BOCK: Bayesian Optimization with Cylindrical Kernels

Oh, C.; Gavves, E.; Welling, M.

Publication date
2018

Document Version
Final published version

Published in
Proceedings of Machine Learning Research

License
Other

Citation for published version (APA):
http://proceedings.mlr.press/v80/oh18a.html

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
**BOCK : Bayesian Optimization with Cylindrical Kernels**

ChangYong Oh \(^1\) Efstratios Gavves \(^1\) Max Welling \(^1\) \(^2\)

**Abstract**

A major challenge in Bayesian Optimization is the **boundary issue** (Swersky, 2017) where an algorithm spends too many evaluations near the boundary of its search space. In this paper we propose BOCK, Bayesian Optimization with Cylindrical Kernels, whose basic idea is to transform the ball geometry of the search space using a cylindrical transformation. Because of the transformed geometry, the Gaussian Process-based surrogate model spends less budget searching near the boundary, while concentrating its efforts relatively more near the center of the search region, where we expect the solution to be located. We evaluate BOCK extensively, showing that it is not only more accurate and efficient, but it also scales successfully to problems with a dimensionality as high as 500. We show that the better accuracy and scalability of BOCK even allows optimizing modestly sized neural network layers, as well as neural network hyperparameters.

1. Introduction

When we talk about stars and galaxies we use parsecs to describe structures, yet when we discuss the world around us we use meters. In other words, the natural lengthscale scale with which we describe the world increases with distance away from us. We believe this same idea is useful when performing optimization in high dimensional spaces.

In Bayesian Optimization (or other forms of hyperparameter optimization) we define a cube or a ball and search for the solution inside that volume. The origin of that sphere is special in the sense that this represents the part of space with the highest probability if finding the solution. Moreover, in high dimensions, when we move outwards, the amount of volume contained in an annulus with width \(\delta R\), \(A(c; R−\delta R, R) = \{x| R−\delta R < \|x−c\| < R\}\), grows exponentially with distance \(R\). As such, if we would spend an equal amount of time searching each volume element \(\delta V\), we would spend all our time at the boundary of our search region. This effective attraction to the places with more volume is the equivalent of an “entropic force” in physics, and in the case of optimization is highly undesirable, since we expect the solution at a small radius \(R\).

In this paper we, therefore, reformulate Bayesian Optimization in a transformed space, where a ball, \(B(x; R) = \{x|\|x−c\| \leq R\}\), is mapped to a cylinder, \(C(p, q; c, L = \{(r, a)\|r \in [p, q], \|d−c\| = L\}\) (see Figure 1). In this way, every annulus of width \(\delta R\) contains an equal amount of volume for every radius \(R\), and the entropic force pulling the optimizer to the boundary disappears. We call our method BOCK, for **Bayesian Optimization with Cylindrical Kernel**. We find that our algorithm is able to successfully handle much higher dimensional problems than standard Bayesian optimizers. As a result, we manage to not only optimize modestly sized neural network layers (up to 500 dimensions in our experiments), obtaining solutions competitive to SGD training, but also hyper-optimize stochastic depth Resnets (Huang et al., 2016).

---

\(^1\)QUvA Lab, Informatic Institute, University of Amsterdam, Amsterdam, Netherlands. \(^2\)Canadian Institute for Advanced Research, Toronto, Canada. Correspondence to: ChangYong Oh <C.Oh@uva.nl>.

Algorithm 2 Bayesian Optimization pipeline.

1: **Input:** surrogate model $\mathcal{M}$, acquisition function $\alpha$, search space $X$, initial training data $D_{\text{init}}$, function $f$  
2: **Output:** optimum $x_{\text{opt}} \in X$ of $f$  
3: Initialize $D = D_{\text{init}}$  
4: while evaluation budget available do  
5: Set $\mu(\cdot|D), \sigma^2(\cdot|D) \leftarrow \mathcal{M}|D$ // Surrogate function returns predictive mean function and predictive variance function by fitting $\mathcal{M}$ to $D$  
6: Maximize $\hat{x} = \max_{x \in X} \alpha(\mu(x|D), \sigma^2(x|D))$ // Acquisition function suggests next evaluation by maximization  
7: Evaluate $\hat{y} = f(\hat{x})$ // Evaluate the score of the point selected by the acquisition function  
8: Set $D \leftarrow D \cup \{ (\hat{x}, \hat{y}) \}$ // Update the training dataset by including the newly evaluated pair $(\hat{x}, \hat{y})$  
9: end while

2. Background

2.1. Bayesian Optimization

Bayesian optimization aims at finding the global optimum of black-box functions, namely

$$x_{\text{opt}} = \arg \min_{x} f(x)$$

The general pipeline of Bayesian Optimization is given in Alg. 1. Prior to starting, a search space must be defined, where the optimum $f(x_{\text{opt}})$ will be searched for. Given this search space, the initial training dataset must be set, typically by naive guessing where the solution might lie or by informed expert knowledge of the problem. Having completed these two steps, Bayesian Optimization proceeds in an iterative fashion. At each round, in the absence of any other information regarding the nature of $f(x)$ a surrogate model attempts to approximate the behavior of $f(x)$ based on the so far observed points $(x_i, y_i), y_i = f(x_i)$. The surrogate function is then followed by an acquisition function that suggests the next most interesting point $x_{i+1}$ that should be evaluated. The pair $(x_i, y_i)$ is added to the training dataset, $D = D \cup \{ (x_i, y_i) \}$, and the process repeats until the optimization budget is depleted.

The first design choice of the Bayesian Optimization pipeline is the surrogate model. The task of the surrogate model is to model probabilistically the behavior of $f(\cdot)$ in the $x$-space in terms of (a) a predictive mean $\mu(x_\cdot | D)$ that approximates the value of $f(x)$ at any point $x_\cdot$, and (b) a predictive variance that represents the uncertainty of the surrogate model in this prediction. Any model that can provide a predictive mean and variance can be used as a surrogate model, including random forests (Hutter et al., 2011), tree-based models (Bergstra et al., 2011) and neural networks (Snoek et al., 2015; Springenberg et al., 2016).

Among other things, Gaussian Processes not only provide enough flexibility it terms of kernel design but also allow for principled and tractable quantification of uncertainty (Rasmussen & Williams, 2006). Therefore, we choose Gaussian Processes as our surrogate model. The predictive mean and the predictive variance of Gaussian processes are given as below

$$\mu(x_i | D) = K_{\ast D}(K_D D + \sigma^2 I)^{-1} y$$

$$\sigma^2(x_i | D) = K_{\ast \ast} - K_{\ast D}(K_D D + \sigma^2 I)^{-1} K_{D \ast}$$

where $K_{\ast \ast} = K(x_i, x_i)$, $K_{\ast D}$ is a row vector whose $i$th entry is $K(x_i, x_\ast)$, $K_{D \ast} = (K_D D)^T$, $[K_D D]_{i,i} = K(x_i, x_j)$, $\sigma^2_{\text{obs}}$ is the variance of observational noise and $D = \{ (x_i, y_i) \}_i$ is the dataset of observations so far.

The second design choice of the Bayesian Optimization pipeline is the acquisition function. The predictive mean and the predictive variance from the surrogate model is input to the acquisition function that quantifies the significance of every point in $x$ as a next evaluation point. While different acquisition functions have been explored in the literature (Thompson, 1933; Kushner, 1964; Močkus, 1975; Srinivas et al., 2009; Hennig & Schuler, 2012; Hernández-Lobato et al., 2014), they all share the following property: they return high scores at regions of either high predictive variance (high but uncertain reward), or low predictive mean (modest but certain reward).

Last, the third design choice of the Bayesian Optimization pipeline, often overlooked, is the search space. In (Snoek et al., 2014) the kernel of the surrogate model is defined on a warped search space, thus allowing for a more flexible modeling of $f(x)$ by the surrogate function. As the search space defines where optimal solutions are to be sought for, the search space definition is a means of infusing prior knowledge into the Bayesian Optimization. Usually, a search space is set so that the expected optimum is close to the center.

2.2. High-dimensional Bayesian Optimization

Even with its successes in many applications, several theoretical as well as practical issues (Shahriari et al., 2016) still exist when employing Bayesian Optimization to real world problems. Among others, many Bayesian optimization algorithms are restricted in practice to problems of moderate dimensions. In high dimensional problems, one suffers from the curse of dimensionality. To overcome the curse of dimensionality, several works make structural assumptions, such as low effective dimensionality (Wang et al., 2016; Bergstra & Bengio, 2012) or additive structure (Kandasamy et al., 2015; Wang et al., 2017).

Because of the way Gaussian Processes quantify uncertainty, the curse of dimensionality is a serious challenge for
Gaussian Processes-based Bayesian Optimization in high dimensions. Since in high dimensions data points typically lie mostly on the boundary, and anyways far away from each other, the predictive variance tends to be higher in the regions near the boundary. Thus, the acquisition function is somewhat biased to choose evaluations near the boundary, hence, biasing Bayesian Optimization towards solution near the boundary and away from the center, contradicting with the prior assumption. This is the boundary issue (Swersky, 2017).

2.3. Contributions

Different from the majority of the Bayesian Optimization methods that rely on a Euclidean geometry of the search space implicitly or explicitly (Hutter et al., 2011; Bergstra et al., 2012; Snoek et al., 2012; 2014; 2015; Swersky et al., 2013; Wang et al., 2017), the proposed BOCK applies a cylindrical geometric transformation on it. The effect is that the volume near the center of the search space is expanded, while the volume near the boundary is shrunk. Compared to (Snoek et al., 2014; Binois et al., 2015), where warping functions were introduced with many kernel parameters to be learned, we do not train transformations. Also, we avoid learning many additional kernel parameters for better efficiency and scalability. Because of the transformation, the proposed BOCK solves also the issue of flat optimization surfaces of the acquisition function in high dimensional spaces (Rana et al., 2017). And compared to REMBO (Wang et al., 2016; Binois et al., 2015), BOCK does not rely on assumptions of low dimensionality of the latent search space.

3. Method

3.1. Prior assumption and search space geometry

The flexibility of a function $f$ on a high-dimensional domain $X$ can be, and usually is, enormous. To control the flexibility and make the optimization feasible some reasonable assumptions are required. A standard assumption in Bayesian Optimization is the prior assumption (Swersky, 2017), according to which the optimum of $f(x)$ should lie somewhere near the center of the search space $X$. Since the search space is set with the prior assumption in mind, it is reasonable for Bayesian Optimization to spend more evaluation budget in areas near the center of $X$. It is interesting to study the relation of the prior assumption and the geometry of the search space. The ratio of the volume of two concentric balls $B(c; R - \delta R)$ and $B(c; R)$, with a radius difference of $\delta R$, is

$$\frac{\text{volume}(B(c; R - \delta R))}{\text{volume}(B(c; R))} = o((1 - \delta)^D), \quad (4)$$

which rapidly goes to zero with increasing dimensionality $D$. This means that the volume of $B(c; R)$ is mostly concentrated near the boundary, which in combination with Gaussian processes’ behavior of high predictive variance at points far from data, creates the boundary issue (Swersky, 2017).

It follows, therefore, that with a transformation of the search space we could avoid excessively biasing our search towards large values of $R$.

3.2. Cylindrical transformation of search space

The search space geometry has a direct influence on the kernel $K(x, x')$ of the Gaussian Process surrogate model, and, therefore, its predictive variance $\sigma^2(x)$, see eq. (3). A typical design choice for Gaussian Processes (Snoek et al., 2012; 2014; González et al., 2016) are stationary kernels, $K(x, x') \propto f(x - x')$. Unfortunately, stationary kernels are not well equipped to tackle the boundary issue. Specifically, while stationary kernels compute similarities only in terms of relative locations $x - x'$, the boundary issue dictates the use of location-aware kernels $K(x, x')$ to recognize whether $x, x'$ lie near the boundary or the center areas of the search space.

A kernel that can address this should have the following two properties. First, the kernel must define the similarity between two points $x, x'$ in terms of their absolute locations, namely the kernel has to be non-stationary. Second, the kernel must transform the geometry of its input (i.e., the search space for the Gaussian Process surrogate model) such that regions near the center and the boundaries are equally represented. To put it otherwise, we need a geometric transformation of the search space that expands the region near the center while contracting the regions near the boundary. A transformation with these desirable properties is the cylindrical one, separating the radius and angular components of a point $x$, namely

$$T(x) = \begin{cases} (\|x\|_2, x/\|x\|_2) & \text{for } \|x\|_2 \neq 0 \\ (0, a_{\text{arbitrary}}) & \text{for } \|x\|_2 = 0 \end{cases} \quad (5)$$

$$T^{-1}(r, a) = ra$$

where $a_{\text{arbitrary}}$ is an arbitrarily chosen vector with unit $l_2$-norm. Although polar coordinate (Padonou & Roustant, 2016) appears to be able to used for the same purpose, our specific choice of transformation does not suffer from numerical instability.¹

After applying the geometric transformation we arrive at a new kernel $K_{\text{cyl}}(x_1, x_2)$, which we will refer to as the cylindrical kernel. The geodesic similarity measure (kernel)

¹In high dimensional spaces, the inverse transformation from spherical to rectangular coordinate entails multiplication of many trigonometric functions, causing numerical instabilities because of large products of small numbers.
of \( K_{cyl} \) on the transformed cylinder, \( T(X) \), is defined as

\[
K_{cyl}(x_1, x_2) = \tilde{K}(T(x_1), T(x_2)) = K_r(r_1, r_2) \cdot K_\alpha(a_1, a_2)
\]

where the final kernel decomposes into a 1-D radius kernel \( K_r \) measuring the similarity of the radii of \( r_1, r_2 \) and a angle kernel \( K_\alpha \).

For the angle kernel \( K_\alpha(a_1, a_2) \), we opt for a continuous radial kernel on the (hyper-)sphere (Jayasumana et al., 2014),

\[
K_\alpha(a_1, a_2) = \sum_{p=0}^{p} c_p(a_1^p a_2^p), \quad c_p \geq 0, \forall p
\]

with trainable kernel parameters of \( c_0, \cdots, c_P \) and \( P \) user-defined. The advantages of a continuous radial kernel is two-fold. First, with increasing \( P \) a continuous radial kernel can approximate any continuous positive definite kernel on the sphere with arbitrary precision (Jayasumana et al., 2014).

Second, the cylindrical kernel has \( P + 1 \) parameters, which is independent of the dimensionality of \( X \). This means that while the continuous radial kernel retains enough flexibility, only few additional kernel parameters are introduced, which are independent of the dimensionality of the optimization problem and can, thus, easily scale to more than 50 dimensions. This compares favorably to Bayesian optimization with ARD kernels that introduce at least \( d \) kernel parameters for a \( d \)-dimensional search space.

Although the boundary issue is mitigated by the cylindrical transformation of the search space, the prior assumption (good solutions are expected near the center) can be promoted. To this end, and to reinforce the near-center expansion of the cylindrical transformation, we consider input warping (Snoek et al., 2014) on the radius kernel \( K_r(r_1, r_2) \). Specifically, we use the cumulative distribution function of the Kumaraswamy distribution, \( \text{Kuma}(r|\alpha, \beta) = 1 - (1 - r^\alpha)^\beta \) (with \( \alpha > 0, \beta > 0 \)),

\[
K_r(r_1, r_2) = K_{\text{base}}(\text{Kuma}(r_1|\alpha, \beta), \text{Kuma}(r_1|\alpha, \beta)) = K_{\text{base}}(1 - (1 - r_1^\alpha)^\beta, 1 - (1 - r_2^\alpha)^\beta|\alpha, \beta)
\]

where the non-negative \( \alpha, \beta \) are learned together with the kernel parameters. \( K_{\text{base}} \) is the base kernel for measuring the radius-based similarity. Although any kernel is possible for \( K_{\text{base}} \), in our implementations we opt for the Matérn52 kernel used in Spearmint (Snoek et al., 2012). By making radius warping concave and non-decreasing, \( K_r \) and, in turn, \( K_{cyl} \) focus more on areas with small radii.

Overall, the transformation of the search space has two effects. The first effect is that the volume is redistributed, such that areas near the center are expanded, while areas near the boundaries are contracted. Bayesian optimization’s attention in the search space, therefore, is also redistributed from the boundaries to the center of the search space. The second effect is that the kernel similarity changes, such that the predictive variance depends mostly on the angular difference between the existing data points and the ones to be evaluated. An example is illustrated in Fig. 1, where our dataset comprises of \( D = \{x_1, x_2\} \) and the acquisition function must select between two points, \( x_{\star 1} \) and \( x_{\star 2} \). Whereas in the original Euclidean geometry (Figure 1 to the left) \( x_{\star 1} \) is further away from \( D \), thus having higher predictive variance, in the cylindrical geometry both \( x_{\star 1} \) and \( x_{\star 2} \) are equally far, thus reducing the artificial preference to near-boundary points.

3.3. Balancing center over-expansion

The transformation \( T \) maps an annulus \( A(0; R - \delta R, R) \) of width \( \delta R \) to the cylinder \( C(R - \delta R, R; 0, 1) \), where \( (0, 1) \) is the center and the radius of the cylinder. For almost any point in the original ball there is a one-to-one mapping to a point on the cylinder. The only exception is the extreme case of the ball origin, which is mapped to the 0-width sphere \( C(0, 0; 0, 1) = \{(0, a)||a|| = 1\} \) on the base of the cylinder (bright green circle in the Figure 2 to the right). Namely, the center point \( x_{\text{center}} \) is overly expanded, corresponding to a set of points. Because of the one-to-many correspondence between \( x_{\text{center}} \) and \( C(0, 0; 0, 1) \), an arbitrary point is selected in eq. (5).

Unfortunately, the dependency on a point that is both arbitrary and fixed incurs an arbitrary behavior of \( K_{cyl} \) as well. For any point \( x_\star \in X \setminus \{0\} \) the kernel \( K_{cyl}(x_{\text{center}}, x_\star) \) changes arbitrarily, depending on the choice of \( a_{\text{arbitrary}} \), see Figure 2. Having a fixed arbitrary point, therefore, is undesirable as it favors points lying closer to it. To this end, we define \( a_{\text{arbitrary}} \) as the angular component of a test point \( x_\star \), \( a_{\text{arbitrary}} = x_\star / ||x_\star|| \), thus being not fixed anymore. Geometrically, this is equivalent to using the point in \( C(0, 0; 0, 1) \) closest to \( T(x_\star) \), see Figure 2 to the right. This implies that, if the origin is in the dataset, the Gram matrix needed for computing the predictive density now depends on the angular location of the test point under consideration.
Similar method constructing gram matrix dependent with prediction point is proposed in (Pronzato & Rendas, 2017), where the functional form of the kernel changes according to prediction point, while training data changes in BOCK. This may look somewhat unconventional but still well behaved (the kernel is still positive definite and the predictive mean and variance change smoothly). More details can be found in the supplementary material.

4. Experiments

In Bayesian optimization experiments, we need to define (a) how to train the surrogate model, (b) how to optimize the acquisition function and (c) how to set the search space. For BOCK we use Gaussian Process surrogate models, where following (Snoek et al., 2012; 2014) we train parameters of BOCK with MCMC (slice sampling (Murray & Adams, 2010; Neal, 2003)) . For the acquisition function, we use the Adam (Kingma & Ba, 2014) optimizer, instead of L-BFGS-B (Zhu et al., 1997). To begin the optimization we feed 20 initial points to Adam. To select the 20 initial points, a sobol sequence (Bratley & Fox, 1988) of 20,000 points is generated on the cube (we used the cube for fair comparison with others). The acquisition function is evaluated on these points and the largest 20 points are chosen as the initial ones. Instead of using a static sobol sequence in the entire course of Bayesian optimization (Snoek et al., 2012; 2014), we generate different sobol sequences for different evaluations, as fixed grid point impose too strong constraints in high dimensional problems. In the d-dimensional space, our search space is a ball $B(0, \sqrt{d})$ circumscribing a cube $[-1, 1]^d$, which is the scaled and translated version of the typical search region, unit cube $[0, 1]^d$. Our search space is much larger than a cube. By generating sobol sequence on the cube, the reduction of the boundary issue mostly happens at corners of the cube $[-1, 1]^d$. The implementation is available online (https://github.com/ChangYong-Oh/HyperSphere)

4.1. Benchmarking

First, we compare different Bayesian Optimization methods and BOCK on four benchmark functions. Specifically, following (Eggersperger et al., 2013; Laguna & Marti, 2005) we use the repeated Branin, repeated Hartmann6 and Levy to assess Bayesian Optimization in high dimensions. To test the ability of Bayesian Optimization methods to optimize functions with more complex structure and stronger intra-class dependencies, we additionally include the Rosenbrock benchmark, typically used as benchmark for gradient-based optimization (Laguna & Marti, 2005). The precise formulas for the four benchmark functions are added to the supplementary material. We solve the benchmark functions in 20 and 100 dimensions, using 200 and 600 function evaluations respectively for all Bayesian Optimization methods. We compare the proposed BOCK with the following Bayesian Optimization methods using publicly available software: SMAC (Hutter et al., 2011), TPE (Bergrsta et al., 2011), Spearmint (Snoek et al., 2012), Spearmint+ (Snoek et al., 2014), additive BO (Kandasamy et al., 2015), elastic BO (Rana et al., 2017). We also report an in-house improved Spearmint implementation, which we refer to as Matérn. 3

We focus on four aspects: (a) accuracy, (b) efficiency (wall clock time) vs accuracy, (c) scalability (number of dimensions) vs efficiency, and (d) robustness of BOCK to hyperparameters and other design choices. We study (a) in all four benchmark functions. For brevity, we report (b)-(d) on the Rosenbrock benchmark only, the hardest of the four benchmark functions for all Bayesian Optimization methods in terms of accuracy, and report results the rest of the benchmark functions in the supplementary material.

Accuracy. We first present the results regarding the accuracy of BOCK and the Bayesian Optimization baselines in Table 1. BOCK and Matérn outperform others with large margin in discovering near optimal solutions. For benchmark functions with complicated dependencies between variables, such as the repeated Hartmann6 and Rosenbrock, BOCK consistently discovers smaller values compared to other baselines, while not being affected by an increasing number of dimensions. What is more, BOCK is on par even with methods that are designed to exploit the specific geometric structures, if the same geometric structures can be found in the evaluated functions. For instance, the repeated Branin and Levy have an additive structure, where the same low dimensional structure is repeated. The non-ARD kernel of Matérn can exploit such special, additive structures. BOCK is able to reach a similar near-optimum solution without being explicitly designed to exploit such structures.

3Differences with standard Spearmint: (a) a non-ARD, Matém52 kernel for the surrogate model, (b) dynamic search grid generation per evaluation, (c) Adam (Kingma & Ba, 2014) instead of L-BFGS-B (Zhu et al., 1997), (d) more updates for optimizer.
Table 1. Bayesian Optimization on four benchmark functions for 20 and 200 dimensions, with the exception of Spearmint+ (Snoek et al., 2014) and Elastic BO (Rana et al., 2017) evaluated only on the 20-dimensional cases because of prohibitive execution times. For benchmark functions with complicated dependencies between variables (repeated Hartmann6, Rosenbrock), BOCK consistently discovers good solutions compared to other baselines, while not being affected by an increasing number of dimensions. Also, BOCK matches the accuracies of methods, like Matérn, designed to exploit specific geometric structures, e.g., the additive structures of repeated Branin and Levy. We conclude that BOCK is accurate, especially when we have no knowledge of the geometric landscape of the evaluated functions.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>REPEATED BRANIN</th>
<th>REPEATED HARTMANN6</th>
<th>ROSENROCK</th>
<th>LEVY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions</td>
<td>20</td>
<td>100</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.3979</td>
<td>0.3979</td>
<td>-3.3223</td>
<td>-3.3223</td>
</tr>
<tr>
<td>BOCK</td>
<td>0.12</td>
<td>1.03</td>
<td>-0.20</td>
<td>0.26</td>
</tr>
<tr>
<td>Matérn</td>
<td>0.41</td>
<td>0.54</td>
<td>-0.29</td>
<td>0.26</td>
</tr>
<tr>
<td>SPEARMINT+</td>
<td>5.75</td>
<td>14.07</td>
<td>-0.03</td>
<td>0.13</td>
</tr>
<tr>
<td>ADITITIVE BO</td>
<td>6.77</td>
<td>0.54</td>
<td>-0.28</td>
<td>0.25</td>
</tr>
<tr>
<td>ELASTIC BO</td>
<td>0.41</td>
<td>0.54</td>
<td>-0.29</td>
<td>0.26</td>
</tr>
</tbody>
</table>

We conclude that BOCK is accurate, especially when we have no knowledge of the geometric landscape of the evaluated functions. We focus on the Bayesian Optimization methods with competitive performance, namely BOCK, Spearmint and Matérn.

**Efficiency vs accuracy.** Next, we compare in Figure 3 the accuracy of the different Bayesian Optimization methods as a function of their wall clock times for the 20-dimensional case for Rosenbrock. As the function minimum is \( f(x_{\text{opt}}) = 0 \), the optimal operating point is at \((0, 0)\). BOCK is the closest to the optimal point. Matérn is the second most accurate, while being considerably slower to run. SMAC (Hutter et al., 2011) and AdditiveBO (Kandasamy et al., 2015) are faster than BOCK, however, they are also considerably less accurate.

**Scalability.** In Figure 4 we evaluate the most accurate Bayesian Optimization methods from Table 1 (Spearmint, Matérn and BOCK.) with respect to how scalable they are, namely measuring the wall clock time for an increasing number of dimensions. Compared to Spearmint BOCK is less affected by the increasing number of dimensions. Not only the BOCK surrogate kernel requires fewer parameters, but also the number of surrogate kernel parameters is independent of the number of input dimensions, thus making the surrogate model fitting faster. BOCK is also faster than Matérn, although the latter uses a non-ARD kernel that is also independent of the number of input dimensions. Presumably, this is due to a better, or smoother, optimization landscape after the cylindrical transformation of geometry of the input space, affecting positively the search dynamics. We conclude that BOCK is less affected by the increasing number of dimensions, thus scaling better.

**Robustness.** To study the robustness of BOCK to design choices, we compare three BOCK variants. The first is the standard BOCK as described in Section 3. The second variant, BOCK-W, removes the input warping on the radius component. The third variant, BOCK+B, includes an additional boundary treatment to study whether further reduction of the predictive variance is beneficial. Specifically, we reduce the predictive variance by adding “fake” data. We present results in Table 2.
Figure 5. Training on MNIST a two-layer neural network: $784 \xrightarrow{W_{1,b_1}} N_{\text{hidden}} \xrightarrow{W_{2,b_2}} 10$. For all experiments, $W_1$, $b_1$ and $b_2$ are optimized with Adam (Kingma & Ba, 2014) and $W_2$ with Bayesian Optimization. In this experiment, Bayesian Optimization repeats the following steps. (a) A new $W_2$ is suggested by BOCK. (b) Given this $W_2$, the $W_1, b_1, b_2$ are fine-tuned by SGD on the training set. (c) The loss on the validation is returned as the evaluation on $W_2$. Therefore, in this experiment, Bayesian Optimization algorithms optimize the validation loss. We observe that BOCK can optimize successfully a modestly sized neural network layer. BOCK consistently finds a better solution than existing Bayesian optimization algorithms. In high dimensional cases, BOCK outperforms other algorithms with a significant margin. We conclude that BOCK is capable of optimizing in high-dimensional and complex spaces.

### Table 2. Comparison between different BOCK variants on Rosenbrock

<table>
<thead>
<tr>
<th>DIMENSIONS</th>
<th>20</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOCK</td>
<td>47.87±</td>
<td>33.94</td>
<td>29.65±</td>
</tr>
<tr>
<td>BOCK-W</td>
<td>1314.03±</td>
<td>1619.73</td>
<td>51.14±</td>
</tr>
<tr>
<td>BOCK+B</td>
<td>48.87±</td>
<td>18.33</td>
<td>33.90±</td>
</tr>
</tbody>
</table>

Removing the input warping on the radius is hurting the robustness, as BOCK-W tends to reach slightly worse minima than BOCK. However, introducing additional boundary treatments brings only marginal benefits.

Further, we assess the sensitivity of BOCK with respect to the hyperparameter $P$ in eq.(5). For $P = 3, 5, 7, 9$, we observe that higher $P$ tends to give slightly better minima, while increasing the computational cost.

For clarity of presentation, as well as to maintain the experimental efficiency, in the rest of the experiments we focus on BOCK with $P = 3$.

### 4.2. Optimizing a neural network layer

As BOCK allows for accurate and efficient Bayesian Optimization for high-dimensional problems, we next perform a stress test, attempting to optimize neural network layers of 100, 200 and 500 dimensions. Specifically, we define a two-layered neural network with architecture: $784 \xrightarrow{W_{1,b_1}} N_{\text{hidden}} \xrightarrow{W_{2,b_2}} 10$, using ReLU as the intermediate non-linearity.

In this experiment, we split the data set into train (first 45000 images of MNIST train data set), validation (next 5000 images of MNIST train data set) and test (10000 images of MNIST test data set). For all Bayesian optimization experiments $W_1, b_1$ and $b_2$ are optimized with Adam (Kingma & Ba, 2014) and $W_2$ with Bayesian Optimization. The training proceeds as follows. First, Bayesian Optimization suggests a $W_2$ based on evaluations on the validation set. Given this $W_2$ we train on the train sets the $W_1, b_1, b_2$ with Adam, then repeat.

We show the validation loss in Figure 5, where we report mean and standard deviation over 5 runs for all methods. We observe that BOCK clearly outperforms Spearmint and Matérn in terms of validation loss in Figure 5, with the gap increasing for higher $W_2$ dimensions. This show that BOCK effectively optimizes a target quantity (validation loss) and thus is a competitive optimizer even for high dimensional problems.

Evaluation of generalization performance is given in Table 3. For Bayesian Optimization algorithms, given each $W_2$ optimizing loss on the validation set, we train $W_1, b_1$ and $b_2$ on the train+validation set 5 times with Adam. For SGD, we train a network with Adam on the train+validation set and report test loss 5 times. We compare BOCK with the competitive Spearmint and Matérn on both validation loss and test loss. To the best of our knowledge we are the first to apply Gaussian Process-based Bayesian Optimization in so high-dimensional and complex, representation learning spaces.  

Somewhat surprisingly, BOCK is able to match and even outperform the Adam-based SGD in terms of generalization. There are three reasons for this. First, in this experiment, all Bayesian optimization algorithms directly optimize the loss.

---

5To our knowledge, running Bayesian Optimization on 200 or 500 dimensional problems has only been tried with methods assuming low effective dimensionality (Wang et al., 2016; Chen et al., 2012)
Table 4. Frobenius Norm of optimized $W_2$. From 5 runs of Bayesian optimization, we have 5 optimized $W_2$ and mean and std of 5 of them are reported.

<table>
<thead>
<tr>
<th>Method</th>
<th>Frobenius Norm Mean ± Std</th>
<th>100</th>
<th>200</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpearMint</td>
<td>7.4491 ± 0.3768</td>
<td>7.8577 ± 1.3314</td>
<td>11.6091 ± 2.4745</td>
<td></td>
</tr>
<tr>
<td>Matérn</td>
<td>6.0120 ± 0.2843</td>
<td>8.6870 ± 0.2014</td>
<td>13.1189 ± 0.3077</td>
<td></td>
</tr>
<tr>
<td>BOCK</td>
<td>2.8805 ± 1.1378</td>
<td>3.3917 ± 1.6424</td>
<td>1.9467 ± 0.2183</td>
<td></td>
</tr>
<tr>
<td>ADAM-SGD</td>
<td>1.9734 ± 0.2350</td>
<td>5.1930 ± 0.2975</td>
<td>7.1933 ± 0.1111</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Test loss with optimized $W_2$. Bayesian Optimization methods train $W_1$, $b_1$, and $b_2$ on train+validation set for each optimized $W_2$ 5 times (25 runs in total). SGD is trained 5 times on the train+validation set and its test loss is reported.

<table>
<thead>
<tr>
<th>Method</th>
<th>Test Acc. ± Std</th>
<th>Val. Acc. ± Std</th>
<th>Exp. Depth ± Std</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESNet-110</td>
<td>73.03 ± 0.36</td>
<td>72.98 ± 0.43</td>
<td>110.00</td>
<td></td>
</tr>
<tr>
<td>SDResNet-110+BOCK</td>
<td>75.06 ± 0.04</td>
<td>74.90 ± 0.15</td>
<td>82.50</td>
<td></td>
</tr>
<tr>
<td>SDResNet-110+LINEAR</td>
<td>75.21 ± 0.05</td>
<td>75.06 ± 0.19</td>
<td>74.51 ± 1.22</td>
<td></td>
</tr>
</tbody>
</table>

BOCK : Bayesian Optimization with Cylindrical Kernels

4.3. Hyper-optimizing stochastic depth ResNets

As BOCK allows for accurate and efficient Bayesian Optimization, in our last experiment we turn our attention to a practical hyperparameter optimization application. Stochastic Depth ResNet (SDResNet) (Huang et al., 2016) was shown to obtain better accuracy and faster training by introducing a stochastic mechanism that randomly suppresses ResNet blocks (ResBlock) (He et al., 2016). The stochastic mechanism for dropping ResBlocks is controlled by a vector $p \in [0, 1]^t$ of probabilities for $t$ ResBlocks, called “death rate”. In (Huang et al., 2016) a linearly increasing (from input to output) death rate was shown to improve accuracies.

Instead of pre-defined death rates, we employ BOCK to find the optimal death rate vector for SDRes-110 on CIFAR100 (Krizhevsky & Hinton, 2009). We first train an SDResNet for 250 epochs and linear death rates with exactly the same configuration in (Huang et al., 2016) up to 250 epochs. In this experiment BOCK has access to the training and validation set only. Then, per iteration BOCK first proposes the next candidate $p$ based on evaluation on the validation set. Given the candidate $p$ we run 100 epochs of SGD on the training set and repeat with an annealed learning rate (0.01 for 50 epochs, then 0.001 for 50 more). We initialize the death rate vector to $p = [0.5, 0.5, ..., 0.5]$. We report the final accuracies computed in the unseen test set in Table 5, using only 50 evaluations.

We observe that BOCK learns a vector $p$ that results in an improved validation accuracy compared to SDResNet, all the while allowing for a lower expected depth. The improved validation accuracy materializes to an only slightly better test accuracy, however. One reason is that optimization is not directly equivalent to learning, as also explained in Section 4.2. What is more, it is likely that the accuracy of SDResNet-110 on CIFAR-100 is maxed out, especially considering that only 50 evaluations were made. We conclude that BOCK allows for successful and efficient Bayesian Optimization even for practical, large-scale learning problems.

5. Conclusion

We propose BOCK, Bayesian Optimization with Cylindrical Kernels. Many of the problems in Bayesian Optimization relate to the boundary issue (too much value near the boundary), and the prior assumption (optimal solution probably near the center). Because of the boundary issue, not only much of the evaluation budget is unevenly spent to the boundaries, but also the prior assumption is violated. The basic idea behind BOCK is to transform the ball geometry of the search space with a cylindrical transformation, expanding the volume near the center while contracting it near the boundaries. As such, the Bayesian optimization focuses less on the boundaries and more on the center.

We test BOCK extensively in various settings. On standard benchmark functions BOCK is not only more accurate, but also more efficient and scalable compared to state-of-the-art Bayesian Optimization alternatives. Surprisingly, optimizing a neural network up to 500 dimensions with BOCK allows for even better parameters than SGD with Adam (Kingma & Ba, 2014) with respect to both validation loss and test loss. And hyper-optimizing the “death rate” of stochastic depth ResNet (Huang et al., 2016) results in smaller ResNets while maintaining accuracy.

We conclude that BOCK allows for accurate, efficient and scalable Gaussian Process-based Bayesian Optimization.
References


