Bayesian Optimization with Tree-structured Dependencies

Rodolphe Jenatton [1] **Cedric Archambeau** [1] **Javier Gonzalez** [2] **Matthias Seeger** [1]

Abstract

Bayesian optimization has been successfully used to optimize complex black-box functions whose evaluations are expensive. In many applications, like in deep learning and predictive analytics, the optimization domain is itself complex and structured. In this work, we focus on use cases where this domain exhibits a known dependency structure. The benefit of leveraging this structure is twofold: we explore the search space more efficiently and posterior inference scales more favorably with the number of observations than Gaussian Process-based approaches published in the literature. We introduce a novel surrogate model for Bayesian optimization which combines independent Gaussian Processes with a linear model that encodes a tree-based dependency structure and can transfer information between overlapping decision sequences. We also design a specialized two-step acquisition function that explores the search space more effectively. Our experiments on synthetic tree-structured objectives and on the tuning of feedforward neural networks show that our method compares favorably with competing approaches.

1. Introduction

In recent years, Bayesian optimization has gained a growing attention from machine learning experts in, both, academia and industry (Shahriari et al., 2016). It takes the widespread application of machine learning to the next level of sophistication as it enables to automatically fine-tune hyperparameters (Snoek et al., 2012), whether they are parametrizing data pre-processors, models or the learning algorithms. Fine-tuning is essential to obtain state-of-the-art performance with complex machine learning models, such as deep neural networks. Historically, this vital step has been done, either manually, or via regular or random grid search, which can consume vast amounts of human expert time and are wasteful of computing resources. Hence, one of the main benefits of Bayesian optimization is that it removes this burden from the shoulders of the practitioners, who can then focus their attention on more rewarding value-adding tasks.

To set the stage, our goal is to solve a *global optimization* problem:

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}),$$

where $\mathcal{X}$ is the optimization domain and f is a *black-box* function, typically continuous and multimodal. We further assume that querying f is costly. For example, f may be the outcome of a physical experiment or require a large amount of computation. The latter arises when f corresponds to a model selection score for a machine learning model trained on a possibly large dataset.

The protocol for *sequential* Bayesian optimization proceeds as follows (Mockus et al., 1978; Shahriari et al., 2016). Given n noisy evaluations $y_i \approx f(\mathbf{x}_i)$, $i \in \{1, \ldots, n\}$, a surrogate probabilistic model of f is maintained. Our goal is to find a global optimum of f by querying it as few times as possible. The location $\mathbf{x}_{n+1}$ is chosen by maximizing an acquisition function which performs an exploration-exploitation trade-off. A common choice for the surrogate model is a Gaussian process (GP) (Rasmussen & Williams, 2006). For a GP surrogate model, common acquisition functions can be tractably computed and optimized via gradient-based optimization algorithms. While existing Bayesian optimization approaches mitigate the high evaluation cost of f, they suffer from the curse of dimensionality when facing a high-dimensional space $\mathcal{X}$.

In this paper, we introduce a novel methodology able to exploit a given tree-shaped dependency structure on $\mathcal{X}$ by transferring information between overlapping paths. By constructing a surrogate model tailored to the structure, we can reduce the number of *evaluations* of commonly used acquisition functions. The same structure also allows us to take acquisition decisions more efficiently, thus speeding up the *search of candidates*.

Tree-based dependencies occur often in practice. For exam-

[1]Amazon, Berlin, Germany. [2]Amazon, Cambridge, United Kingdom. Correspondence to: Rodolphe Jenatton <jenatton@amazon.de>, Cedric Archambeau <cedrica@amazon.de>, Javier Gonzalez <gojav@amazon.co.uk>, Matthias Seeger <matthias@amazon.de>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

ple, faced with a classification problem, we may want to simultaneously search over many different machine learning models, each coming with their own hyperparameters. Some configurations may also share parameters (e.g., logistic regression with ℓ_2 and ℓ_1 penalty may share the learning rate). These choices could be encoded in a decision tree, where inner nodes select between different models and hyperparameters populate leaf nodes. Another example arises when having to decide on a deep neural network architecture: the size of a layer, choice of activation function, or dropout fraction may depend on the number of layers (Bengio, 2009).

1.1. Baselines and Related Work

A baseline approach to Bayesian optimization in this setting is to ignore the structure of $\mathcal{X}$ and, as a result, choose a GP with covariance kernel $\mathcal{K}(\mathbf{x}, \mathbf{x}')$ defined over the joint input space. When comparing a pair of points, all coordinates are taken into account. While easy to run in existing Bayesian optimization toolboxes, this approach can be highly inefficient. Not only do we encounter a cost of $\mathcal{O}(n^3)$ after n acquisitions due to the global nature of the GP, but we also suffer from the curse of dimensionality when searching over $\mathcal{X}$. Several authors attempted to design covariance functions that are aware of the structure: Duvenaud et al. (2011) consider kernels with an additive structure, while Swersky et al. (2014a); Hutter & Osborne (2013) introduce the *Arc-kernel*. However, the cost remains $\mathcal{O}(n^3)$.

Another idea is to consider an independent GP for every valid subset of hyperparameters, as proposed by Bergstra et al. (2011). This approach corresponds to having an independent GP per leaf in the dependency tree. It scales as $\mathcal{O}(\sum_p n_p^3)$, where n_p is the number of evaluations at leaf node p and $\sum_p n_p = n$. However, it lacks a mechanism for *information sharing* across the leaves. As we will show, information sharing can be beneficial in order to cut down on the number of evaluations. Moreover, the independent approach requires a sizable number of evaluations at each leaf, which can be problematic when there are many leafs.

Tree-structured dependencies can also be dealt with by assigning default values to coordinates of $\mathbf{x}$ which do not fall into the leaf node under consideration, using a Random Forest model to make this choice (Hutter et al., 2011). This strategy is implemented in the SMAC library.

Finally, Zhang et al. (2016) proposed a dedicated approach to tune data analytic pipelines, via a two-layer Bayesian optimization framework. Their method first uses a parametric model to select some promising algorithms, whose hyperparameters are then refined by a nonparametric model.

1.2. Contributions

First, we introduce a novel Bayesian optimization methodology able to leverage conditional dependencies between hyperparameters. To this end, we build a tree-structured surrogate model, with separate GPs at the leaf nodes, and random linear (or constant) functions at the inner nodes. This allows us to transfer information between leafs that share nodes on their respective paths, which enables us in turn to efficiently search the space $\mathcal{X}$. Yet, we also retain the beneficial scaling of the independent approach (Bergstra et al., 2011). To our knowledge, no prior published work satisfied these two aspects. The Arc-kernel allows for information sharing, but comes with $\mathcal{O}(n^3)$ computations. Hutter et al. (2011) rely on Random Forests to represent correlations, but no particular sharing mechanism exists.

Second, we introduce a novel acquisition function which is also able to exploit the tree structure and relies on the *expected improvement* (Mockus et al., 1978). The acquisition operates in two steps. We first select the most promising leaf node to score, effectively restricting our attention to a portion of $\mathcal{X}$. We then optimize over all possible anchor points in this portion of space. This can result in a drastic reduction in the number of surrogate functions to optimize over. In comparison, the independent baseline requires to score every anchor point of every leaf in the tree at each iteration.

The paper is organized as follows. In Section 2, we detail our surrogate GP model and inference computations. In Section 3, we show how the model structure gives rise to efficient acquisition optimization. For a range of experiments on simulated and real data, we report in Section 4 favorable comparisons with existing alternatives. We conclude with possible extensions in Section 5.

2. Tree-structured semi-parametric Gaussian process regression model

We assume that the hyperparameters exhibit conditional dependencies, which can be modeled with decision tree $\mathcal{T}$. The set of inner nodes $\mathcal{V}$ is indexed by $v \in \{1, \dots, V\}$; each v has a decision variable and a weight variable c_v. The set of leaf nodes $\mathcal{P}$ is indexed by $p \in \{1, \dots, P\}$. Equivalently, p indexes (unique) paths from the root to a leaf.

Further, let $\mathcal{D}_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the set of observations. We introduce a set of auxiliary variables $\{p_i \mid p_i \in \mathcal{P}\}_{i=1}^n$ that indicate the leaf to which observation i is associated and let $n_p = |\{i \mid p_i = p\}|$. Note that $\mathbf{x}_i \in \mathcal{X}_{p_i}$ since the input domain may vary from one leaf to another.

2.1. Model with Random Inner Node Parameters

We consider a surrogate model that associates with leaf p a latent function g_p with GP prior, whose mean function and covariance kernel are b_p and $\mathcal{K}_p(\mathbf{x}, \mathbf{x}')$. We impose a zero-mean Gaussian prior over the weight vector $\mathbf{c} = [c_1, \ldots, c_V]^T$.

The resulting generative model is given by

$$\begin{aligned} \mathbf{c} &\sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_c), \\ g_p(\cdot) &\sim \mathcal{GP}\big(b_p, \mathcal{K}_p\big), \\ y_i | p_i, \{g_p(\mathbf{x}_i)\}_{p=1}^P, \mathbf{c} &\sim \mathcal{N}(g_{p_i}(\mathbf{x}_i) + \mathbf{z}_{p_i}^\top \mathbf{c}, \sigma^2). \end{aligned} \quad (1)$$

where $\mathbf{\Sigma}_c$, $\{b_p\}_{p=1}^P$ and σ^2 are the prior covariance, the scalar offsets and the noise variance. Vector $\mathbf{z}_p \in \{0,1\}^V$ is a binary mask that activates the weights of the inner node decision variables on the path to leaf node p. In other words, $(\mathbf{z}_p)_v = 1$ iff v lies on the path from the root to p. The prior covariance $\mathbf{\Sigma}_c$ will be diagonal in our experiments.

When $\mathbf{c} = \mathbf{0}$, model (1) boils down to assuming P independent GPs. While inference only scales as $\mathcal{O}(\sum_p n_p^3)$ in this case, information is not transferred between overlapping paths. Introducing the weight vector $\mathbf{c}$ allows us to couple inference for such paths, while keeping the favorable scaling and better exploring the optimization space (see Section 4).

Next, we show how to perform efficient inference in this model and give an interpretation of the induced kernel when computing the marginal likelihood. Posterior inference over the surrogate models $\{g_p(\cdot)\}$ and the random weights $\mathbf{c}$ is needed to compute the acquisition functions (see Section 3).

2.2. Posterior Inference

Before starting, we need some notation. Let $\mathbf{y} \in \mathbb{R}^n$ be the vector of all observations and $\mathbf{g} \in \mathbb{R}^n$ the vector of latent function values at $\{\mathbf{x}_i\}_{i=1}^n$. Further, let $I_p = \{i \mid p_i = p\}$, noting that $n_p = |I_p|$. We partition the data accordingly, so that $\mathbf{y}_p = [y_i]_{i \in I_p}$, and similarly $\mathbf{g}_p = [g_p(\mathbf{x}_i)]_{i \in I_p}$. Also, we define the matrix $\mathbf{Z}_p = \mathbf{z}_p \mathbf{1}_{n_p}^\top \in \mathbb{R}^{V \times n_p}$, where $\mathbf{1}_{n_p} = [1] \in \mathbb{R}^{n_p}$, and the vector $\mathbf{b}_p = b_p \mathbf{1}_{n_p} \in \mathbb{R}^{n_p}$.

The joint distribution $P(\mathbf{y}, \mathbf{g}, \mathbf{c})$ of our model is given by

$$P(\mathbf{c}) \prod_p \mathcal{N}(\mathbf{g}_p; \mathbf{b}_p, \mathbf{K}_p) \mathcal{N}(\mathbf{y}_p; \mathbf{g}_p + \mathbf{Z}_p^\top \mathbf{c}, \sigma^2 \mathbf{I}_{n_p}), \quad (2)$$

where $\mathbf{K}_p = [\mathcal{K}_p(\mathbf{x}_i, \mathbf{x}_j)]_{i,j \in I_p}$ are kernel matrices, with the prior $P(\mathbf{c}) = \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{\Sigma}_c)$. Our goal is to obtain the posterior process $P(g_p(\cdot) | \mathbf{c}, \mathbf{y}_p)$ and the posterior distribution $P(\mathbf{c} | \mathbf{y})$.

We can directly read off the posterior over the latent functions and parameters after rewriting the joint distribution into the following form (see Section 2 of the Appendix for details):

$$P(\mathbf{y}) P(\mathbf{c} | \mathbf{y}) \prod_p P(\mathbf{g}_p | \mathbf{c}, \mathbf{y}_p).$$

First, we obtain the posterior GP over the latent functions:

$$g_p(\cdot) | \mathbf{c}, \mathbf{y}_p \sim \mathcal{GP}\big(m_p(\cdot), S_p(\cdot, \cdot)\big),$$

where $m_p(\mathbf{x}) = \mathbf{k}_p(\mathbf{x})^\top \mathbf{M}_p^{-1}(\mathbf{y}_p - \mathbf{Z}_p^\top \mathbf{c} - \mathbf{b}_p) + b_p$, $S_p(\mathbf{x}, \mathbf{x}') = \mathcal{K}_p(\mathbf{x}, \mathbf{x}') - \mathbf{k}_p(\mathbf{x})^\top \mathbf{M}_p^{-1} \mathbf{k}_p(\mathbf{x}')$ and $\mathbf{M}_p = \mathbf{K}_p + \sigma^2 \mathbf{I}_{n_p}$.

Next, we obtain the posterior for the weights $\mathbf{c}$:

$$\mathbf{c} | \mathbf{y} \sim \mathcal{N}(\mathbf{\Lambda}_c^{-1} \mathbf{f}_c, \mathbf{\Lambda}_c^{-1}),$$

where $\mathbf{f}_c = \sum_p \mathbf{Z}_p \mathbf{M}_p^{-1}(\mathbf{y}_p - \mathbf{b}_p)$ and $\mathbf{\Lambda}_c = \mathbf{\Sigma}_c^{-1} + \sum_p \mathbf{Z}_p \mathbf{M}_p^{-1} \mathbf{Z}_p^\top$.

In the sequel, we compute expressions such as $\mathbf{M}_p^{-1}$ and $\log |\mathbf{M}_p|$ by using the Cholesky decomposition $\mathbf{M}_p = \mathbf{L}_p \mathbf{L}_p^\top$. Similarly, the expressions depending on $\mathbf{\Lambda}_c$ are computed using its Cholesky decomposition.

2.3. Marginal Likelihood and Its Interpretation

As shown in Section 2 of the Appendix, we can derive the expression for the log-marginal likelihood $\log P(\mathbf{y})$ in closed form:

$$\begin{aligned} \log P(\mathbf{y}) = &\sum_p \log \mathcal{N}(\mathbf{y}_p; \mathbf{Z}_p^\top \mathbf{c} + \mathbf{b}_p, \mathbf{M}_p) \\ &+ \log \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{\Sigma}_c) - \log P(\mathbf{c} | \mathbf{y}), \end{aligned} \quad (3)$$

The p-dependent terms require computing the Cholesky decompositions of all $\mathbf{M}_p \in \mathbb{R}^{n_p \times n_p}$, whereas the final term needs the Cholesky decomposition of $\mathbf{\Lambda}_c \in \mathbb{R}^{V \times V}$. Therefore, $\log P(\mathbf{y})$ can be computed in $\mathcal{O}(V^3 + \sum_p n_p^3)$. Note that this computation is required for optimizing the hyperparameters of the GPs.

We can also obtain an interesting interpretation for the induced kernel of the marginal likelihood by computing it in a different way. Let $\mathbf{Z} = [\mathbf{Z}_p] \in \mathbb{R}^{V \times n}$, $\mathbf{b} = [\mathbf{b}_p] \in \mathbb{R}^n$ and $\mathbf{K}^{\text{block}} \in \mathbb{R}^{n \times n}$ the block-diagonal matrix with blocks $\mathbf{K}_p$. With these notations, it can be shown that $P(\mathbf{y}_p | \mathbf{c}) = \mathcal{N}(\mathbf{y}_p | \mathbf{Z}_p^\top \mathbf{c} + \mathbf{b}_p, \mathbf{M}_p)$. Integrating out $\mathbf{c}$ leads to

$$P(\mathbf{y}) = \mathcal{N}\left(\mathbf{b}, \mathbf{Z}^\top \mathbf{\Sigma}_c \mathbf{Z} + \mathbf{K}^{\text{block}} + \sigma^2 \mathbf{I}\right). \quad (4)$$

If we further assume that $\mathbf{\Sigma}_c = \sigma_c^2 \mathbf{I}_V$, then

$$\mathbf{Z}^\top \mathbf{\Sigma}_c \mathbf{Z} = \left[\sigma_c^2 \mathbf{Z}_p^\top \mathbf{Z}_{p'}\right]_{p,p'} = \left[\sigma_c^2 (\mathbf{z}_p^\top \mathbf{z}_{p'}) \mathbf{1}_{n_p} \mathbf{1}_{n_{p'}}^\top\right]_{p,p'}.$$

Hence, the diagonal blocks are proportional to $\mathbf{z}_p^\top \mathbf{z}_p$, which is the length of path p, and the off-diagonal blocks are proportional to $\mathbf{z}_p^\top \mathbf{z}_{p'}$, which is the path overlap length between p and p'. The resulting kernel is thus the intersection kernel (see Shawe-Taylor & Cristianini (2004), Section 9.5).

2.4. Model with Random Linear Inner Node Functions

We have so far associated a random scalar c_v with each inner node. More generally, we can use *linear functions* $\mathbf{c}_v^\top \mathbf{r}_v$, where $\mathbf{c}_v$ is a weight vector and $\mathbf{r}_v \in \mathbb{R}^{d_v}$ is a feature vector. The special case above is obtained with $d_v = 1$ and $\mathbf{r}_v = [1]$.

We collect the weight vectors in $\mathbf{c} = [\mathbf{c}_v] \in \mathbb{R}^d$, where $d = \sum_v d_v$. Let $V_p \subseteq V$ be the set of inner nodes on the path from the root to leaf p. Concatenating the $\mathbf{r}_v$'s, we define the induced feature vector $\mathbf{z}_p = [\mathbf{r}_v]_{v \in V_p}$ such that

$$\mathbf{c}^\top \mathbf{z}_p = \sum\nolimits_{v \in V_p} \mathbf{c}_v^\top \mathbf{r}_v.$$

Hence, the dataset we collect during the optimization is now the extended set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i, p_i, \mathbf{z}_{p_i} = [\mathbf{r}_{v,i}]_{v \in V_{p_i}})\}_{i=1}^n$. It is easy to see that all our results above transfer to this more general case, if only we redefine

$$\mathbf{Z}_p = [\mathbf{z}_{p_i}]_{i \in I_p} = [\mathbf{r}_{v,i}]_{i \in I_p, v \in V_{p_i}} \in \mathbb{R}^{d \times n_p}.$$

Except for an increased dimensionality $d \geqslant V$ of the weight vector $\mathbf{c}$, the extension with random linear inner node functions is not more difficult to implement or run.

In Section 4, we will use $\mathbf{r}_v$ to encode both numerical (i.e., $d_v = 1$) and categorical parameters (via one-hot representations, so that d_v equals the number of categories). In our deep learning use case, parameters such as the learning rate, the number of units and the type of activation functions are encoded via the $\mathbf{r}_v$'s (see Figure 3, bottom). We will refer to the parameter associated with $\mathbf{r}_v$ as a *shared parameter* since it is shared across all the leaves whose paths contain v.

3. Acquisition Functions

Bayesian optimization generally proceeds by discretizing the search space $\mathcal{X}$ into a set of anchor points, for example by using quasi-random sequences (Sobol, 1967). We then maximize an *acquisition function* starting from the most promising anchor point(s), typically with a numerical solver like L-BFGS (Nocedal & Wright, 2006). Acquisition functions are defined in terms of expectations over the surrogate model posterior. Frequently used choices include Thompson sampling (Thompson, 1933), probability of improvement (PI) (Kushner, 1964), expected improvement (EI) (Mockus et al., 1978), or GP-UCB (Srinivas et al., 2010). We will focus on EI in the sequel as it has been shown to perform better than PI. Our initial experiments also showed that Thompson sampling was not performing well.

The naive approach of globally optimizing EI over anchor points does not scale well with a high-dimensional $\mathcal{X}$. In the previous section, we specified a tree-structured model for the (random) surrogate function, with which the *evaluation* of an acquisition function at some $\mathbf{x} \in \mathcal{X}$ is sped up. In this section, we show how the model structure can also be exploited in order to speed up the *optimization* itself.

3.1. Acquisition Strategies

The acquisition function $\alpha(\mathbf{x}|\mathcal{D}_n)$ plays a critical role in Bayesian Optimization as it selects anchor points by performing an exploration-exploitation trade-off. The key question that concerns us is whether we can leverage the explicit structure in high-dimensional structured space in order to make the search more efficient. The *naive* approach ignores structure in the search space, using a surrogate model based on a *global* kernel, like the one proposed by Swersky et al. (2014a). While the design of a kernel that incorporate structure is non-trivial, it is not explicitly used to guide the search and the cost of evaluations still scales as $\mathcal{O}(n^3)$.

As noted above, we can speed up evaluations to $\mathcal{O}(\sum_p n_p^3)$ by adopting an *independent model*, which corresponds to our tree model with $\mathbf{c} = \mathbf{0}$, so that the surrogate models $\{g_p(\cdot)\}_{p=1}^P$ can be learnt and queried independently from each other. With this approach, the search decouples across the leaf nodes and can be parallelized accordingly. However, if acquisitions are done sequentially, then all leafs have to be searched in order to find the overall best candidate. The independent model also fails to represent dependencies between the leaf nodes, so that a larger total number of evaluations may be required to reach a good solution.

Given our tree-structured surrogate model, we can improve on both the naive and the independent approach. The acquisition function becomes $\alpha(\mathbf{x}, p|\mathcal{D}_n)$, p being the leaf node where $\mathbf{x}$ is evaluated. For our model, $\alpha(\mathbf{x}, p|\mathcal{D}_n)$ can be evaluated in $\mathcal{O}(V^3 + \sum_p n_p^3)$, which is often much cheaper than $\mathcal{O}(n^3)$ required in the naive approach, and is comparable to $\mathcal{O}(\sum_p n_p^3)$ for independent. We could maximize $\alpha(\mathbf{x}, p|\mathcal{D}_n)$ separately at each leaf p, and then pick the best candidate across leaf nodes:

$$(\mathbf{x}_\star, p_\star) \in \operatorname*{arg\,max}_{p \in \mathcal{P}, \mathbf{x} \in \mathcal{X}_p} \alpha(\mathbf{x}, p|\mathcal{D}_n).$$

In practice, the set of leaf nodes $\mathcal{P}$ can become large, in which case the requirement to search in every leaf node can be costly. We propose to further exploit the tree structure of our surrogate model in order to speed up the optimization. Namely, our model implies a *path* acquisition function $\alpha(p|\mathcal{D}_n)$. Based on this, we select $p_\star$ and $\mathbf{x}_\star$ in two steps:

$$p_\star = \operatorname*{arg\,max}_{p \in \mathcal{P}} \alpha(p|\mathcal{D}_n), \quad \mathbf{x}_\star \in \operatorname*{arg\,max}_{\mathbf{x} \in \mathcal{X}_{p_\star}} \alpha(\mathbf{x}, p_\star|\mathcal{D}_n).$$

This strategy can greatly speed up the optimization. There are obvious intermediates, such as searching in a subset of top-ranked leafs p, which we defer for future work.

3.2. Two-step Expected Improvement

Given our surrogate model, the EI acquisition function is:

$$\alpha(\mathbf{x}, p|\mathcal{D}_n) = \mathbb{E}\left\{[y_{\min} - g_p(\mathbf{x}) - \mathbf{z}_p^\top \mathbf{c}]_+\right\}, \qquad (5)$$

Table 1. Comparison of different surrogate models and acquisition strategies (see text for details). Here, $\mathcal{M}(\mathcal{X})$ is the complexity of optimizing a surrogate function over the space $\mathcal{X}$. $p_\star$ is the path selected by *tree*, and $\mathcal{X}_{p_\star}$ is the corresponding leaf domain.

	sharing?	complexity
independent	×	$\mathcal{O}\big(\sum_p n_p^3 \cdot \mathcal{M}(\mathcal{X}_p)\big)$
naive	✓	$\mathcal{O}\big((\sum_p n_p)^3 \cdot \mathcal{M}(\mathcal{P}\times\prod_p \mathcal{X}_p)\big)$
tree	✓	$\mathcal{O}\big(V^3 + n_{p_\star}^3 \cdot \mathcal{M}(\mathcal{X}_{p_\star})\big)$

where $[u]_+ = \max\{u, 0\}$, and $y_{\min}$ is the best evaluation so far (across all leafs). The expectation is computed with respect to the posterior of $g_p(\mathbf{x}) + \mathbf{z}_p^\top \mathbf{c}$, which is a GP with mean and covariance functions respectively given by

$$\widetilde{m}_p(\mathbf{x}) = \mathbf{k}_p(\mathbf{x})^\top \mathbf{M}_p^{-1}(\mathbf{y}_p - \mathbf{b}_p) + \mathbf{t}_p(\mathbf{x})^\top \mathbf{\Lambda}_c^{-1}\mathbf{f}_c + b_p,$$
$$\tilde{S}_p(\mathbf{x}, \mathbf{x}') = S_p(\mathbf{x}, \mathbf{x}') + \mathbf{t}_p(\mathbf{x})^\top \mathbf{\Lambda}_c^{-1}\mathbf{t}_p(\mathbf{x}'),$$

where $\mathbf{t}_p(\mathbf{x}) = \mathbf{z}_p - \mathbf{Z}_p\mathbf{M}_p^{-1}\mathbf{k}_p(\mathbf{x})$. We can analytically compute (5), leading to

$$\alpha(\mathbf{x}, p|\mathcal{D}_n) = \tilde{\sigma}_p(\mathbf{x})\left(\xi\Phi(\xi) + \mathcal{N}(\xi; 0, 1)\right),$$

where $\tilde{\sigma}_p(\mathbf{x}) = \{\tilde{S}_p(\mathbf{x}, \mathbf{x})\}^{1/2}$, $\xi = \frac{\tilde{m}_p(\mathbf{x}) - y_{\min}}{\tilde{\sigma}_p(\mathbf{x})}$ and $\Phi(\xi)$ is the CDF of a standard Gaussian.

As noted above, we could optimize $\alpha(\mathbf{x}, p|\mathcal{D}_n)$ at all leaves and pick the overall winner. Instead, we propose a two-step approach, based on a *path* EI acquisition function:

$$\alpha(p|\mathcal{D}_n) = \mathbb{E}\left\{[y_{\min} - b_p - \mathbf{z}_p^\top \mathbf{c}]_+\right\}, \qquad (6)$$

where the expectation is taken with respect to $\mathbf{z}_p^\top \mathbf{c} + b_p \sim \mathcal{N}(\mathbf{z}_p^\top \mathbf{\Lambda}_c^{-1}\mathbf{f}_c + b_p, \mathbf{z}_p^\top \mathbf{\Lambda}_c^{-1}\mathbf{z}_p)$. We first select the path $p_\star = \arg\max_p \alpha(p|\mathcal{D}_n)$, then find $\mathbf{x}_\star$ by maximizing $\alpha(\mathbf{x}, p_\star|\mathcal{D}_n)$ at leaf $p_\star$ only. Our *tree* acquisition strategy is related to the *naive* and *independent* ones in Table 1. Interestingly, *tree* can be faster than *independent* overall. Finally, (6) is easily extended to the case where we have random linear functions at the inner nodes by considering the augmented induced variable $\mathbf{z}_p = [\mathbf{r}_v]_{v\in V_p}$ (see Section 2.4 for details). In particular, the resulting optimization of (6) is carried out jointly over p and $\mathbf{z}_p = [\mathbf{r}_v]_{v\in V_p}$.

4. Experiments

In this section, we conduct two sets of experiments. First, we focus on optimizing synthetic functions designed to have tree-structured conditional relationships. We then consider the tuning of a multi-layer perceptron for binary classification, which we evaluate over a large number of datasets.

Throughout the experiments, we use the following acronyms to refer to the different competing methods: `tree` is our proposed approach, `independent` is a baseline that consider an independent GP for every leaf, `arc` corresponds to (Swersky et al., 2014a), `smac` refers to (Hutter et al., 2011) and `gp-baseline` is a standard GP-based Bayesian optimization solver taken from (GPyOpt, 2016). For `tree`, `independent` and `gp-baseline`, we use 5/2 Matérn kernels. `marginal` is another baseline obtained by replacing the kernel of `gp-baseline` by that stemming from the marginal (4), where $\mathbf{c}$ is viewed as a nuisance variable and integrated out. Finally, `random` is standard random search (Bergstra & Bengio, 2012).

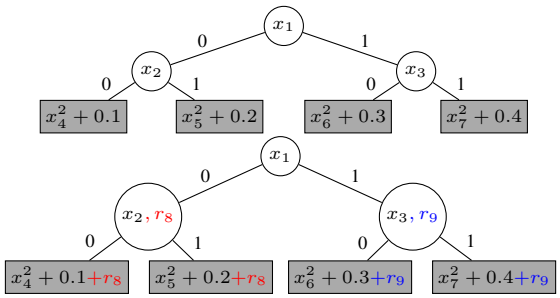

Figure 1. Two examples of functions with tree-structured conditional relationships. Each inner node is a binary variable, and a path in those trees represents successive binary decisions. The leaves contain univariate quadratic functions that are shifted by different constant terms. *Top*: Setting without shared variables. *Bottom*: r_8 and r_9 are shared variables that are common to the functions at the leaves of their respective subtrees. In this example, the shared variables have a linear dependency in the leaf objectives. r_8 and r_9 are encoded following the description of Section 2.4.

Unless otherwise specified, all the results displayed in this section correspond to the means and twice the standard errors computed over 25 random replications. Also, in order to minimize the initialization bias, all methods (except `smac`[1]) start from the same set of random candidates; there is one random candidate drawn per conditional path. Our implementation is in `Python` and we ran the experiments on a fleet of Amazon AWS `c4.8xlarge` machines.

4.1. Synthetic Tree-structured Functions

The functions we consider are defined over binary trees: Each inner node, including the root, corresponds to a binary variable. A path in this tree thus represents successive binary decisions. The leaves contain univariate quadratic functions that are shifted by different constant terms. We give an example of such a function in Figure 1. In the sequel, we study the two functions from Figure 1 (referred to as `small balanced`), in addition to a higher-dimensional version of those, with a depth of 4 and 8 leaves whose constant shifts are $\{a \times 0.1\}_{a=1}^{8}$ (referred to as `large balanced`). In the supplementary material, we provide further results based

[1] We use `https://github.com/sfalkner/pySMAC`. To the best of our knowledge, we cannot specify the starting point.

on unbalanced binary trees of increasing sizes, for which similar conclusions hold. All the non-shared continuous variables x_j's are defined in $[-1, 1]$, while the shared ones are in $[0, 1]$. The best function value will thus always be 0.1.

Those functions encode conditional relationships since given a path p and its leaf ℓ_p, all the binary variables outside of the path p and all the continuous variables defined in the leaves $\ell' \neq \ell_p$ are irrelevant. We report in Figure 2 the optimization results for the different competing methods. We make the following observations:

Approaches blind to structure perform poorly: The results show that, both `gp-baseline` and `random`, which cannot use the conditional structure, do not fare well. As expected, the performance gap widens as the trees get deeper.

Independent vs. tree vs. arc: `independent`, `tree` and `arc` represent 3 ways of increasingly incorporating conditional structure. Indeed, `independent` takes into account the tree structure but does not allow for any sharing of information across different paths, `arc` defines a joint kernel over the union of all the leaves, while `tree` makes intermediate modeling assumptions. We can observe that, thanks to its joint nature, `arc` tends to perform well initially, but it is quickly overtaken by `tree` and later also by `independent` that lags behind because of the absence of sharing, but catches up once sufficient observations were collected. Also, as the dimension of the optimization space gets larger, the performance of `independent` worsens, while that of `tree` is barely affected (we do observe the same scalability with respect to the dimension on unbalanced binary trees, as reported in the supplementary material). At this juncture, we would also like to emphasize that while `independent` catches up with `tree` in some cases, it is more wasteful of resources as it requires to score every leaf at each iteration unlike `tree` (see also Table 1).

Importance of exploiting the latent variables c: It is interesting to observe that `marginal`, which considers $\mathbf{c}$ to be a nuisance variable and integrates it out, performs significantly worse than `tree`. Note that `marginal` cannot de facto be applied in presence of shared variables, which explains why it does not appear in the right panels of Figure 1.

Approach not based on GPs: `smac` is known to be state-of-the-art for optimization tasks in presence of conditional relationships (Eggensperger et al., 2013). In particular, it is known to work better than GP-based approaches, especially when the dimension gets large. We observe in our experiments (e.g., for `large balanced`, 7 categorical and 10 continuous parameters, 2 of which being shared) that `smac` does not reach good solutions on these synthetic tasks.

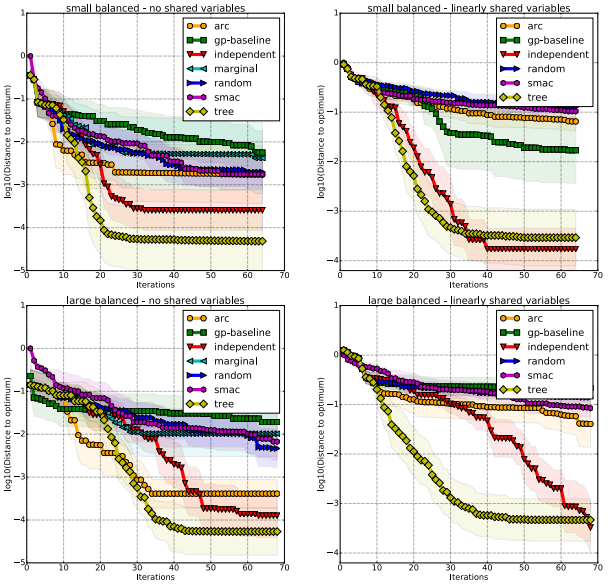

Figure 2. Optimization performance over synthetic tree-structured functions, as measured by the $\log_{10}$ distance to the (known) minimum versus the number of iterations. *Top*: Results for the balanced binary trees displayed in Figure 1, without and with shared variables (respectively left and right). *Bottom*: Results for larger balanced binary trees with depth 4 and 8 leaves. Best seen in color.

4.2. Multi-layer Perceptron Tuning

We now focus our attention on the tuning of a multi-layer perceptron (MLP) for binary classification. The setting we consider is reminiscent of that proposed by Swersky et al. (2014a). We optimize for the number of hidden layers in $\{0, 1, 2, 3, 4\}$, the number of units per layer in $\{1, 2, \ldots, 30\}$ (provided the corresponding layer is activated), the choice of the activation function in $\{$`identity, logistic, tanh, relu`$\}$, which we constrain to be identical across all layers, the amount of ℓ_2 regularization in $[10^{-6}, 10^{-1}]$, the learning rate in $[10^{-5}, 10^{-1}]$ of the underlying Adam solver (Kingma & Ba, 2014), the tolerance in $[10^{-5}, 10^{-2}]$ of the solver (based on relative decrease), and the type of data pre-processing, which can be unit ℓ_2-norm observation-wise normalization, ℓ_∞-norm feature-wise normalization, mean/standard-deviation feature-wise whitening or no normalization at all.

The optimization task can be specified in various ways, resulting in different topologies for the trees of conditional relationships.We consider the two instantiations of conditional relationships illustrated in Figure 3. The first one has all the variables duplicated (top tree), which is similar to how `independent` proceeds. The second one consists in having most of the variables shared (bottom tree). Note that in the two settings, we have one regularization parameter λ_k per number k of hidden layer(s) of the network. We do

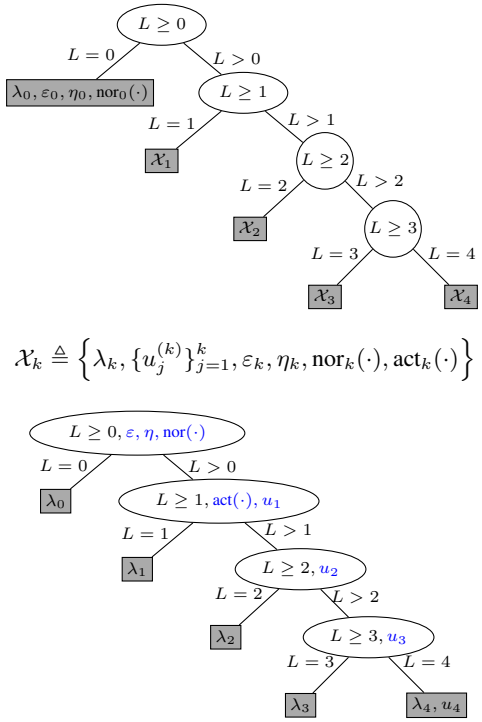

Figure 3. Conditional relationships for MLP tuning. L refers to the number of hidden layers, u_j is the number of units of the j-th layer, λ_k controls the ℓ_2 regularization of a network with k hidden layers, η and ε are respectively the learning rate and stopping criterion of the Adam optimizer, while $\text{nor}(\cdot)$ and $\text{act}(\cdot)$ respectively stand for the normalization of the dataset and the activation function. *Top*: An independent topology where the domain $\mathcal{X}_k$ of each leaf is made of duplicated parameters (indexed by k in the figure). *Bottom*: A topology where the parameters (in blue) are shared across leaves, following Section 2.4..

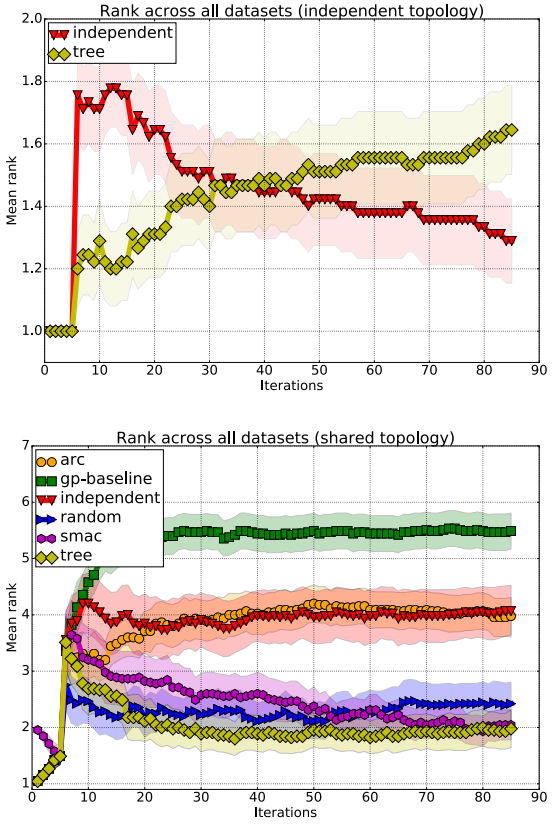

Figure 4. Tuning of a MLP for binary classification. Average rank aggregated over 45 datasets versus the number of iterations (lower is better; see text for details). Shaded regions correspond to two standard errors tube around the average rank. *Top*: Comparison of `independent` with the `tree`-based independent topology (see bottom tree of Figure 3). *Bottom*: Comparison of all the methods based on the shared topology (see top tree of Figure 3). Best seen in color.

so to account for the fact that the λ_k's regularize matrices of different dimensions. In between those two extreme settings, we could consider intermediate modeling assumptions (e.g., a learning rate η_{linear} for the case with no hidden layers and a shared learning rate $\eta_{\text{non-linear}}$ otherwise).

To provide a robust evaluation of the different competing methods, we consider a subset of the datasets from the `Libsvm` repository (Chang & Lin, 2011). More specifically, we consider all the datasets whose number of features is smaller than 10^6, which results in 45 data sets. In absence of pre-defined default train-test split, we took a random $80\%-20\%$ split. To limit the overall computational burden, we cap the training and test set sizes to a maximum of respectively 10^3 and 10^4 instances (randomly selected when the subsampling applies). Note that this subsampling step is not related to a computational limitation of our approach, but is a practical consideration only modifying the properties of the black-box function we optimize. We use the MLP implementation of `scikit-learn` (Pedregosa et al., 2011) and we add a CPU-time constraint of 5 minutes to each evaluation, beyond which the worst classification error 1.0 is returned. Under this constraint, the total computational time of the experiment was roughly 100 CPU days.

We run all the methods for 85 iterations and initialize them with one random choice for each of the 5 conditional paths. We aggregate the average classification errors per dataset by displaying the average rank of each method as a function of the number of iterations. We say that the rank of a method is equal to i if it performs the i^{th} best (see, e.g., Bardenet et al. (2013); Feurer et al. (2015)). We can draw the following conclusions:

Effect of $\mathbf{z}_p^\top \mathbf{c}$ *without* shared variables: The top panel in Figure 4 compares `independent` with `tree`-based method when it is defined on the independent topology shown in Figure 3(top). Since there are no shared variables in the inner nodes, the sharing mechanism of `tree` only

happens via the term $\mathbf{z}_p^\top \mathbf{c}$ which contributes to the mean. As expected, sharing results in `tree` makes faster progress towards the optimum. However, when more observations are collected, `independent` outperforms `tree` because it better explores all the leafs (though, at a higher computational cost; see Table 1). We next show how we can additionally benefit from sharing parameters at inner nodes.

Shared topology: The lower panel in Figure 4 compares all the methods using the shared topology shown in Figure 3(bottom). We found that `arc`, `gp-baseline`, `random` and `smac` all benefitted from running with the shared topology. The results show that `tree` not only greatly improves upon all other GP-based approaches, but also converges faster than `smac` that finally reaches the same level of performance after about 75 iterations. We can observe that a standard GP-based technique that is blind to the conditional structure, like `gp-baseline`, performs poorly. Of independent interest is the comparison of `arc` with `smac`, which was not reported by Swersky et al. (2014a). Finally, it is worth emphasizing that `tree` obtains good results while only modeling shared variables at the inner nodes *in a linear fashion*. This conclusion is in agreement with the recent observations from (Zhang et al., 2016) where linear models lead to good results in the context of the optimization of data analytic pipelines. Next, we discuss an extension to model the shared variables non-linearly.

4.3. Nonlinear Extensions

The approach we have introduced in Section 2.4 can easily be extended to account for non-linearities through the use of *basis expansions*. More specifically, we focus on the use of random Fourier features (Rahimi et al., 2007) that proved successful for large-scale kernel methods (Lu et al., 2014). Combining basis expansion with linear models for Bayesian optimization is by no means new (see (Shahriari et al., 2016) and references therein). We also follow this methodology since it naturally fits our proposed semi-parametric model.

In the supplementary material, we report results on synthetic tree-structured functions where the objectives at the leaves depend now quadratically on the shared variables and on the MLP tuning task. In a nutshell, on the synthetic functions with linearly-dependent shared variables, `tree-nonlinear` converges slower than the linear version `tree`, which might be due to the fact that $\mathbf{c}$ is of higher dimensionality. Moreover, in presence of quadratically-dependent shared variables, we observe that `tree` fails to model adequately the non-linearities, while `tree-nonlinear`, as expected, can. As for the MLP task, we notice that the non-linear extension of `tree` tends to perform worse than its linear counterpart.

5. Concluding Remarks

The black-box functions typically encountered in machine learning rely on incremental learning procedures, such as the application of (stochastic) gradient descent over several epochs. A recent line of work has been focusing on leveraging this property to speed up Bayesian optimization (Swersky et al., 2013; 2014b; Domhan et al., 2014; Li et al., 2016; Klein et al., 2016). In particular, Li et al. (2016) and Klein et al. (2016) have reported state-of-the-art results with methods based respectively on bandits and GPs, exploiting a dynamic subsampling of the training sets.

The goal of our work is orthogonal to this idea and consists instead in efficiently encoding conditional relationships with GPs. We next outline ways of combining our work with the aforementioned subsampling idea:

Combination with Klein et al. (2016): The proposal of Klein et al. (2016) uses some contextual variable (also referred to as *environmental variable*) to encode the subsampling rate of the training set. Let us denote it by $\beta \in [0, 1]$. Klein et al. (2016) define the following joint kernel:

$$\mathcal{K}((\mathbf{x}, \beta), (\mathbf{x}', \beta')) = \mathcal{K}_0(\mathbf{x}, \mathbf{x}') \cdot \mathcal{K}_{\text{context}}(\beta, \beta').$$

The optimization is then driven by a cost-normalized acquisition function $\max_{\beta', \mathbf{x}} \alpha(\mathbf{x}, \beta = 1 | \mathcal{D}_n) / \text{cost}(\mathbf{x}, \beta')$ where both $\mathbf{x}$ and β are sought to perform well on the final task of interest where no subsampling is applied (i.e., $\beta = 1$).

Looking at our case, we could easily replace our kernel $\mathcal{K}_p$ by $\tilde{\mathcal{K}}_p((\mathbf{x}, \beta), (\mathbf{x}', \beta')) \triangleq \mathcal{K}_p(\mathbf{x}, \mathbf{x}') \cdot \mathcal{K}_{\text{context}}(\beta, \beta')$. To apply the two-step procedure, we could normalize (6) by a cost following a separate model (1) where the contextual variable β would be a shared variables at the root. Formally, we could consider a joint path/subsampling selection criterion:

$$(p^\star, \beta^\star) \in \underset{p \in \mathcal{P}, \beta' \in [0,1]}{\arg\max} \frac{\mathbb{E}\{[y_{\min} - b_p - \mathbf{z}_p(\beta = 1)^\top \mathbf{c}]_+\}}{\mathbb{E}\{\mathbf{z}_p(\beta')^\top \mathbf{c}_{\text{cost}}\}},$$

where $\mathbf{z}_p(\beta)$ refers to the feature representation of the path p with context variable β.

Combination with Li et al. (2016): The approach of Li et al. (2016) is based on successive *halving* procedures where a pool containing initially many models is progressively refined and trimmed. The output of their theoretically-justifed algorithm, named `hyperband`, can be seen as triplets $\mathcal{H}_n \triangleq \{(\mathbf{x}_i, y_i, \beta_i)\}_{i=1}^n$ representing all the tested configurations $\mathbf{x}_i$ along with their corresponding evaluations y_i and subsampling rates β_i.

A natural approach to leverage `hyperband` is therefore to use $\mathcal{H}_n$ to warm-start our context-aware extension with kernel $\tilde{\mathcal{K}}_p$ while fixing $\beta = 1$. In other words, our approach would be used to refine the smart and computationally-efficient initialization provided by `hyperband`.

References

Bardenet, Rémi, Brendel, Mátyás, Kégl, Balázs, and Sebag, Michele. Collaborative hyperparameter tuning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 199–207, 2013.

Bengio, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1), 2009.

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

Bergstra, James, Bardenet, Rémi, Kégl, B, and Bengio, Y. Implementations of algorithms for hyper-parameter optimization. In *NIPS Workshop on Bayesian optimization*, pp. 29, 2011.

Chang, Chih-Chung and Lin, Chih-Jen. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Domhan, Tobias, Springenberg, Tobias, and Hutter, Frank. Extrapolating learning curves of deep neural networks. In *ICML 2014 AutoML Workshop*, 2014.

Duvenaud, David K, Nickisch, Hannes, and Rasmussen, Carl E. Additive gaussian processes. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 24*, pp. 226–234. Curran Associates, Inc., 2011.

Eggensperger, Katharina, Feurer, Matthias, Hutter, Frank, Bergstra, James, Snoek, Jasper, Hoos, H, and Leyton-Brown, Kevin. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.

Feurer, Matthias, Springenberg, T, and Hutter, Frank. Initializing Bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

GPyOpt. GPyOpt: A Bayesian Optimization framework in Python. `http://github.com/SheffieldML/GPyOpt`, 2016.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION-5*, pp. 507?523, 2011.

Hutter, Frank and Osborne, Michael A. A kernel for hierarchical parameter spaces. Technical report, preprint arXiv:1310.5738, 2013.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. Technical report, preprint arXiv:1412.6980, 2014.

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

Kushner, Harold J. 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.

Li, Lisha, Jamieson, Kevin, DeSalvo, Giulia, Rostamizadeh, Afshin, and Talwalkar, Ameet. Hyperband: A novel bandit-based approach to hyperparameter optimization. Technical report, preprint arXiv:1603.06560, 2016.

Lu, Zhiyun, May, Avner, Liu, Kuan, Garakani, Alireza Bagheri, Guo, Dong, Bellet, Aurélien, Fan, Linxi, Collins, Michael, Kingsbury, Brian, Picheny, Michael, et al. How to scale up kernel methods to be as good as deep neural nets. Technical report, preprint arXiv:1411.4000, 2014.

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

Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer, 2006.

Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

Rahimi, Ali, Recht, Benjamin, et al. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, volume 3, pp. 5, 2007.

Rasmussen, Carl and Williams, Chris. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

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

Shawe-Taylor, J. and Cristianini, N. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical Bayesian optimization of machine learning algorithms.

In *Advances in Neural Information Processing Systems*, pp. 2960–2968, 2012.

Sobol, Ilya M. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4): 86–112, 1967.

Srinivas, N., Krause, A., Kakade, S. M., and M., Seeger. Gaussian Process optimization in the bandit setting: No regret and experimental design. In Furnkranz, J. and Joachims, T. (eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML)*, Haifa, June 2010.

Swersky, Kevin, Snoek, Jasper, and Adams, Ryan P. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems*, pp. 2004–2012, 2013.

Swersky, Kevin, Duvenaud, David, Snoek, Jasper, Hutter, Frank, and Osborne, Michael A. Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. Technical report, preprint arXiv:1409.4011, 2014a.

Swersky, Kevin, Snoek, Jasper, and Adams, Ryan Prescott. Freeze-thaw Bayesian optimization. Technical report, preprint arXiv:1406.3896, 2014b.

Thompson, William R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, pp. 285–294, 1933.

Zhang, Yuyu, Bahadori, Mohammad Taha, Su, Hang, and Sun, Jimeng. Flash: Fast bayesian optimization for data analytic pipelines. In Krishnapuram, Balaji, Shah, Mohak, Smola, Alexander J., Aggarwal, Charu, Shen, Dou, and Rastogi, Rajeev (eds.), *KDD*, pp. 2065–2074. ACM, 2016. ISBN 978-1-4503-4232-2.