Batched Large-scale Bayesian Optimization in High-dimensional Spaces

Zi Wang
MIT CSAIL

Clement Gehring
MIT CSAIL

Pushmeet Kohli
DeepMind

Stefanie Jegelka
MIT CSAIL

Abstract

Bayesian optimization (BO) has become an effective approach for black-box function optimization problems when function evaluations are expensive and the optimum can be achieved within a relatively small number of queries. However, many cases, such as the ones with high-dimensional inputs, may require a much larger number of observations for optimization. Despite an abundance of observations thanks to parallel experiments, current BO techniques have been limited to merely a few thousand observations. In this paper, we propose *ensemble Bayesian optimization* (EBO) to address three current challenges in BO *simultaneously*: (1) large-scale observations; (2) high dimensional input spaces; and (3) selections of batch queries that balance quality and diversity. The key idea of EBO is to operate on an ensemble of additive Gaussian process models, each of which possesses a randomized strategy to divide and conquer. We show unprecedented, previously impossible results of scaling up BO to tens of thousands of observations within minutes of computation.

1 Introduction

Global optimization of black-box and non-convex functions is an important component of modern machine learning. From optimizing hyperparameters in deep models to solving inverse problems encountered in computer vision and policy search for reinforcement learning, these optimization problems have many important applications in machine learning and its allied disciplines. In the past decade, Bayesian optimization has become a popular approach for global optimization of non-convex functions that are expensive to evaluate. Recent work addresses better query strategies (Kushner, 1964; Močkus, 1974; Srinivas et al., 2012; Hennig and Schuler, 2012; Hernández-Lobato et al., 2014; Wang et al., 2016a; Kawaguchi et al., 2015), techniques for batch queries (Desautels et al., 2014; González et al., 2016), and algorithms for high dimensional problems (Wang et al., 2016b; Kandasamy et al., 2015).

Despite the above-mentioned successes, Bayesian optimization remains somewhat impractical, since it is typically coupled with expensive function estimators (Gaussian processes) and non-convex acquisition functions that are hard to optimize in high dimensions and sometimes expensive to evaluate. To alleviate these difficulties, recent work explored the use of random feature approximations (Snoek et al., 2015; Lakshminarayanan et al., 2016) and sparse Gaussian processes (McIntire et al., 2016), but, while improving scalability, these methods still suffer from misestimation of confidence bounds (an essential part of the acquisition functions), and expensive or inaccurate Gaussian process (GP) hyperparameter inference. Indeed, to the best of our knowledge, Bayesian optimization is typically limited to a few thousand evaluations (Lakshminarayanan et al., 2016). Yet, reliable search and estimation for complex functions in very high-dimensional spaces may well require more evaluations. With the increasing availability of parallel computing resources, large number of function evaluations are possible if the underlying approach can leverage the parallelism. Comparing to the millions of evaluations possible (and needed) with *local* methods like stochastic gradient descent, the scalability of *global* Bayesian optimization leaves large room for desirable progress. In particular, the lack of scalable uncertainty estimates to guide the search is a major roadblock for huge-scale Bayesian optimization.

In this paper, we propose ensemble Bayesian optimization (EBO), a global optimization method targeted to high dimensional, large scale parameter search problems whose queries are parallelizable. Such problems are abundant in hyper and control parameter optimization in machine learning and robotics (Calandra, 2017; Snoek et al., 2012). EBO relies on two main ideas that are implemented at multiple levels: (1) we use efficient partition-based function approximators (across both data *and* features) that simplify and accelerate search and optimization; (2) we enhance the expressive power of these approximators by using ensembles and a stochastic approach. We maintain an evolving (posterior)

Proceedings of the 21^{st} International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84. Copyright 2018 by the author(s).

distribution over the (infinite) ensemble and, in each iteration, draw one member to perform search and estimation.

In particular, we use a new combination of three types of partition-based approximations: (1-2) For improved GP estimation, we propose a novel *hierarchical* additive GP model based on tile coding (a.k.a. random binning or Mondrian forest features). We learn a posterior distribution over kernel width and the additive structure; here, Gibbs sampling prevents overfitting. (3) To accelerate the sampler, which depends on the likelihood of the observations, we use an efficient, randomized block approximation of the Gram matrix based on a Mondrian process. Sampling and query selection can then be parallelized across blocks, further accelerating the algorithm.

As a whole, this combination of simple, tractable structure with ensemble learning and randomization improves efficiency, uncertainty estimates and optimization. Moreover, we show that our realization of these ideas offers an alternative explanation for global optimization heuristics that have been popular in other communities, indicating possible directions for further theoretical analysis. Our empirical results demonstrate that EBO can speed up the posterior inference by 2-3 orders of magnitude (400 times in one experiment) compared to the state-of-the-art, without sacrificing quality. Furthermore, we demonstrate the ability of EBO to handle sample-intensive hard optimization problems by applying it to real-world problems with tens of thousands of observations.

Related Work There has been a series of works addressing the three big challenges in BO: selecting batch evaluations (Contal et al., 2013; Desautels et al., 2014; González et al., 2016; Wang et al., 2017; Daxberger and Low, 2017), high-dimensional input spaces (Wang et al., 2016b; Djolonga et al., 2013; Li et al., 2016; Kandasamy et al., 2015; Wang et al., 2017; Wang and Jegelka, 2017), and scalability (Snoek et al., 2015; Lakshminarayanan et al., 2016; McIntire et al., 2016). Although these three problems tend to co-occur, this paper is the first (to the best of our knowledge) to address all three challenges jointly in one framework.

Most closely related to parts of this paper is (Wang et al., 2017), but our algorithm significantly improves on that work in terms of scalability (see Sec. 4.1 for an empirical comparison), and has fundamental technical differences. First, the Gibbs sampler by Wang et al. (2017) only learns the additive structure but not the kernel parameters, while our sampler jointly learns both of them. Second, our proposed algorithm partitions the input space for scalability and parallel inference. We achieve this by a Mondrian forest. Third, as a result, our method automatically generates batch queries, while the other work needs an explicit batch strategy.

Other parts of our framework are inspired by the Mondrian forest (Lakshminarayanan et al., 2016), which partitions the input space via a Mondrian tree and aggregates trees into a forest. The closely related Mondrian kernels (Balog et al., 2016) use random features derived from Mondrian forests to construct a kernel. Such a kernel, in fact, approximates a Laplace kernel. In fact, Mondrian forest features can be considered a special case of the popular tile coding features widely used in reinforcement learning (Sutton and Barto, 1998; Albus et al., 1975). Lakshminarayanan et al. (2016) showed that, in low-dimensional settings, Mondrian forest kernels scale better than the regular GP and achieve good uncertainty estimates in many low-dimensional problems.

Besides Mondrian forests, there is a rich literature on sparse GP methods to address the scalability of GP regression (Seeger et al., 2003; Snelson and Ghahramani, 2006; Titsias, 2009; Hensman et al., 2013). However, these methods are mostly only shown to be useful when the input dimension is low and there exist redundant data points, so that inducing points can be selected to emulate the original posterior GP well. However, data redundancy is usually not the case in high-dimensional Bayesian optimization. Recent applications of sparse GPs in BO (McIntire et al., 2016) only consider experiments with less than 80 function evaluations in BO and do not show results on large scale observations. Another approach to tackle large scale GPs distributes the computation via local experts (Deisenroth and Ng, 2015). However, this is not very suitable for the acquisition function optimization needed in Bayesian optimization, since every valid prediction needs to synchronize the predictions from all the local experts. Our paper is also related to Gramacy and Lee (2008). While Gramacy and Lee (2008) focuses on modeling non-stationary functions with treed partitions, our work integrates tree structures and Bayesian optimization in a novel way.

2 Background and Challenges

Consider a simple but high-dimensional search space $\mathcal{X} = [0, R]^D \subseteq \mathbb{R}^D$. We aim to find a maximizer $x^* \in \arg\max_{x\in\mathcal{X}} f(x)$ of a black-box function $f : \mathcal{X} \to \mathbb{R}$.

Gaussian processes. Gaussian processes (GPs) are popular priors for modeling the function f in Bayesian optimization. They define distributions over functions where any finite set of function values has a multivariate Gaussian distribution. A Gaussian process $\mathcal{GP}(\mu, \kappa)$ is fully specified by a mean function $\mu(\cdot)$ and covariance (kernel) function $\kappa(\cdot, \cdot)$. Let f be a function sampled from $\mathcal{GP}(0, \kappa)$. Given observations $\mathcal{D}_n = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^n$ where $y_t \sim \mathcal{N}(f(\boldsymbol{x}_t), \sigma)$, we obtain the posterior mean and variance of the function as

$$\mu_n(\boldsymbol{x}) = \boldsymbol{\kappa}_n(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{K}_n + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}_n, \tag{2.1}$$

$$\sigma_n^2(\boldsymbol{x}) = \kappa(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{\kappa}_n(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{K}_n + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{\kappa}_n(\boldsymbol{x}) \tag{2.2}$$

via the kernel matrix $\boldsymbol{K}_n = [\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{\boldsymbol{x}_i,\boldsymbol{x}_j\in\mathcal{D}_n}$ and $\boldsymbol{\kappa}_n(\boldsymbol{x}) = [\kappa(\boldsymbol{x}_i, \boldsymbol{x})]_{\boldsymbol{x}_i\in D_n}$ (Rasmussen and Williams, 2006). The log data likelihood for $\mathcal{D}_n$ is given by

$$\log p(\mathcal{D}_n) = -\frac{1}{2}\boldsymbol{y}_n^{\mathrm{T}}(\boldsymbol{K}_n + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}_n - \frac{1}{2}\log|\boldsymbol{K}_n + \sigma^2\boldsymbol{I}| - \frac{n}{2}\log 2\pi. \tag{2.3}$$

While GPs provide flexible, broadly applicable function estimators, the $O(n^3)$ computation of the inverse $(\boldsymbol{K}_n + \sigma^2\boldsymbol{I})^{-1}$ and determinant $|\boldsymbol{K}_n + \sigma^2\boldsymbol{I}|$ can become major bottlenecks as n grows, for both posterior function value predictions and data likelihood estimation.

Additive structure. To reduce the complexity of the vanilla GP, we assume a latent decomposition of the input dimensions $[D] = \{1, \dots, D\}$ into disjoint subspaces, namely, $\bigcup_{m=1}^{M} A_m = [D]$ and $A_i \cap A_j = \emptyset$ for all $i \neq j$, $i, j \in [M]$. As a result, the function f decomposes as $f(x) = \sum_{m\in[M]} f_m(x^{A_m})$ (Kandasamy et al., 2015). If each component f_m is drawn independently from $\mathcal{GP}(\mu^{(m)}, \kappa^{(m)})$ for all $m \in [M]$, the resulting f will also be a sample from a GP: $f \sim \mathcal{GP}(\mu, \kappa)$, with $\mu(x) = \sum_{m\in[M]} \mu_m(x^{A_m})$, $\kappa(x, x') = \sum_{m\in[M]} \kappa^{(m)}(x^{A_m}, x'^{A_m})$.

The additive structure reduces sample complexity and helps BO to search more efficiently and effectively since the acquisition function can be optimized component-wise. But it remains challenging to learn a good decomposition structure $\{A_m\}$. Recently, Wang et al. (2017) proposed learning via Gibbs sampling. This sampler takes hours for merely a few hundred points, because it needs a vast number of expensive data likelihood computations.

Random features. It is possible use random features (Rahimi et al., 2007) to approximate the GP kernel and alleviate the $O(n^3)$ computation in Eq. (2.1) and Eq. (2.3). Let $\boldsymbol{\phi} : \mathcal{X} \mapsto \mathbb{R}^{D_R}$ be the (scaled) random feature operator and $\boldsymbol{\Phi}_n = [\boldsymbol{\phi}(\boldsymbol{x}_1), \cdots, \boldsymbol{\phi}(\boldsymbol{x}_n)]^{\mathrm{T}} \in \mathbb{R}^{n \times D_R}$. The GP posterior mean and variance can be written as

$$\mu_n(\boldsymbol{x}) = \sigma^{-2}\boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\Sigma}_n\boldsymbol{\Phi}_n^{\mathrm{T}}\boldsymbol{y}_n, \tag{2.4}$$

$$\sigma_n^2(x) = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\Sigma}_n\boldsymbol{\phi}(\boldsymbol{x}), \tag{2.5}$$

where $\boldsymbol{\Sigma}_n = (\boldsymbol{\Phi}_n^{\mathrm{T}}\boldsymbol{\Phi}_n\sigma^{-2} + \boldsymbol{I})^{-1}$. By the Woodbury matrix identity and the matrix determinant lemma, the log data likelihood becomes

$$\log p(\mathcal{D}_n) = \frac{\sigma^{-4}}{2}\boldsymbol{y}_n^{\mathrm{T}}\boldsymbol{\Phi}_n\boldsymbol{\Sigma}_n\boldsymbol{\Phi}_n^{\mathrm{T}}\boldsymbol{y}_n - \frac{1}{2}\log|\boldsymbol{\Sigma}_n^{-1}| - \frac{\sigma^{-2}}{2}\boldsymbol{y}_n^{\mathrm{T}}\boldsymbol{y}_n - \frac{n}{2}\log 2\pi\sigma^2. \tag{2.6}$$

The number of random features necessary to approximate the GP well in general increases with the number of observations (Rudi et al., 2017). Hence, for large-scale observations, we cannot expect to solely use a fixed number of features. Moreover, learning hyperparameters for random features is expensive: for Fourier features, the computation of Eq. (2.6) means re-computing the features, plus $O(D_R^3)$ for the inverse and determinant. With Mondrian features (Lakshminarayanan et al., 2016), we can learn the kernel width efficiently by adding more Mondrian blocks, but this procedure is not well compatible with learning additive structure, since the whole structure of the sampled Mondrian features will change. In addition, we typically need a forest of trees for a good approximation.

Tile coding. Tile coding (Sutton and Barto, 1998; Albus et al., 1975) is a k-hot encoding widely used in reinforcement learning as an efficient set of non-linear features. In its simplest form, tile coding is defined by k partitions, referred to as layers. An encoded data point becomes a binary vector with a non-zero entry for each bin containing the data point. There exists methods for sampling random partitions that allow to approximate various kernels, such as the 'hat' kernel (Rahimi et al., 2007), making tile coding well suited for our purposes.

Variance starvation. It is probably not surprising that using finite random features to learn the function *distribution* will result in a loss in accuracy (Forster, 2005). For example, we observed that, while the mean predictions are preserved reasonably well around regions where we have observations, both mean and confidence bound predictions can become very bad in regions where we do not have observations, once there are more observations than features. We refer to this underestimation of variance scale compared to mean scale, illustrated in Fig. 1, as *variance starvation*.

3 Ensemble Bayesian Optimization

Next, we describe an approach that scales Bayesian Optimization when parallel computing resources are available. We name our approach, outlined in Alg.1, *Ensemble Bayesian optimization (EBO)*. At a high level, EBO uses a (stochastic) series of Mondrian trees to partition the input space, learn the kernel parameters of a GP locally, and aggregate these parameters. Our forest hence spans across BO iterations.

In the t-th iteration of EBO in Alg. 1, we use a Mondrian process to randomly partition the search space into J parts (line 4), where J can be dependent on the size of the observations $\mathcal{D}_{t-1}$. For the j-th partition, we have a subset $\mathcal{D}_{t-1}^j$ of observations. From those observations, we learn a local GP with random tile coding *and* additive structure, via Gibbs sampling (line 6). For conciseness, we refer to such GPs as TileGPs. The probabilistic tile coding can be replaced by a Mondrian grid that approximates a Laplace kernel (Balog and Teh, 2015). Once a TileGP is learned locally, we can run BO with the acquisition function η in each partition to generate a candidate set of points, and, from

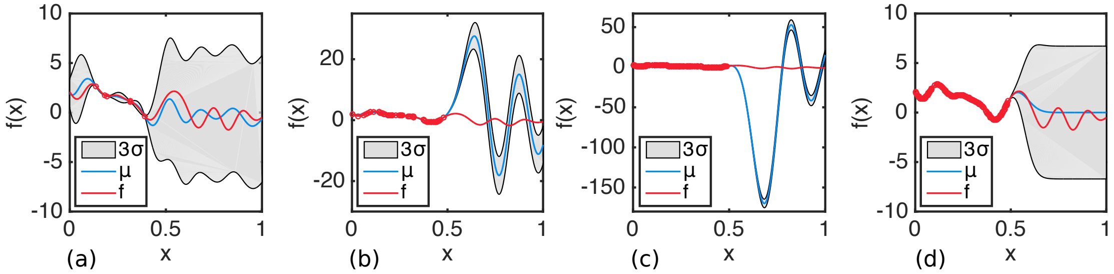

Figure 1: We use 1000 Fourier features to approximate a 1D GP with a squared exponential kernel. The observations are samples from a function f (red line) drawn from the GP with zero mean in the range $[-10, 0.5]$. (a) Given 100 sampled observations (red circles), the Fourier features lead to reasonable confidence bounds. (b) Given 1000 sampled observations (red circles), the quality of the variance estimates degrades. (c) With additional samples (5000 observations), the problem is exacerbated. The scale of the variance predictions relative to the mean prediction is very small. (d) For comparison, the proper predictions of the original full GP conditioned on the same 5000 observations as (c). Variance starvation becomes a serious problem for random features when the size of data is close to or larger than the size of the features.

those, select a batch that is both informative (high-quality) and diverse (line 14).

Algorithm 1 Ensemble Bayesian Optimization (EBO)

```
1:  function EBO ($f, \mathcal{D}_0$)
2:      Initialize $z, k$
3:      for $t = 1, \cdots, T$ do
4:          $\{\mathcal{X}_j\}_{j=1}^J \leftarrow$ MONDRIAN($[0, R]^D, z, k, J$)
5:          parfor $j = 1, \cdots, J$ do
6:              $z^j, k^j \leftarrow$ GIBBSSAMPLING($z, k \mid \mathcal{D}_{t-1}^j$)
7:              $\eta_{t-1}^j(\cdot) \leftarrow$ ACQUISITION ($\mathcal{D}_{t-1}^j, z^j, k^j$)
8:              $\{A_m\}_{m=1}^M \leftarrow$ DECOMPOSITION($z^j$)
9:              for $m = 1, \cdots, M$ do
10:                 $\boldsymbol{x}_{tj}^{A_m} \leftarrow \arg\max_{\boldsymbol{x} \in \mathcal{X}_j^{A_m}} \eta_{t-1}^j(\boldsymbol{x})$
11:             end for
12:         end parfor
13:         $z \leftarrow$ SYNC($\{z^j\}_{j=1}^J$),  $k \leftarrow$ SYNC($\{k^j\}_{j=1}^J$)
14:         $\{\boldsymbol{x}_{tb}\}_{b=1}^B \leftarrow$ FILTER ($\{\boldsymbol{x}_{tj}\}_{j=1}^J \mid z, k$)
15:         parfor $b = 1, \cdots, B$ do
16:             $y_{tb} \leftarrow f(\boldsymbol{x}_{tb})$
17:         end parfor
18:         $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{\boldsymbol{x}_{tb}, y_{tb}\}_{b=1}^B$
19:     end for
20: end function
```

Since, in each iteration, we draw an input space partition and update the kernel width and the additive structure, the algorithm may be viewed as implicitly and stochastically running BO on an ensemble of GP models. In the following, we describe our model and the procedures of Alg. 1 in detail. In the Appendix, we show an illustration how EBO optimizes a 2D function.

3.1 Partitioning the input space via a Mondrian process

When faced with a "big" problem, a natural idea is to divide and conquer. For large scale Bayesian optimization, the question is how to divide without losing the valuable local information that gives good uncertainty measures. In EBO, we use a Mondrian process to divide the input space and the observed data, so that nearby data points remain together in one partition, preserving locality[1]. The Mondrian process uses axis-aligned cuts to divide the input space $[0, R]^D$ into a set of partitions $\{\mathcal{X}_j\}_{j=0}^J$ where $\cup_j \mathcal{X}_j = [0, R]^D$ and $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset, \forall i \neq j$. Each partition $\mathcal{X}_j$ can be conveniently described by a hyperrectangle $[l_1^j, h_1^j] \times \cdots \times [l_D^j, h_D^j]$, which facilitates the efficient use of tile coding and Mondrian grids in a TileGP. In the next section, we define a TileGP and introduce how its parameters are learned.

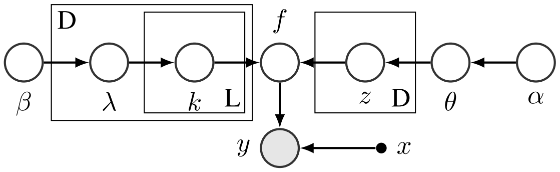

Figure 2: The graphical model for TileGP, a GP with additive and tile kernel partitioning structure. The parameter λ controls the rate for the number of cuts k of the tilings (inverse of the kernel bandwidth); the parameter z controls the additive decomposition of the input feature space.

3.2 Learning a local TileGP via Gibbs sampling

For the j-th hyperrectangle partition $\mathcal{X}_j = [l_1^j, h_1^j] \times \cdots \times [l_D^j, h_D^j]$, we use a TileGP to model the function f locally. We use the acronym "TileGP" to denote the Gaussian process model that uses additive kernels, with each component represented by tilings. We show the details of the generative model for TileGP in Alg. 2 and the graphical model in Fig. 3.2 with fixed hyper-parameters α, β_0, β_1. The main difference to the additive GP model used in (Wang et al.,

[1] We include the algorithm for input space partitioning in the appendix.

2017) is that TileGP constructs a hierarchical model for the random features (and hence, the kernels), while Wang et al. (2017) do not consider the kernel parameters to be part of the generative model. The random features are based on tile coding or Mondrian grids, with the number of cuts generated by D Poisson processes on $[l_d^j, h_d^j]$ for each dimension $d = 1, \cdots, D$. On the i-th layer of the tilings, tile coding samples the offset δ from a uniform distribution $U[0, \frac{h_d^j - l_d^j}{k_{di}}]$ and places the cuts uniformly starting at $\delta + l_d^j$. The Mondrian grid samples k_{di} cut locations uniformly randomly from $[l_d^j, h_d^j]$. Because of the data partition, we always have more features than observations, which can alleviate the variance starvation problem described in Section 2.

We can use Gibbs sampling to efficiently learn the cut parameter k and decomposition parameter z by marginalizing out λ and θ. Notice that both k and z take discrete values; hence, unlike other continuous GP parameterizations, we only need to sample discrete variables for Gibbs sampling.

Algorithm 2 Generative model for TileGP

```
Draw mixing proportions $\theta \sim \textsc{Dir}(\alpha)$
for $d = 1, \cdots, D$ do
    Draw additive decomposition $z_d \sim \textsc{Multi}(\theta)$
    Draw Poisson rate parameter $\lambda_d \sim \textsc{Gamma}(\beta_0, \beta_1)$
    for $i = 1, \cdots, L$ do
        Draw number of cuts $k_{di} \sim \textsc{Poisson}(\lambda_d(h_d^j - l_d^j))$
        { Draw offset $\delta \sim U[0, \frac{h_d^j - l_d^j}{k_{di}}]$        Tile Coding
        { Draw cut locations $\boldsymbol{b} \sim U[l_d^j, h_d^j]$    Mondrian Grids
    end for
end for
Construct the feature projection $\boldsymbol{\phi}$ and the kernel $\kappa = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\phi}$
from $z$ and sampled tiles
Draw function $f \sim \mathcal{GP}(0, \kappa)$
Given input $\boldsymbol{x}$, draw function value $y \sim \mathcal{N}(f(\boldsymbol{x}), \sigma)$
```

Given the observations $\mathcal{D}_{t-1}$ in the j-th hyperrectangle partition, the posterior distribution of the (local) parameters λ, k, z, θ is

$$\begin{aligned} &p(\lambda, k, z, \theta \mid \mathcal{D}_{t-1}; \alpha, \beta) \\ &\quad \propto p(\mathcal{D}_{t-1} \mid z, k) p(z \mid \theta) p(k \mid \lambda) p(\theta; \alpha) p(\lambda; \beta). \end{aligned}$$

Marginalizing over the Poisson rate parameter λ and the mixing proportion θ gives

$$\begin{aligned} &p(k, z \mid \mathcal{D}_{t-1}; \alpha, \beta) \\ &\propto p(\mathcal{D}_{t-1}|z, k) \int p(z|\theta) p(\theta; \alpha) \, \mathrm{d}\theta \int p(k|\lambda) p(\lambda; \beta) \, \mathrm{d}\lambda \\ &\propto p(\mathcal{D}_{t-1} \mid z, k) \prod_m \frac{\Gamma(|A_m| + \alpha_m)}{\Gamma(\alpha_m)} \\ &\qquad \times \prod_d \frac{\Gamma(\beta_1 + |k_d|)}{(\prod_{i=1}^L k_{di}!)(\beta_0 + L)^{\beta_1 + |k_d|}} \end{aligned}$$

where $|k_d| = \sum_{i=1}^L k_{di}$. Hence, we only need to sample k and z when learning the hyperparameters of the TileGP kernel. For each dimension d, we sample the group assignment z_d according to

$$\begin{aligned} &p(z_d = m \mid \mathcal{D}_{t-1}, k, z_{\neg d}; \alpha) \propto p(\mathcal{D}_{t-1} \mid z, k) p(z_d \mid z_{\neg d}) \\ &\quad \propto p(\mathcal{D}_{t-1} \mid z, k)(|A_m| + \alpha_m). \end{aligned} \tag{3.1}$$

We sample the number of cuts k_{di} for each dimension d and each layer i from the posterior

$$\begin{aligned} &p(k_{di} \mid \mathcal{D}_{t-1}, k_{\neg di}, z; \beta) \propto p(\mathcal{D}_{t-1} \mid z, k) p(k_{di} \mid k_{\neg di}) \\ &\quad \propto \frac{p(\mathcal{D}_n \mid z, k)\Gamma(\beta_1 + |k_d|)}{(\beta_0 + L)^{k_{di}} k_{di}!}. \end{aligned} \tag{3.2}$$

If distributed computing is available, each hyperrectangle partition of the input space is assigned a worker to manage all the computations within this partition. On each worker, we use the above Gibbs sampling method to learn the additive structure and kernel bandwidth jointly. Conditioned on the observations associated with the partition on the worker, we use the learned posterior TileGP to select the most promising input point in this partition, and eventually send this candidate input point back to the main process together with the learned decomposition parameter z and the cut parameter k. In the next section, we introduce the acquisition function we used in each worker and how to filter the recommended candidates from all the partitions.

3.3 Acquisition functions and filtering

In this paper, we mainly focus on parameter search problems where the objective function is designed by an expert and the global optimum or an upper bound on the function is known. While any BO acquisition functions can be used within the EBO framework, we use an acquisition function from (Wang and Jegelka, 2017) to exploit the knowledge of the upper bound. Let f^* be such an upper bound, i.e., $\forall x \in \mathcal{X}, f^* \geq f(x)$. Given the observations $\mathcal{D}_{t-1}^j$ associated with the j-th partition of the input space, we minimize the acquisition function $\eta_{t-1}^j(x) = \frac{f^* - \mu_{t-1}^j(x)}{\sigma_{t-1}^j(x)}$. Since the kernel is additive, we can optimize $\eta_{t-1}^j(\cdot)$ separately for each additive component. Namely, for the m-th component of the additive structure, we optimize $\eta_{t-1}^j(\cdot)$ only on the active dimensions A_m. This resembles a block coordinate descent, and greatly facilitates the optimization of the acquisition function.

Filtering. Once we have a proposed uery point from each partition, we select B of them according to the scoring function $\xi(X) = \log\det K_X - \sum_{b=1}^B \eta(\boldsymbol{x}_b)$ where $X = \{\boldsymbol{x}_b\}_{b=1}^B$. We use the log determinant term to force diversity and η to maintain quality. We maximize this function greedily. In some cases, the number of partitions J can be smaller than the batch size B. In this case, one may either use just J candidates, or use batch BO on each partition. We use the latter, and discuss details in the appendix.

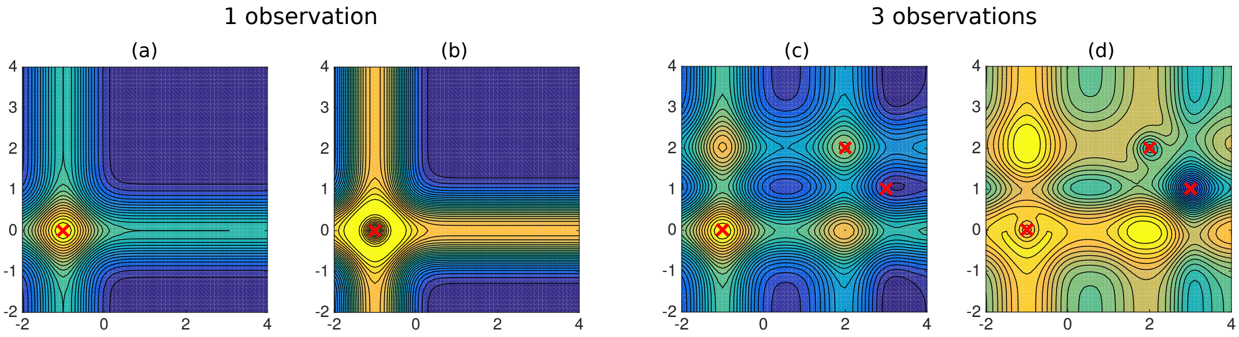

Figure 3: Posterior mean function (a, c) and GP-UCB acquisition function (b, d) for an additive GP in 2D. The maxima of the posterior mean and acquisition function are at the points resulting from an exchange of coordinates between "good" observed points (-1,0) and (2,2).

3.4 Efficient data likelihood computation and parameter synchronization

For the random features, we use tile coding due to its sparsity and efficiency. Since non-zero features can be found and computed by binning, the computational cost for encoding a data point scales linearly with dimensions and number of layers. The resulting representation is sparse and convenient to use. Additionally, the number of non-zero features is quite small, which allows us to efficiently compute a sparse Cholesky decomposition of the inner product (Gram matrix) or the outer product of the data. This allows us to efficiently compute the data likelihoods.

In each iteration t, after the batch workers return the learned decomposition indicator z^b and the number of tiles $k^b, b \in [B]$, we synchronize these two parameters (line 13 of Alg. 1). For the number of tiles k, we set k_d to be the rounded mean of $\{k_d^b\}_{b=1}^B$ for each dimension $d \in [D]$. For the decomposition indicator, we use correlation clustering to cluster the input dimensions.

3.5 Relations to Mondrian kernels, random binning and additive Laplace kernels

Our model described in Section 3.2 can use tile coding and Mondrian grids to construct the kernel. Tile coding and Mondrian grids are also closely related to Mondrian Features and Random Binning: All of the four kinds of random features attempt to find a sparse random feature representation for the raw input $\boldsymbol{x}$ based on the partition of the space with the help of layers. We illustrate the differences between one layer of the features constructed by tile coding, Mondrian grids, Mondrian features and random binning in the appendix. Mondrian grids, Mondrian features and random binning all converge to the Laplace kernel as the number of layers L goes to infinity. The tile coding kernel, however, does not approximate a Laplace kernel. Our model with Mondrian grids approximates an additive Laplace kernel:

Lemma 3.1. *Let the random variable* $k_{di} \sim \textsc{Poisson}(\lambda_d R)$ *be the number of cuts in the Mondrian grids of TileGP for dimension* $d \in [D]$ *and layer* $i \in [L]$*. The TileGP kernel* κ_L *satisfies* $\lim_{L\to\infty} \kappa_L(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{M}\sum_{m=1}^{M} e^{\lambda_d R |\boldsymbol{x}^{A_m} - \boldsymbol{x}'^{A_m}|}$*, where* $\{A_m\}_{m=1}^{M}$ *is the additive decomposition.*

We prove the lemma in the appendix. Balog et al. (2016) showed that in practice, the Mondrian kernel constructed from Mondrian features may perform slightly better than random binning in certain cases. Although it would be possible to use a Mondrian partition for each layer of tile coding, we only consider uniform, grid based binning with random offests because this allows the non-zero features to be computed more efficiently ($O(1)$ instead of $O(\log k)$). Note that as more dimensions are discretized in this manner, the number of features grows exponentially. However, the number of non-zero entries can be independently controlled, allowing to create sparse representations that remain computationally tractable.

3.6 Connections to evolutionary algorithms

Next, we make some observations that connect our randomized ensemble BO to ideas for global optimization heuristics that have successfully been used in other communities. In particular, these connections offer an explanation from a BO perspective and may aid further theoretical analysis.

Evolutionary algorithms (Back, 1996) maintain an ensemble of "good" candidate solutions (called chromosomes) and, from those, generate new query points via a number of operations. These methods too, implicitly, need to balance exploration with local search in areas known to have high function values. Hence, there are local operations (mutations) for generating new points, such as random perturbations or local descent methods, and global operations. While it is relatively straightforward to draw connections between those local operations and optimization methods used in machine learning, we here focus on global exploration.

A popular global operation is *crossover*: given two "good" points $x, y \in \mathbb{R}^D$, this operation outputs a new point z whose coordinates are a combination of the coordinates of x and y, i.e., $z_i \in \{x_i, y_i\}$ for all $i \in [D]$. In fact, this operation is analogous to BO with a (randomized) additive kernel: the crossover strategy implicity corresponds to the assumption that high function values can be achieved by combining coordinates from points with high function values. For comparison, consider an additive kernel $\kappa(x, x') = \sum_{m=1}^{M} \kappa^{(m)}(x^{A_m}, x'^{A_m})$ and $f(x) = \sum_{m=1}^{M} f^m(x^{A_m})$. Since each sub-kernel $\kappa^{(m)}$ is "blind" to the dimensions in the complement of A_m, any point x' that is close to an observed high-value point x in the dimensions A_m will receive a high value $f^m(x)$, independent of the other dimensions, and, as a result, looks like a "good" candidate.

We illustrate this reasoning with a 2D toy example. Figure 3 shows the posterior mean prediction and GP-UCB criterion $\hat{f}(x) + 0.1\sigma(x)$ for an additive kernel with $A_1 = \{1\}$, $A_2 = \{2\}$ and $\kappa^m(x_m, y_m) = \exp(-2(x_m - y_m)^2)$. High values of the observed points generalize along the dimensions "ignored" by the sub-kernels. After two good observations $(-1, 0)$ and $(2, 2)$, the "crossover" points $(-1, 2)$ and $(2, 0)$ are local maxima of GP-UCB and the posterior mean.

In real data, we do not know the best fitting underlying grouping structure of the coordinates. Hence, crossover does a random search over such partitions by performing random coordinate combinations, whereas our adaptive BO approach maintains a posterior distribution over partitions that adapts to the data.

4 Experiments

We empirically verify the scalability of EBO and its effectiveness of using random adaptive Mondrian partitions, and finally evaluate EBO on two real-world problems.[2]

4.1 Scalability of EBO

We compare EBO with a recent, state-of-the-art additive kernel learning algorithm, Structural Kernel Learning (SKL) (Wang et al., 2017). EBO can make use of parallel resources both for Gibbs sampling and BO query selections, while SKL can only parallelize query selections but not sampling. Because the kernel learning part is the computationally dominating factor of large scale BO, we compare the time each method needs to run 10 iterations of Gibbs sampling with 100 to 50000 observations in 20 dimensions. We show the timing results for the Gibbs samplers in Fig. 4(a), where EBO uses 240 cores via the Batch Service of Microsoft Azure. Due to a time limit we imposed, we did not finish SKL for more than 1500 observations. EBO runs more than 390 times faster than SKL when the observation size is 1500. Comparing the quality of learned parameter z for the additive structure, SKL has a Rand Index of 96.3% and EBO has a Rand Index of 96.8%, which are similar. In Fig. 4(b), we show speed-ups for different number of cores. EBO with 500 cores is not significantly faster than with 240 cores because EBO runs synchronized parallelization, whose runtime is decided by the slowest core. It is often the case that most of the cores have finished while the program is waiting for the slowest 1 or 2 cores to finish.

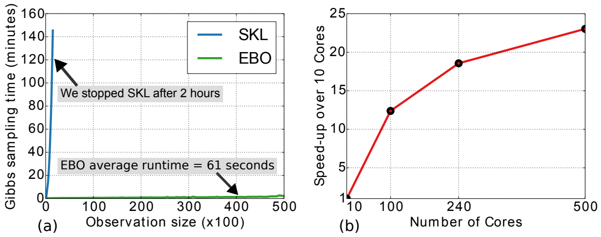

Figure 4: (a) Timing for the Gibbs sampler of EBO and SKL. EBO is significantly faster than SKL when the observation size N is relatively large. (b) Speed-up of EBO with 100, 240, 500 cores over EBO with 10 cores on 30,000 observations. Running EBO with 240 cores is almost 20 times faster than with 10 cores.

4.2 Effectiveness of EBO

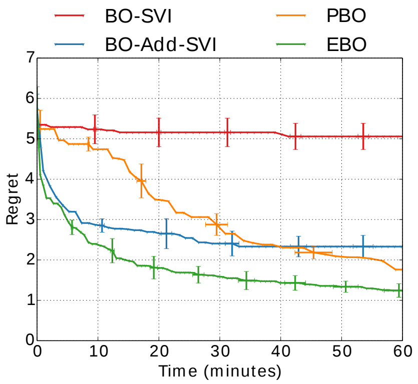

Figure 5: Averaged results of the regret of BO-SVI, BO-Add-SVI, PBO and EBO on 4 different functions drawn from a 50D GP with an additive Laplace kernel. BO-SVI has the highest regret for all functions. Using an additive GP within SVI (BO-Add-SVI) significantly improves over the full kernel. In general, EBO finds a good point much faster than the other methods.

Optimizing synthetic functions We verify the effectiveness of using ensemble models for BO on 4 functions randomly sampled from a 50-dimensional GP with an additive Laplace kernel. The hyperparameter of the Laplace kernel is known. In each iteration, each algorithm evaluates a batch of parameters of size B in parallel. We denote $\tilde{r}_t = \max_{x \in \mathcal{X}} f(x) - \max_{b \in [B]} f(x_{t,b})$ as the im-

[2] Our code is publicly available at `https://github.com/zi-w/Ensemble-Bayesian-Optimization`.

mediate regret obtained by the batch at iteration t, and $r_T = \min_{t \leq T} \tilde{r}_t$ as the regret, which captures the minimum gap between the best point found and the global optimum of the black-box function f.

We compare BO using SVI (Hensman et al., 2013) (BO-SVI), BO using SVI with an additive GP (BO-Add-SVI) and a distributed version of BO with a fixed partition (PBO) against EBO with a randomly sampled partition in each iteration. PBO has the same 1000 Mondrian partitions in all the iterations while EBO can have at most 1000 Mondrian partitions. BO-SVI uses a Laplace isotropic kernel without any additive structure, while BO-Add-SVI, PBO, EBO all use the known prior. More detailed experimental settings can be found in the appendix. Our experimental results in Fig. 5 shows that EBO is able to find a good point much faster than BO-SVI and BO-Add-SVI; and, randomization and the ensemble of partitions matters: EBO is much better than PBO.

Optimizing control parameters for robot pushing We follow Wang et al. (2017) and test our approach, EBO, on a 14 dimensional control parameter tuning problem for robot pushing. We compare EBO, BO-SVI, BO-Add-SVI and CEM Szita and Lörincz (2006) with the same 10^4 random observations and repeat each experiment 10 times. We run all the methods for 200 iterations, where each iteration has a batch size of 100. We plot the median of the best rewards achieved by CEM and EBO at each iteration in Fig. 6. More details on the experimental setups and the reward function can be found in the appendix. Overall CEM and EBO performed comparably and much better than the sparse GP methods (BO-SVI and BO-Add-SVI). We noticed that among all the experiments, CEM achieved a maximum reward of 10.19 while EBO achieved 9.50. However, EBO behaved slightly better and more stable than CEM as reflected by the standard deviation on the rewards.

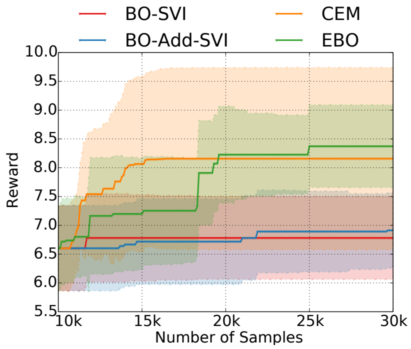

Figure 6: Comparing BO-SVI, BO-Add-SVI, CEM and EBO on a control parameter tuning task with 14 parameters.

Optimizing rover trajectories To further explore the performance of our method, we consider a trajectory optimization task in 2D, meant to emulate a rover navigation task. We describe a problem instance by defining a start position s and a goal position g as well as a cost function over the state space. Trajectories are described by a set of points on which a BSpline is to be fitted. By integrating the cost function over a given trajectory, we can compute the trajectory cost $c(\boldsymbol{x})$ of a given trajectory solution $\boldsymbol{x} \in [0,1]^{60}$. We define the reward of this problem to be $f(\boldsymbol{x}) = c(\boldsymbol{x}) + \lambda(\|\boldsymbol{x}_{0,1} - s\|_1 + \|\boldsymbol{x}_{59,60} - g\|_1) + b$. This reward function is non smooth, discontinuous, and concave over the first two and last two dimensions of the input. These 4 dimensions represent the start and goal position of the trajectory. The results in Fig. 7 showed that CEM was able to achieve better results than the BO methods on these functions, while EBO was still much better than the BO alternatives using SVI. More details can be found in the appendix.

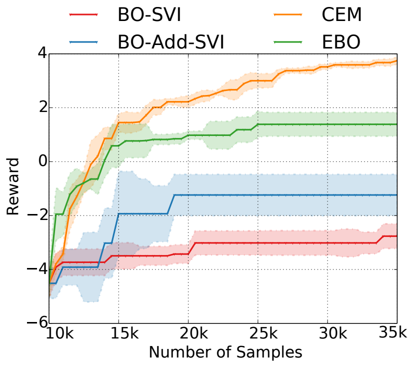

Figure 7: Comparing BO-SVI, BO-Add-SVI, CEM and EBO on a 60 dimensional trajectory optimization task.

5 Conclusion

Many black box function optimization problems are intrinsically high-dimensional and may require a huge number of observations in order to be optimized well. In this paper, we propose a novel framework, ensemble Bayesian optimization, to tackle the problem of scaling Bayesian optimization to both large numbers of observations and high dimensions. To achieve this, we propose a new framework that jointly integrates randomized partitions at various levels: our method is a stochastic method over a randomized, adaptive ensemble of partitions of the input data space; for each part, we use an ensemble of TileGPs, a new GP model we propose based on tile coding and additive structure. We also developed an efficient Gibbs sampling approach to learn the latent variables. Moreover, our method automatically generates batch queries. We empirically demonstrate the effectiveness and scalability of our method on high dimensional parameter search tasks with tens of thousands of observation data.

Acknowledgements

We gratefully acknowledge support from NSF CAREER award 1553284, NSF grants 1420927 and 1523767, from ONR grant N00014-14-1-0486, and from ARO grant W911NF1410433. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

James S Albus et al. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97(3):220–227, 1975.

Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

Matej Balog and Yee Whye Teh. The Mondrian process for machine learning. *arXiv preprint arXiv:1507.05181*, 2015.

Matej Balog, Balaji Lakshminarayanan, Zoubin Ghahramani, Daniel M Roy, and Yee Whye Teh. The Mondrian kernel. In *Uncertainty in Artificial Intelligence (UAI)*, 2016.

Roberto Calandra. *Bayesian Modeling for Optimization and Control in Robotics*. PhD thesis, Technische Universität, 2017.

Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013.

Erik A Daxberger and Bryan Kian Hsiang Low. Distributed batch Gaussian process optimization. In *International Conference on Machine Learning (ICML)*, 2017.

Marc Peter Deisenroth and Jun Wei Ng. Distributed Gaussian processes. *arXiv preprint arXiv:1502.02843*, 2015.

Thomas Desautels, Andreas Krause, and Joel W Burdick. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *Journal of Machine Learning Research*, 2014.

Josip Djolonga, Andreas Krause, and Volkan Cevher. High-dimensional Gaussian process bandits. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

Malcolm R Forster. Notice: No free lunches for anyone, bayesians included. *Department of Philosophy, University of Wisconsin–Madison Madison, USA*, 2005.

Javier González, Zhenwen Dai, Philipp Hennig, and Neil D Lawrence. Batch Bayesian optimization via local penalization. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

Robert B Gramacy and Herbert K H Lee. Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483):1119–1130, 2008.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13:1809–1837, 2012.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence (UAI)*, 2013.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Kirthevasan Kandasamy, Jeff Schneider, and Barnabas Poczos. High dimensional Bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning (ICML)*, 2015.

Kenji Kawaguchi, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Bayesian optimization with exponential convergence. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

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

Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. Mondrian forests for large-scale regression when uncertainty matters. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

Chun-Liang Li, Kirthevasan Kandasamy, Barnabás Póczos, and Jeff Schneider. High dimensional Bayesian optimization via restricted projection pursuit models. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

Mitchell McIntire, Daniel Ratner, and Stefano Ermon. Sparse Gaussian processes for Bayesian optimization. In *Uncertainty in Artificial Intelligence (UAI)*, 2016.

J. Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, 1974.

Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

Carl Edward Rasmussen and Christopher KI Williams. *Gaussian Processes for Machine Learning*. 2006.

Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Matthias Seeger, Christopher Williams, and Neil Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Artificial Intelligence and Statistics 9*, 2003.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, 2015.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias W Seeger. Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 2012.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

István Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Learning*, 18(12), 2006.

Michalis K Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning (ICML)*, 2017.

Zi Wang, Bolei Zhou, and Stefanie Jegelka. Optimization as estimation with Gaussian processes in bandit settings. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016a.

Zi Wang, Chengtao Li, Stefanie Jegelka, and Pushmeet Kohli. Batched high-dimensional Bayesian optimization via structural kernel learning. In *International Conference on Machine Learning (ICML)*, 2017.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016b.