $5/2$ kernel (Matérn, 1960), in its Automatic Relevance Determination form (MacKay and Neal, 1994). This stationary, twice-differentiable model constitutes a relatively standard choice in the Bayesian optimization literature. In contrast to the Gaussian kernel popular elsewhere, it makes less restrictive smoothness assumptions, which can be helpful in the optimization setting (Snoek et al., 2012):

$$k_{5/2}(\boldsymbol{x}, \boldsymbol{x}') = \theta \left(1 + \sqrt{5} d_{\boldsymbol{\lambda}}(\boldsymbol{x}, \boldsymbol{x}') + 5/3 d^2_{\boldsymbol{\lambda}}(\boldsymbol{x}, \boldsymbol{x}')\right) e^{-\sqrt{5} d_{\boldsymbol{\lambda}}(\boldsymbol{x}, \boldsymbol{x}')}. \tag{1}$$

Here, θ and $\boldsymbol{\lambda}$ are free parameters—hyperparameters of the GP surrogate model—and $d_{\boldsymbol{\lambda}}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{x}')^\intercal \operatorname{diag}(\boldsymbol{\lambda})(\boldsymbol{x} - \boldsymbol{x}')$ is the Mahalanobis distance. An additional hyperparameter of the GP model is a overall noise covariance needed to handle noisy observations. For clarity: These GP hyperparameters are *internal* hyperparameters of the Bayesian optimizer, as opposed to those of the target machine learning algorithm to be tuned. Section 5.4 shows how we handle them.

2.2. Acquisition functions

The role of the acquisition function is to trade off exploration vs. exploitation. Popular choices include Expected Improvement (EI) (Mockus et al., 1978), Upper Confidence Bound (UCB) (Srinivas et al., 2010), Entropy Search (ES) (Hennig and Schuler, 2012), and Predictive Entropy Search (PES) (Hernández-Lobato et al., 2014). In our experiments, we will use EI and ES.

We found EI to perform robustly in most applications, providing a solid baseline; it is defined as

$$a_{\mathrm{EI}}(\boldsymbol{x}|\mathcal{D}_n) = \mathbb{E}_p[\max(f_{\min} - f(\boldsymbol{x}), 0)]\,. \tag{2}$$

where $f_{\min}$ is the best function value known (also called the *incumbent*). This expected drop over the best known value is high for points predicted to have small mean and/or large variance. Its performance is, in our experience, comparable to UCB which is why we do not include it in our later experiments.

Both ES and PES, estimate the information about the location of the minimum as the measure of utility. This quantity takes global information into account contrasting the local nature of EI and UCB. The difference between ES and PES stems from different approximations made to compute the acquisition function, but no conceptual distinction. Why we decided to use ES over PES is discussed in Section 5.4. Due to the complexity of the algorithm and to provide the necessary detail to extend ES to our method, the following section contains a detailed introduction.

2.3. Entropy search

Entropy Search is a more recent acquisition function that selects evaluation points based on the predicted *information gain* about the optimum, rather than aiming to evaluate near the optimum. At the heart of ES lies the probability distribution $p_{\min}(\boldsymbol{x} \mid \mathcal{D}) := p(\boldsymbol{x} \in \arg\min_{\boldsymbol{x}' \in \mathbb{X}} f(\boldsymbol{x}') \mid \mathcal{D})$, the belief about the function's minimum given the prior on f and observations $\mathcal{D}$. Given $p(f)$, the probability that a point is the minimum is defined with suggestive notation as

$$\begin{aligned} p_{min}(\boldsymbol{x}|\mathcal{D}) &= p(\boldsymbol{x} \in \arg\min_{\boldsymbol{x}' \in \mathcal{X}} f(\boldsymbol{x}')|D) \\ &= \int p(f|\mathcal{D}) \prod_{\substack{\tilde{\boldsymbol{x}} \in [a,b] \\ \tilde{\boldsymbol{x}} \neq \boldsymbol{x}}} \Theta[f(\tilde{\boldsymbol{x}}) - f(\boldsymbol{x})]\, df \end{aligned} \tag{3}$$

where Θ is the Heaviside step function. The product in this equation is over an infinite domain (yet well-defined if $p(f|D)$ is sufficiently regular). In practice, it has to be represented in a finite form. We follow the approach of Hennig and Schuler (2012), who approximate $p(f|D)$ by a finite-dimensional Gaussian over an irregular grid of points $\boldsymbol{r}_1, ..., \boldsymbol{r}_Z$, which are designed heuristically to provide good interpolation resolution on $p_{\min}$. Like Hennig and Schuler (2012), we sample these so called representer points using Expected Improvement. This step

reduces $p_{\min}$ to a discrete distribution, and turns the infinite product in Equation 3 into a finite one. That distribution itself is still analytically intractable, but an analytically tractable (in particular, differentiable) approximation $q_{\min}(\boldsymbol{r}_j)$ of good empirical quality can be computed using *Expectation Propagation (EP)* (Minka, 2001), of computational cost $\mathcal{O}(Z^4)$. EP does not only yield $p_{\min}$, but also the gradient with respect to means and covariances of the model at the representer points allowing efficient computations after an expensive initial calculation of these quantities. This particular application of EP (dubbed EPMGP) to Gaussian integrals was introduced by Cunningham et al. (2012) where all the details can be found.

The *information gain* at $\boldsymbol{x}$ is then measured by the expected Kullback-Leibler divergence (relative entropy) between $p_{\min}(\cdot \mid \mathcal{D} \cup \{(\boldsymbol{x}, y)\})$ and the uniform distribution $u(\boldsymbol{x})$, with expectations taken over the measurement y to be obtained at $\boldsymbol{x}$:

$$a_{\mathrm{ES}}(\boldsymbol{x}) := \mathbb{E}_{p(y|\boldsymbol{x},\mathcal{D})}\left[\int p_{\min}(\boldsymbol{x}' \mid \mathcal{D}') \cdot \log \frac{p_{\min}(\boldsymbol{x}' \mid \mathcal{D}')}{u(\boldsymbol{x}')} \, \mathrm{d}\boldsymbol{x}'\right], \tag{4}$$

where $\mathcal{D}' = \mathcal{D} \cup \{(\boldsymbol{x}, y)\}$. The primary numerical challenge in this framework is the computation of $p_{\min}(\cdot \mid \mathcal{D}')$ and the integral above. Due to the intractability, several approximations have to be made.

Algorithm 2 provides pseudocode for our implementation of Entropy Search. Lines 1–12 precompute various quantities that are needed for evaluating the acquisition function, which is optimized in line 13. Specifically, after sampling K hyperparameter settings from the marginal loglikelihood for the GP using MCMC (line 1), for every hyperparameter setting $\boldsymbol{\theta}_i$, the algorithm

- fits a GP (line 4),
- samples representer points with respect to a_{EI} (line 5),

Algorithm 2 Selection of next point by Entropy Search

Require: $\mathcal{D}_n = (\boldsymbol{x}_j, y_j)_{j=1\ldots n}$
1: Sample K instantiations of the GP hyperparameters $\boldsymbol{\Theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K]$ w.r.t. marginal likelihood
2: $p_{min} \leftarrow [], \Omega \leftarrow [], \boldsymbol{R} \leftarrow [], \boldsymbol{U} \leftarrow []$
3: **for** $i = 1 \ldots K$ **do**
4: Fit GP model $\mathcal{M}^{(i)}$ on $\mathcal{D}_n$ with hyperparameter $\boldsymbol{\theta}_i$
5: $(\boldsymbol{r}_1, a_{EI}(\boldsymbol{r}_1)) \ldots, (\boldsymbol{r}_Z, a_{EI}(\boldsymbol{r}_Z)) \sim a_{EI}(\boldsymbol{x}|\mathcal{M}^{(i)})$ ▷ Sample Z representer points
6: $\boldsymbol{R}[i] \leftarrow \boldsymbol{r}_1, \ldots, \boldsymbol{r}_Z$ ▷ Store representer points $R \in \mathbb{R}^{K \times Z \times D}$
7: $\boldsymbol{U}[i] \leftarrow a_{EI}(\boldsymbol{r}_1), \ldots, a_{EI}(\boldsymbol{r}_Z)$ ▷ Store LogEI values of the representer points $U \in \mathbb{R}^{K \times Z}$
8: Let $\boldsymbol{\mu}, \Sigma$ be the mean and covariance matrix at $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_Z$ based on $\mathcal{M}^{(i)}$
9: $p_{min}[i] \leftarrow$ computePmin $(\boldsymbol{\mu}, \Sigma)$ ▷ Probability of each $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_Z$ to be the minimum.
10: For $p = 1, \ldots, P$: $\boldsymbol{\omega}_p \sim \mathcal{N}(0, \boldsymbol{I}_Z)$ ▷ Stochastic change to hallucinate P values at representer points
11: $\Omega[i] \leftarrow [\boldsymbol{\omega_1}, \ldots, \boldsymbol{\omega}_P]$ ▷ Store stochastic change for the innovations $\Omega \in \mathbb{R}^{K \times Z \times P}$
12: **end for**
13: $\boldsymbol{x}_{n+1} \leftarrow \arg\max_{\boldsymbol{x} \in \mathcal{X}}$ InformationGain$(\boldsymbol{x}, \mathcal{D}_n, \boldsymbol{R}, \boldsymbol{U}, \Omega, \boldsymbol{\Theta})$
14: **return** $\boldsymbol{x}_{n+1}$

- stores the representer points and their logarithmic EI values (lines 6 and 7),
- computes $\boldsymbol{\mu}$ and Σ for the joint predictive distribution at the representer points (line 8),
- computes p_{min} given $\boldsymbol{\mu}$ and Σ, using EPMGP (line 9),
- draws random points from a normal distribution centered at 0 and unit variance (line 10) for the innovation in Algorithm 4, and stores them (line 11) for later usage.

Given these quantities, Algorithm 3 then computes the ES acquisition function from Equation 4.

For each hyperparameter $\boldsymbol{\theta}_i$ of the GP, it then carries out the following steps:

- train a model $\mathcal{M}^{(i)}$ on the data $\mathcal{D}$ by computing the Cholesky decomposition (line 3)
- based on this model $\mathcal{M}^{(i)}$, compute the mean and the variance for the test point $\boldsymbol{x}$ and the mean and covariance for the representer points $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_Z$ (line 4 and 5)
- For each of the P stochastic change vectors $\boldsymbol{\omega}_p$ sampled in Algorithm 1 (line 10) and stored in $\Omega[i, j, :]$,
  - fantasize the change $\Delta\boldsymbol{\mu}, \Delta\Sigma$ of the current posterior $p(f|D)$ (line 7) with Algorithm 4
  - estimate the p_{min} distribution of this updated posterior (line 8)
  - compute the relative change in entropy (line 9)
- take the expectation over $p(y|\boldsymbol{x}, D)$ of Equation (3) (line 10)
- marginalize the acquisition function $a_{\mathrm{ES}}(\boldsymbol{x})$ over all hyperparameters $\boldsymbol{\Theta}$ (line 13)

Algorithm 3 InformationGain

Require: $\boldsymbol{x}, \mathcal{D}, \boldsymbol{R}, \boldsymbol{U}, \Omega, \boldsymbol{\Theta}$

1: $a(\boldsymbol{x}) \leftarrow 0$
2: **for** $i = 1, \ldots K$ **do** ▷ Marginalization over $\boldsymbol{\Theta}$
3: Let $\mathcal{M}^{(i)}$ be the trained model on $\mathcal{D}$ with hyperparameters $\boldsymbol{\theta}_i$
4: Let μ, σ^2 be the predictive mean and variance at $\boldsymbol{x}$ based on $\mathcal{M}^{(i)}$
5: Let $\boldsymbol{\mu}, \Sigma$ be the mean and covariance matrix at $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_Z$ based on $\mathcal{M}^{(i)}$
6: **for** $j = 0, \ldots P$ **do** ▷ Averages over all hallucinated values.
7: $\Delta\boldsymbol{\mu}, \Delta\Sigma \leftarrow$ Innovations$(\boldsymbol{x}, \mathcal{M}^{(i)}, \boldsymbol{R}[i, :, :], \sigma^2, \Omega[i, j, :])$ ▷ Change in the posterior believe at $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_Z$ if we would evaluate at $\boldsymbol{x}$
8: $q_{min} \leftarrow$ computePmin $(\boldsymbol{\mu} + \Delta\boldsymbol{\mu}, \Sigma + \Delta\Sigma)$ ▷ New Pmin of the updated posterior
9: $d\,\mathrm{H} \leftarrow -\sum_j q_{min}\big(\log(q_{min}) + \boldsymbol{U}[i]\big) + \sum_j p_{min}[i]\big(\log(p_{min}[i]) + \boldsymbol{U}[i]\big)$
10: $a(\boldsymbol{x}) \leftarrow a(\boldsymbol{x}) + \frac{1}{P} dH$
11: **end for**
12: **end for**
13: **return** $\frac{1}{K} a(\boldsymbol{x})$

This algorithm in turns makes use of Algorithm 4 to compute the innovations, which

- computes the change in the mean $\Delta\boldsymbol{\mu}$ by first computing the correlation $\Sigma(\boldsymbol{x}, \boldsymbol{r})$ of $\boldsymbol{x}$ and the representer points $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_Z$ and multiplying it with the Cholesky decomposition of the $k(\boldsymbol{x}, \boldsymbol{x})$ and the vector $\omega \in \boldsymbol{\Omega}$. Note that this change is stochastic (line 1).
- computes the change of the covariance (line 2) which is deterministic

Algorithm 4 Innovations

Require: $\boldsymbol{x}, \mathcal{M}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_Z, \sigma^2, \boldsymbol{\omega}$
1: $\Delta\boldsymbol{\mu}(\boldsymbol{x}) = \Sigma(\boldsymbol{x}, \boldsymbol{r}) \cdot \sigma^2 \cdot C[\sigma^2 + \sigma^2_{noise}]\boldsymbol{\omega}$ ▷ $\Sigma(\boldsymbol{x}, \boldsymbol{x}')$ denotes the correlation between $\boldsymbol{x}$ and $\boldsymbol{x}'$ based on $\mathcal{M}$
2: $\Delta\Sigma(\boldsymbol{x}) = \Sigma(\boldsymbol{x}, \boldsymbol{r}) \cdot \sigma^2 \cdot \Sigma(\boldsymbol{x}, \boldsymbol{r})^T$
3: **return** $\Delta\boldsymbol{\mu}(\boldsymbol{x}), \Delta\Sigma(\boldsymbol{x})$

Despite the conceptual and computational complexity of ES, it offers a well-defined concept for information gained from function evaluations, which can be meaningfully traded off against other quantities, such as the evaluations' cost.

3. Reasoning across dataset subsets

The runtime of machine learning algorithms usually scales polynomially with the number of data points N_{sub}, i.e. $\mathcal{O}(N^{\alpha}_{\text{sub}})$ for some positive α. While the computational cost of training grows, the loss of machine learning methods usually decreases with the number of training samples. The computational cost is often largely independent of the hyperparameter values, but the loss depends crucially on them (which is the reason we want to optimize them in the first place).

For an intuition on how performance changes with dataset size, we evaluated a grid of 400 configurations of a support vector machine (SVM) on subsets of the MNIST dataset (LeCun et al., 2001); MNIST has $N = 50\,000$ data points and we evaluated relative subset sizes $s := {}^{N_{sub}}/_{N} \in \{{}^{1}/_{512}, {}^{1}/_{256}, {}^{1}/_{128}, \ldots, {}^{1}/_{4}, {}^{1}/_{2}, 1\}$.

Figure 1 visualizes the validation error (top) and training time (bottom) of these configurations on $s = {}^{1}/_{128}$, ${}^{1}/_{16}$, ${}^{1}/_{4}$, and 1. Evidently, just ${}^{1}/_{128}$ of the dataset is quite representative and sufficient to locate a reasonable configuration. Additionally, there are no deceiving local optima on smaller subsets. The training time, however increases substantially with the number of datapoints, single configurations take only a few seconds to train on $s = {}^{1}/_{128}$ but can take up to a few hours on the full dataset. Based on these observations, we expect that relatively small fractions of the dataset yield representative performances and therefore vary our relative size parameter s on a logarithmic scale.

4. Previous work

Making use of dataset subsets to seed up hyperparameter optimization has been investigated by others before. In this Section we will present two approaches that are similar to ours, namely Multi Task Bayesian Optimization and Hyperband.

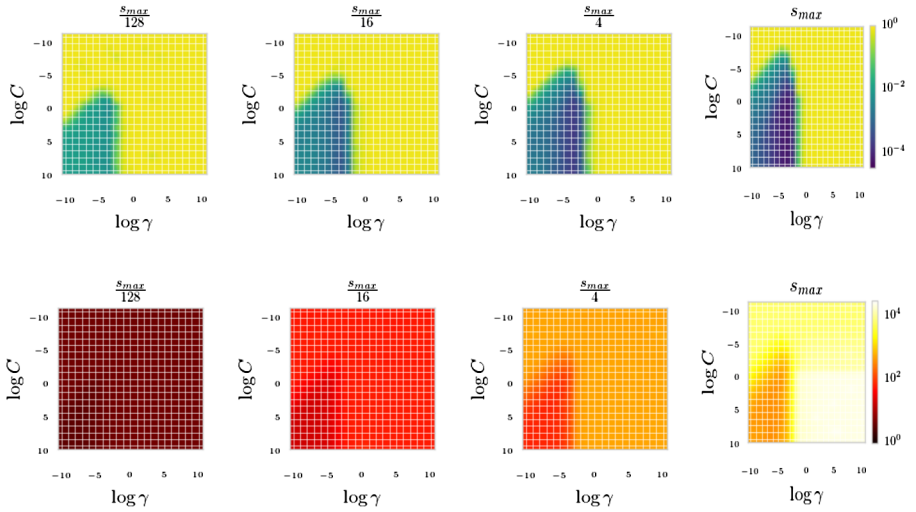

FIG 1. *Validation error (top row) and training time (bottom row) of a grid of* 400 *SVM configurations (20 settings of each of the regularization parameter C and kernel parameter γ, both on a log-scale in $[-10, 10]$) for subsets of the MNIST dataset (LeCun et al., 2001) of various sizes N_{sub}. Small subsets are quite representative: The validation error of bad configuration (yellow) remains constant at around* 0.9*, whereas the region of good configurations (blue) does not change drastically with s. Both hyperparameters also have an influence on the training time, even though it is less dramatic than the influence of the dataset size.*

4.1. Multi-task Bayesian optimization

The *Multi-Task* Bayesian optimization (MTBO) method by Swersky et al. (2013) refers to a general framework for optimizing in the presents of different, but correlated tasks. Given a set of such tasks $\mathbb{T} = \{1, \dots, T\}$, the objective function $f : \mathbb{X} \times \mathbb{T} \to \mathbb{R}$ corresponds to evaluating a given $\boldsymbol{x} \in \mathbb{X}$ on one of the tasks $t \in \mathbb{T}$. The relation between points in $\mathbb{X} \times \mathbb{T}$ is modeled via a GP using a product kernel:

$$k_{\text{MT}}((\boldsymbol{x}, t), (\boldsymbol{x}', t')) = k_T(t, t') \cdot k_{5/2}(\boldsymbol{x}, \boldsymbol{x}') \,. \tag{5}$$

The kernel k_T is represented implicitly by the Cholesky decomposition of $k(\mathbb{T}, \mathbb{T})$ whose entries are sampled via MCMC together with the other hyperparameters of the GP. By considering the distribution over the optimum on the target task $t_* \in \mathbb{T}$, $p^{t_*}_{\text{min}}(\boldsymbol{x} \mid \mathcal{D}) := p(\boldsymbol{x} \in \arg\min_{\boldsymbol{x}' \in \mathbb{X}} f(\boldsymbol{x}', t = t_*) \mid \mathcal{D})$, and computing any information w.r.t. it, Swersky et al. (2013) use the information gain per unit cost as their acquisition function[2]:

[2]In fact, Swersky et al. (2013) deviated slightly from this formula (which follows the ES approach of Hennig and Schuler (2012)) by considering the difference in information gains in $p^{t_*}_{\text{min}}(\boldsymbol{x} \mid \mathcal{D})$ and $p^{t_*}_{\text{min}}(\boldsymbol{x} \mid \mathcal{D} \cup \{(\boldsymbol{x}, y)\})$. They stated this to work better in practice, but we did not find evidence for this in our experiments and thus, for consistency, use the variant presented here throughout.

$$a_{\mathrm{MT}}(\boldsymbol{x}, t) := \frac{1}{c(\boldsymbol{x}, t)} \mathbb{E}_{p(y|\boldsymbol{x},t,\mathcal{D})} \left[\int p_{\min}^{t_*}(\boldsymbol{x}' \mid \mathcal{D}') \cdot \log \frac{p_{\min}^{t_*}(\boldsymbol{x}' \mid \mathcal{D}')}{u(\boldsymbol{x}')} \, \mathrm{d}\boldsymbol{x}' \right], \quad (6)$$

where $\mathcal{D}' = \mathcal{D} \cup \{(\boldsymbol{x}, t, y)\}$. The expectation represents the information gain on the target task averaged over the possible outcomes of $f(\boldsymbol{x}, t)$ based on the current model. If the cost $c(\boldsymbol{x}, t)$ of a configuration $\boldsymbol{x}$ on task t is not known a priori it can be modelled the same way as the objective function.

This model supports machine learning hyperparameter optimization for large datasets by using discrete dataset sizes as tasks. Swersky et al. (2013) indeed studied this approach for the special case of $\mathbb{T} = \{0, 1\}$, representing a small and a large dataset; this will be a baseline in our experiments.

4.2. Hyperband

Hyperband (Li et al., 2017) is a multi-arm bandit strategy based on random search. It was developed concurrently with our method[3], and, similar to it, makes uses of the principle that hyperparameter configurations performing poorly on subsets of the data are very likely to also perform poorly on the full datasets.

In each iteration i, Hyperband samples n_i configurations randomly and uses successive halving (Jamieson and Talwalkar, 2016) to discard hyperparameter configurations after evaluating them on subsets of the data. Hyperband iteratively calls successive halving with different tradeoffs between breadth (i.e., number of configurations) and depth (i.e., subset size), such that each iteration takes roughly the same time. Hyperband returns its first suggested hyperparameter setting after its first run of successive halving.

5. Fabolas

Here, we introduce our new approach for FAst Bayesian Optimization on LArge data Sets (Fabolas). While traditional Bayesian hyperparameter optimizers model the loss of machine learning algorithms on a given dataset as a blackbox function f to be minimized, Fabolas models loss and computational cost *across dataset size* and uses these models to carry out Bayesian optimization with an extra degree of freedom. The blackbox function $f : \mathbb{X} \times \mathbb{R} \to \mathbb{R}$ now takes another input representing the data subset size; we will use relative sizes $s = N_{sub}/N \in [0, 1]$, with $s = 1$ representing the entire dataset. While the eventual goal is to minimize the loss $f(\boldsymbol{x}, s = 1)$ for the entire dataset, evaluating f for smaller s is usually cheaper, and the function values obtained correlate across s. Unfortunately, this correlation structure is initially unknown, so the challenge is to design a strategy that trades off the cost of function evaluations against the benefit of learning about the scaling behavior of f and, ultimately, about which configurations work best on the full dataset. Following the nomenclature of Williams et al. (2000), we call $s \in [0, 1]$ an *environmental variable* that can

[3]The first publications on the two methods were Klein et al. (2015) and Li et al. (2016), respectively.

be changed freely *during* optimization, but that is set to $s = 1$ (i.e., the entire dataset size), at evaluation time.

We propose a principled rule for the automatic selection of the next $(\boldsymbol{x}, s)$ pair to evaluate. In a nutshell, where standard Bayesian optimization would always run configurations on the full dataset, we use ES to reason about, how much can be learned about performance on the full dataset from an evaluation at any s. In doing so, FABOLAS automatically determines the amount of data necessary to (usefully) extrapolate to the full dataset.

5.1. Modelling loss and computational cost

To transfer the insights from the illustrative example in Section 3 into a formal model for the loss and cost across subset sizes, we extend the GP model by an additional input dimension, namely $s \in [0, 1]$. This allows the surrogate to extrapolate to the full data set at $s = 1$ without necessarily evaluating there. We chose a factorized kernel, consisting of the standard stationary kernel over hyperparameters, multiplied with a finite-rank ("degenerate") covariance function in s:

$$k\left((\boldsymbol{x}, s), (\boldsymbol{x}', s')\right) = k_{5/2}\left(\boldsymbol{x}, \boldsymbol{x}'\right) \cdot \left(\phi^T(s) \cdot \Sigma_\phi \cdot \phi(s')\right) . \tag{7}$$

Since any choice of the basis function ϕ yields a positive semi-definite covariance function, this provides a flexible language for prior knowledge relating to s. We use the same form of kernel to model the loss f and cost c, respectively, but with different basis functions ϕ_f and ϕ_c.

The loss of a machine learning algorithms usually decreases with more training data. We incorporate this behavior by choosing $\phi_f(s) = (1, (1 - s)^2)^T$ to enforce monotonic predictions with an extremum at $s = 1$. This kernel choice is equivalent to Bayesian linear regression with these basis functions and Gaussian priors on the weights.

To model computational cost c, we note that the complexity usually grows with relative dataset size s. To fit polynomial complexity $\mathcal{O}(s^\alpha)$ for arbitrary α and simultaneously enforce positive predictions, we model the log-cost and use $\phi_c(s) = (1, s)^T$. As above, this amounts to Bayesian linear regression with shown basis functions.

Figure 2 shows some examples of our basis functions. Figure 3 visualizes the scaling of loss and cost with s for some random SVM configurations from Section 3.

5.2. Algorithm description

FABOLAS starts with an initial design, described in more detail in Section 5.3. Afterwards, at the beginning of each iteration it fits GPs for loss and computational cost across dataset sizes s using the kernel from Eq. 7. Then, capturing the distribution of the optimum for $s = 1$ using $p_{\min}^{s=1}(\boldsymbol{x} \mid \mathcal{D}) := p(\boldsymbol{x} \in \arg\min_{\boldsymbol{x}' \in \mathbb{X}} f(\boldsymbol{x}', s = 1) \mid \mathcal{D})$, it selects the maximizer of the following acquisition function to trade off information gain versus cost:

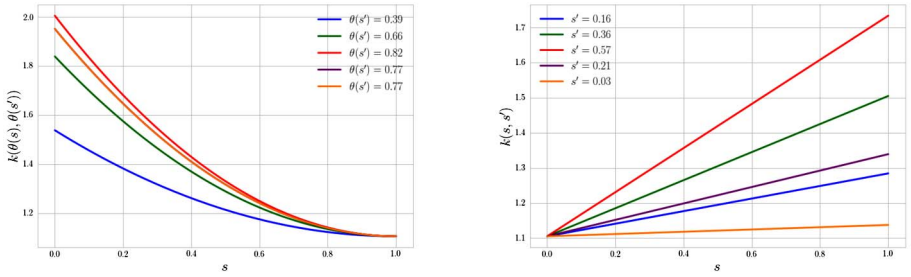

FIG 2. *Kernel values across different s with quadratic basis functions to model the objective function (left) and linear basis function to model the cost (right).*

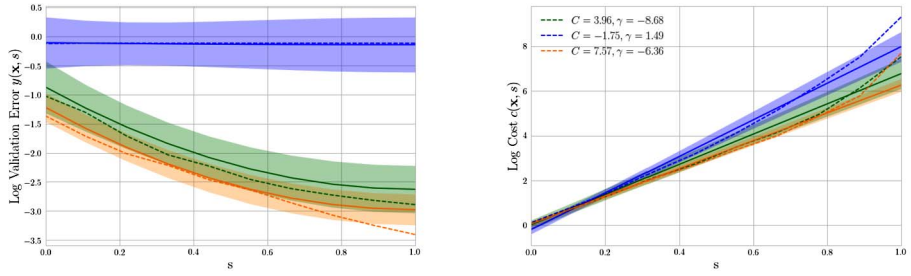

FIG 3. *Gaussian process model prediction (solid line) and the actual values (dashed) for the objective function (left) and the cost function (right) based on a dot product of our Bayesian linear regression kernel and a Matern kernel.*

$$a_{\mathrm{F}}(\boldsymbol{x}, s) := \frac{\mathbb{E}_{p(y|\boldsymbol{x},s,\mathcal{D})}\left[\int p_{\min}^{s=1}(\boldsymbol{x}' \mid \mathcal{D}') \cdot \log \frac{p_{\min}^{s=1}(\boldsymbol{x}'|\mathcal{D}')}{u(\boldsymbol{x}')} \, \mathrm{d}\boldsymbol{x}'\right]}{c(\boldsymbol{x}, s) + c_{\text{overhead}}}, \tag{8}$$

where $\mathcal{D}' = \mathcal{D} \cup \{(\boldsymbol{x}, s, y)\}$. Algorithm 5 shows pseudocode for FABOLAS. Additionally, we provide an open-source implementation at: https://github.com/automl/RoBO.

Algorithm 5 Fast BO for Large Datasets (FABOLAS)

1: Initialize data $\mathcal{D}_0$ using an initial design.
2: **for** $t = 1, 2, \ldots$ **do**
3: Fit GP models for $f(\boldsymbol{x}, s)$ and $c(\boldsymbol{x}, s)$ on data $\mathcal{D}_{t-1}$
4: Choose $(\boldsymbol{x}_t, s_t)$ by maximizing the acquisition function in Equation 8.
5: Evaluate $y_t \sim f(\boldsymbol{x}_t, s_t) + \mathcal{N}(0, \sigma^2)$, also measuring cost $z_t \sim c(\boldsymbol{x}_t, s_t) + \mathcal{N}(0, \sigma_c^2)$, and augment the data: $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(\boldsymbol{x}_t, s_t, y_t, z_t)\}$
6: Choose incumbent $\hat{\boldsymbol{x}}_t$ based on the predicted loss at $s = 1$ of all $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t\}$.
7: **end for**

Our proposed acquisition function resembles the one used by MTBO (Eq. 6), with two differences: First, MTBO's discrete tasks t are replaced by a continuous dataset size s (allowing to learn correlations without evaluations at $s = 1$, and to choose the appropriate subset size automatically). Second, the prediction of

computational cost is augmented by the overhead of the Bayesian optimization method. This inclusion of the reasoning overhead is important to appropriately reflect the information gain per unit time spent: it does not matter whether the time is spent with a function evaluation or with reasoning about which evaluation to perform. In practice, due to cubic scaling in the number of data points of GPs and the computational complexity of approximating $p_{\min}^{s=1}$, the additional overhead of FABOLAS is within the order of minutes, such that differences in computational cost in the order of seconds become negligible in comparison.[4]

Being an anytime algorithm, FABOLAS keeps track of its incumbent at each time step. To select a configuration that performs well on the full dataset, it predicts the loss of all evaluated configurations at $s = 1$ using the GP model and picks the minimizer. We found this to work more robustly than globally minimizing the posterior mean, or similar approaches.

5.3. Initial design

It is common in Bayesian optimization to start with an initial design of points chosen at random or from a Latin hypercube design to allow for reasonable GP models as starting points. To fully leverage the speedups we can obtain from evaluating small datasets, we bias this selection towards points with small (cheap) datasets in order to improve the prediction for dependencies on s: We draw k random points in $\mathbb{X}$ ($k = 10$ in our experiments) and evaluate them on different subsets of the data (for instance on the support vector machine experiments we used $s \in \{1/64, 1/32, 1/16, 1/8\}$). This provides information on scaling behavior, and, assuming that costs increase linearly or superlinearly with s, these k function evaluations cost less than $\frac{k}{4}$ function evaluations on the full dataset. This is important as the cost of the initial design, of course, counts towards FABOLAS' runtime.

5.4. Implementation details

The presentation of FABOLAS above omits some details that impact the performance of our method. As it has become standard in Bayesian optimization (Snoek et al., 2012), we use Markov-Chain Monte Carlo (MCMC) integration to marginalize over the GPs hyperparameters (we use the emcee package (Foreman-Mackey et al., 2013)). To accelerate the optimization, we use hyper-priors to emphasize meaningful values for the parameters, chiefly adopting the choices of the SPEARMINT toolbox (Snoek et al., 2012): a uniform prior between $[-10, 2]$ for all length scales $\boldsymbol{\lambda}$ in log space, a lognormal prior ($\mu_a = 0$, $\sigma_a^2 = 1$) for the covariance amplitude θ, and a horseshoe prior with length scale of 0.1 for the noise variance σ^2.

[4]The same is true for standard ES and MTBO, but was never exploited as no emphasis was put on the total wall clock time spent for the hyperparameter optimization. We want to emphasize that we express budgets in terms of wall clock time (not function evaluations) since this is natural in most practical applications.

We used the original formulation of ES by Hennig and Schuler (2012) rather than the recent reformulation of PES by Hernández-Lobato et al. (2014). The main reason for this is that the latter prohibits non-stationary kernels due to its use of Bochner's theorem for a spectral approximation. PES could in principle be extended to work for our particular choice of kernels (using an Eigen-expansion, from which we could sample features); since this would complicate making modifications to our kernel, we leave it as an avenue for future work, but note that in any case it may only further improve our method. To maximize the acquisition function we used the blackbox optimizer DIRECT (Jones, 2001).

5.5. Heteroscedastic noise

When making the subset size a parameter, we shuffle the data before an evaluation to prevent bias incurred by repeatedly using the same subset. This shuffling introduces additional noise which could be particularly high for small subsets. To investigate this, we again used the SVM grid of 400 configurations from the Section 3. We repeated each run with a given subset size $K = 10$ times using different subsets, and estimate the observation noise variance at each point as:

$$\sigma^2_{obs}(\boldsymbol{x}_j, s_i) = \frac{1}{K}\sum_{k=1}^{K}(y_k(\boldsymbol{x}_j, s_i) - \mu_{i,j})^2\,, \tag{9}$$

where $\mu_{i,j} = K^{-1}\sum_{k=1}^{K} y_k(\boldsymbol{x}_j, s_i)$. The red points in Figure 4 show the mean and standard deviation of $\sigma^2_{obs}(\boldsymbol{x}_j, s_i)$ over all configurations for all s_i values considered. As expected, the noise decreases with an increasing s, to a point where σ^2_{obs} is zero for $s = 1$.

In contrast to this heteroscedastic noise intrinsic to the random subsampling, the commonly used noise hyperparameter σ^2 of a GP (call it σ^2_{GP}) is fixed and typically estimated using MCMC sampling. To compare these two noise values, for each fixed size s, we also trained a GP to predict losses and plotted its estimates σ^2_{GP} as blue markers in Figure 4. To obtain a good estimate of the GP's hyperparameters, we used a relatively long MCMC chain compared to the ones used during Bayesian optimization. Figure 4 clearly shows that the estimated variance σ^2_{GP} is always larger than the observation noise σ^2_{obs}. This might indicate a certain misfit between the true objective and the space of functions the GP can model (Sollich, 2002). Consequently, we believe the heteroscedastic noise from subsampling the data to often be negligible compared to the noise estimated by the MCMC sampling.

6. Experiments

For our empirical evaluation of Fabolas, we compared it to standard Bayesian optimization (using EI and ES as acquisition functions), MTBO, and Hyperband. For each method, we tracked wall clock time (counting both optimization

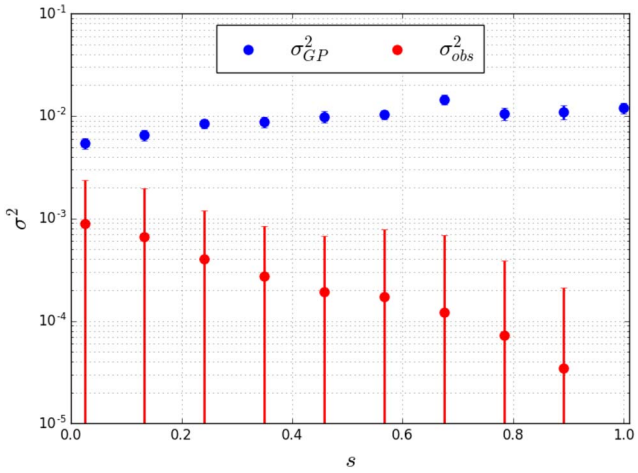

FIG 4. *Evaluating a configurations on a shuffled subset of the data induces an additional noise, σ^2_{obs} that depends on the dataset size s. The noise parameter σ^2_{GP} estimated by MCMC sampling for fixed dataset sizes.*

overhead and the cost of function evaluations, including the initial design), storing the incumbent returned after every iteration. In an offline validation step, we then trained models with all incumbents on the full dataset and measured their test error.[5] To obtain error bars, we performed 10 independent runs of each method with different seeds (except on the grid experiment, where we could afford 30 runs per method) and plot mean and standard deviation for all experiments. Each optimization trajectory starts after all of its runs have evaluated at least one configuration.[6]

We implemented Hyperband following Li et al. (2017) using the recommended setting for the parameter $\eta = 3$ that controls the intermediate subset sizes. For each experiment, we adjusted the budget allocated to each Hyperband iteration to allow the same minimum dataset size as for FABOLAS: 100 datapoints for the support vector machine benchmarks and the maximum batch size for the neural network benchmarks. We also followed the prescribed incumbent estimation after each iteration as the configuration with the best performance on the full dataset size.

6.1. Support vector machine surrogate

First, we considered a benchmark allowing the comparison of the various Bayesian optimization methods on ground truth: we trained a random forest

[5]The residual network in Section 6.4 is an exception: here, we trained networks with the incumbents on the full training set (50000 data points, augmented to 100000 as in the original code) and then measured and plotted performance on the validation set.

[6]This way we avoid assigning a performance to unfinished runs, but we loose information about the runtime distribution across independent runs.

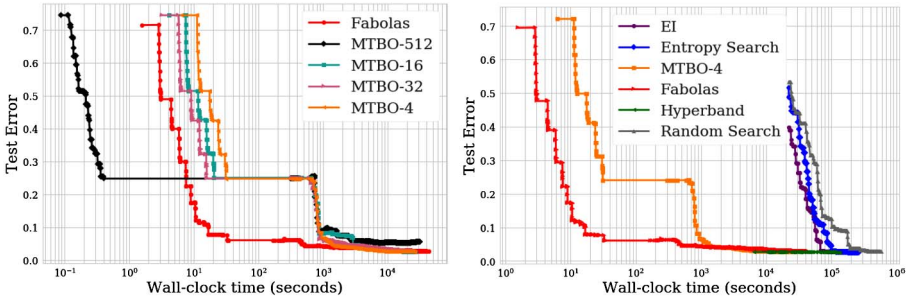

FIG 5. *Evaluation on SVM grid on MNIST. (Left) Test performance over time for variants of MTBO with different dataset sizes for the auxiliary task. (Right) Baseline comparison of test performance of the methods' selected incumbents over time. We only plot means to avoid clutter.*

surrogate (Eggensperger et al., 2015) on our SVM grid on MNIST (described in Section 3), for which we had performed all function evaluations beforehand.

We used this benchmark to adjust the number of data points for MTBO's auxiliary task. Figure 5 (left) evaluates MTBO variants with a single auxiliary task with a relative size of $1/4$, $1/16$, $1/32$, and $1/512$, respectively. We found that the smaller the auxiliary task, the faster MTBO improved initially, but the slower it converged to the optimum. In the plot, MTBO with an auxiliary task of relative size $s = 1/512$ did not achieve the same performance as the other variants in the end. Given the global structure of the error surface (see Figure 1) and the super-linear scaling of the SVM, we chose a very conservative auxiliary task with $s = 1/4$ for the remaining experiments. This value worked consistently in our experience, although the convergence to the best solution in some of the later benchmarks was still rather slow.

At first glance, one might expect many tasks (e.g., with a task for each s value above) to work best, but quite the opposite is true. In preliminary experiments, we evaluated MTBO with up to 3 auxiliary tasks ($s = 1/4$, $1/32$, and $1/512$), but found performance to strongly degrade with a growing number of tasks. We suspect that the $\binom{|T|}{2}$ kernel parameters that have to be learned for the discrete task kernel for $|T|$ tasks are the main reason. If the MCMC sampling is too short, the correlations are not appropriately reflected, especially in early iterations; and an adjusted longer sampling creates a large computational overhead that dominates wall-clock time. We consistently obtained the best performance with only one auxiliary task.

We can now proceed to compare the different methods on this benchmark. The right panel of Figure 5 shows results using EI, ES, random search, Hyperband, MTBO and FABOLAS. EI and ES performed equally well and found the best configuration (which yields an error of 0.014, or 1.4%) after around 10^5 seconds, roughly three times faster than random search. Hyperband outperformed EI and ES by roughly one order of magnitude. MTBO achieves good performance faster, requiring only around 2×10^3 seconds to find close-to-optimal

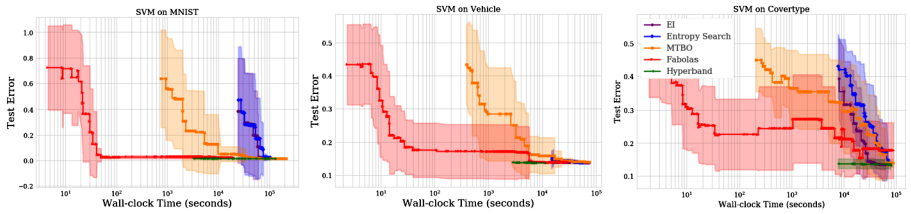

FIG 6. *SVM hyperparameter optimization on the datasets MNIST(left), vehicle (middle) and covertype (right). At each time, the plots show test performance of the methods' respective incumbents.* FABOLAS *can find good configurations between 10 and 1000 times faster than the other methods, but the is not always the fastest to find the true optimum.*

solutions. FABOLAS was roughly another order of magnitude faster than MTBO in finding good configurations, and found close-to-optimal solutions at the same time.

6.2. Support vector machines

For a more realistic scenario, we optimized the same SVM hyperparameters (see Table 1) without a surrogate on MNIST and two other prominent UCI datasets (gathered from OpenML (Vanschoren et al., 2014)), vehicle registration (Siebert, 1987) and forest cover types (Blackard and Dean, 1999) with more than 50000 data points. Training SVMs on these datasets can take several hours, and Figure 6 shows that FABOLAS found good configurations for them between 10 and 1000 times faster than the other methods. On the other hand, both FABOLAS and MTBO sometimes converged more slowly to the true optimum after their initial improvement. This could be a consequence of the GP model and the respective assumptions about the correlation across dataset sizes. Hyperband constitutes a very competitive optimizer on these benchmarks; the super-linear complexity of the SVM and lower cost of good configurations allow Hyperband to recommend its first incumbent faster than the BO methods operating on the full data set.

TABLE 1
Hyperparameters for all support vector machine tasks.

Hyperparameter	lower bound	upper bound	log
Regularization C	e^{-10}	e^{10}	X
Kernel parameter γ	e^{-10}	e^{10}	X

6.3. Convolutional neural networks

Convolutional neural networks (CNNs) have shown superior performance on a variety of computer vision and speech recognition benchmarks, but finding good hyperparameter settings remains challenging, and almost no theoretical guarantees exist. Tuning CNNs for modern, large datasets is often infeasible

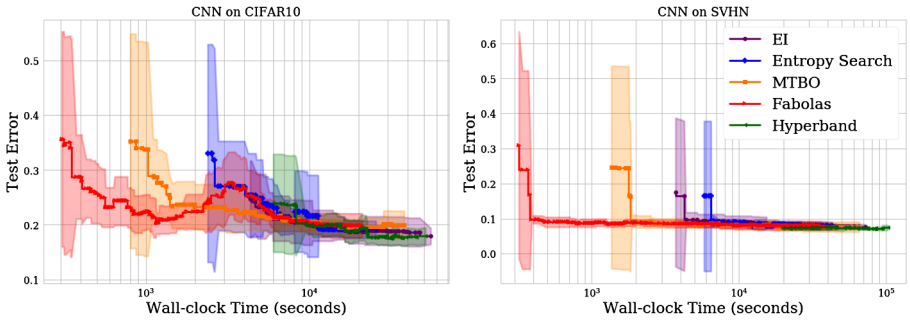

FIG 7. *Test performance of a convolutional neural network on CIFAR10 (left) and SVHN (right).*

via standard Bayesian optimization; in fact, this motivated the development of FABOLAS.

We experimented with hyperparameter optimization for CNNs on two well-established object recognition datasets, namely CIFAR10 (Krizhevsky, 2009) and SVHN (Netzer et al., 2011). We used the same setup for both datasets (a CNN with three convolutional layers, with batch normalization (Ioffe and Szegedy, 2015) in each layer, optimized using Adam (Kingma and Ba, 2014)). We considered a total of five hyperparameters: the initial learning rate, the batch size and the number of units in each layer (see Table 2).

TABLE 2
Hyperparameters for the convolutional neural network task.

Hyperparameter	lower bound	upper bound	log
Initial learning rate	10^{-6}	10^{0}	X
Batch size	32	512	
# units layer 1	2^4	2^8	X
# units layer 2	2^4	2^8	X
# units layer 3	2^4	2^8	X

For CIFAR10, we used 40000 images for training, 10000 to estimate validation error, and the standard 10000 hold-out images to estimate the final test performance of incumbents. For SVHN, we used 6000 of the 73257 training images to estimate validation error, the rest for training, and the standard 26032 images for testing.

The results in Figure 7 show that—compared to the SVM tasks—FABOLAS' speedup was smaller because CNNs scale linearly in the number of datapoints. Nevertheless, it found good configurations about 10 times faster than vanilla Bayesian optimization. For the same reason of linear scaling, Hyperband was substantially slower than vanilla Bayesian optimization to make a recommendation, but it did find good hyperparameter settings when given enough time.

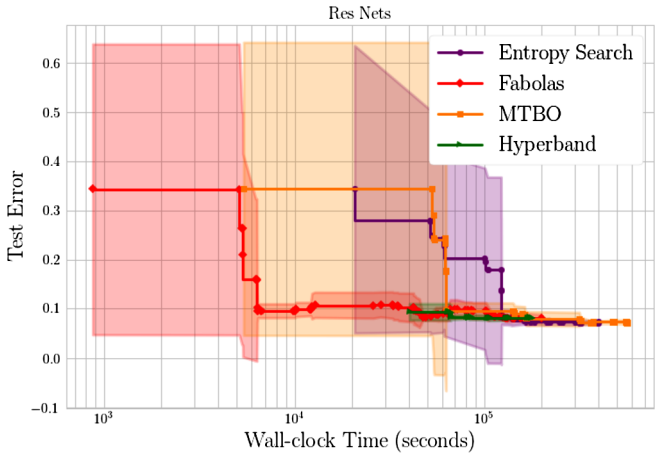

FIG 8. *Validation performance of a residual network on CIFAR10.*

6.4. Residual neural networks

In the final experiment, we evaluated the performance of our method further on a more expensive benchmark, optimizing the validation performance of a deep residual network on the CIFAR10 dataset, using the original architecture from He et al. (2015). As hyperparameters we exposed the learning rate, L_2 regularization, momentum and the factor by which the learning rate is multiplied after 41 and 61 epochs (see Table 3).

TABLE 3
Hyperparameters for the deep residual network task.

Hyperparameter	lower bound	upper bound	log
Learning rate	10^{-6}	1	X
L_2 regularization	10^{-6}	1	X
Learning rate factor	10^{-4}	1	X
Momentum	0.1	0.999	

Figure 8 shows that FABOLAS found configurations with reasonable performance roughly 10 times faster than ES and MTBO. As in the previous convolutional neural network experiment Hyperband's first recommendation for an incumbent takes longer than for the Bayesian optimization methods. However, after the first round of successive halving it already found a very good configuration and only improves slightly in the next iterations.

7. Conclusion

We presented FABOLAS, a new Bayesian optimization method based on Entropy Search that mimics human experts in evaluating algorithms on subsets of the

data to quickly gather information about good hyperparameter settings. FABOLAS extends the standard way of modelling the objective function by treating the dataset size as an additional continuous input variable. This allows the incorporation of strong prior information. It models the time it takes to evaluate a configuration and aims to evaluate points that yield—per time spent—the most information about the globally best hyperparameters for the full dataset. In various hyperparameter optimization experiments using support vector machines and deep neural networks, FABOLAS often found good configurations 10 to 100 times faster than the related approach of Multi-Task Bayesian optimization, Hyperband and standard Bayesian optimization. Our open-source code is available at `https://github.com/automl/RoBO`, along with scripts for reproducing our experiments.

In future work, we plan to expand our algorithm to model other environmental variables, such as the resolution size of images, the number of classes, and the number of epochs, and we expect this to yield additional speedups. Since our method reduces the cost of individual function evaluations but requires more of these cheaper evaluations, we expect the cubic complexity of Gaussian processes to become the limiting factor in many practical applications. We therefore plan to extend this work to other model classes, such as Bayesian neural networks (Neal, 1996; Hernández-Lobato and Adams, 2015; Blundell et al., 2015; Springenberg et al., 2016; Klein et al., 2017b), which may lower the computational overhead while having similar predictive quality.

Acknowledgments

This work has partly been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721, by the European Commission under grant no. H2020-ICT-645403-ROBDREAM, and by the German Research Foundation (DFG) under Priority Programme Autonomous Learning (SPP 1527, grant BR 3815/8-1 and HU 1900/3-1).

References

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. 2017a.

G. Montavon, G. Orr, and K.-R. Müller, editors. *Neural Networks: Tricks of the Trade - Second Edition*. LNCS. Springer, 2012.

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.

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), volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer-Verlag, 2011. MR2470700

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. MR2913701

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.

R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pages 199–207. Omnipress, 2014.

J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pages 115–123. Omnipress, 2014.

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 /ginsInternational Conference on Advances in Neural Information Processing Systems (NIPS'13)*, pages 2004–2012, 2013.

K. Swersky, J. Snoek, and R. Adams. Freeze-thaw bayesian optimization. arXiv:1406.3896, 2014.

J. Snoek, K. Swersky, R. Zemel, and R. Adams. Input warping for Bayesian optimization of non-stationary functions. In E. Xing and T. Jebara, editors, *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, pages 1674–1682. Omnipress, 2014.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, Prabhat, and R. Adams. Scalable Bayesian optimization using deep neural networks. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, volume 37, pages 2171–2180. Omnipress, 2015.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Q. Yang and M. Wooldridge, editors, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3460–3468, 2015.

L. Bottou. Stochastic gradient tricks. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks, Tricks of the Trade, Reloaded*. Springer, 2012.

K. Kandasamy, G. Dasarathy, J. Oliva, J. Schneider, and B. Póczos. Gaus-

sian process optimisation with multi-fidelity evaluations. In *Proceedings of the 30th /International Conference on Advances in Neural Information Processing Systems (NIPS'30)*, 2016.

M. U. Gutmann and J. Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research*, 17(125):1–47, 2016. MR3555016

T. Nickson, M. A Osborne, S. Reece, and S. Roberts. Automated machine learning on big data using stochastic algorithm tuning. *CoRR*, 2014.

T. Krueger, D. Panknin, and M. Braun. Fast cross-validation via sequential testing. *JMLR*, 2015. MR3417778

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

P. Hennig and C. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 98888(1):1809–1837, 2012. MR2956343

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. arXiv:1012.2599, 2010.

B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

J. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust bayesian neural networks. In D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NIPS'16)*, 2016.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006. MR2514435

B. Matérn. Spatial variation. *Meddelanden fran Statens Skogsforskningsinstitut*, 1960. MR0169346

D. J. C. MacKay and R. M. Neal. Automatic relevance detection for neural networks. Technical report, University of Cambridge, 1994.

J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117–129), 1978.

N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*, pages 1015–1022. Omnipress, 2010.

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. Lawrence, and K. Weinberger, editors, *Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NIPS'14)*, 2014.

Thomas P. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the 30th conference on Uncertainty in Artificial Intelligence (UAI'01)*. Morgan Kaufmann Publishers Inc., 2001.

J. Cunningham, P. Hennig, and S. Lacoste-Julien. Approximate gaussian integration using expectation propagation. pages 1–11, January 2012. MR1576213

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In S. Haykin and B. Kosko, editors, *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001. URL http://www.iro.umontreal.ca/~lisa/pointeurs/lecun-01a.pdf.

A. Klein, S. Bartels, S. Falkner, P. Hennig, and F. Hutter. Towards efficient bayesian optimization for big data. In *NIPS 2015 Bayesian Optimization Workshop*, December 2015.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Efficient hyperparameter optimization and infinitely many armed bandits. 2016.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

B. Williams, T. Santner, and W. Notz. Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica*, 2000. MR1804554

D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman. emcee: The MCMC Hammer. *PASP*, 2013.

D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001. MR1869398

Peter Sollich. Gaussian process regression with mismatched models. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15)*, pages 1114–1120. AAAI Press, 2015.

J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, June 2014.

J. P. Siebert. *Vehicle Recognition Using Rule Based Methods*. Turing Institute, 1987.

J. A. Blackard and D. J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Comput. Electron. Agric.*, 1999.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, volume 37. Omnipress, 2015.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv:1412.6980, 2014.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, 2015.

R. Neal. Bayesian learning for neural networks. *PhD thesis, University of Toronto*, 1996.

J. Hernández-Lobato and R. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, volume 37. Omnipress, 2015.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, volume 37, pages 1613–1622. Omnipress, 2015.

A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017b. Published online: iclr.cc.