Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces

Kevin Swersky  
University of Toronto  
kswersky@cs.toronto.edu

David Duvenaud  
University of Cambridge  
dkd23@cam.ac.uk

Jasper Snoek  
Harvard University  
jsnoek@seas.harvard.edu

Frank Hutter  
Freiburg University  
fh@informatik.uni-freiburg.de

Michael A. Osborne  
University of Oxford  
mosb@robots.ox.ac.uk

Abstract

In practical Bayesian optimization, we must often search over structures with differing numbers of parameters. For instance, we may wish to search over neural network architectures with an unknown number of layers. To relate performance data gathered for different architectures, we define a new kernel for conditional parameter spaces that explicitly includes information about which parameters are relevant in a given structure. We show that this kernel improves model quality and Bayesian optimization results over several simpler baseline kernels.

1 Introduction

Bayesian optimization (BO) is an efficient approach for solving blackbox optimization problems of the form $\arg\min_{x \in X} f(x)$ (see [1] for a detailed overview), where $f$ is expensive to evaluate. It employs a prior distribution $p(f)$ over functions that is updated as new information on $f$ becomes available. The most common choice of prior distribution are Gaussian processes (GPs [2]), as they are powerful and flexible models for which the marginal and conditional distributions can be computed efficiently. However, some problem domains remain challenging to model well with GPs, and the efficiency and effectiveness of Bayesian optimization suffers as a result. In this paper, we tackle the common problem of input dimensions that are only relevant if other inputs take certain values [6, 5]. This is a general problem in algorithm configuration [6] that occurs in many machine learning contexts, such as, e.g., in deep neural networks [7]; flexible computer vision architectures [8]; and the combined selection and hyperparameter optimization of machine learning algorithms [9]. We detail the case of deep neural networks below.

Bayesian optimization has recently been applied successfully to deep neural networks [10, 5] to optimize high level model parameters and optimization parameters, which we will refer to collectively as hyperparameters. Deep neural networks represent the state-of-the-art on multiple machine learning benchmarks such as object recognition [11], speech recognition [12], natural language processing [13] and more. They are multi-layered models by definition, and each layer is typically parameterized by a unique set of hyperparameters, such as regularization parameters and the layer capacity or number of hidden units. Thus adding additional layers introduces additional hyperparameters to be optimized. The result is a complex hierarchical conditional parameter space, which is difficult to search over. Historically, practitioners have simply built a separate model for each type

There are prominent exceptions to this rule, though. In particular, tree-based models, such as random forests, can be a better choice if there are many data points (and GPs thus become computationally inefficient), if the input dimensionality is high, if the noise is not normally distributed, or if there are non-stationarities [3, 4, 5].
of architecture [5] or assumed a fixed architecture [10]. However, if there is any relation between networks with different architectures, separately modeling each is wasteful.

GPs with standard kernels fail to model the performance of architectures with such conditional hyperparameters. To remedy this, the contribution of this paper is the introduction of a kernel that allows observed information to be shared across architectures when this is appropriate. We demonstrate empirically on a GP regression task and a Bayesian optimization task that this kernel models the conditional parameter space of a typical deep learning problem better than previous ad-hoc methods.

2 A Kernel for Conditional Parameter Spaces

GPs employ a positive-definite kernel function \( k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \) to model the covariance between function values. Typical GP models cannot, however, model the covariance between function values whose inputs have different (possibly overlapping) sets of relevant variables.

In this section, we construct a kernel between points in a space that may have dimensions which are irrelevant under known conditions (further details are available in [14]). As an explicit example, we consider a deep neural network: if we set the network depth to 2 we know that the 3rd layer’s hyperparameters do not have any effect (as there is no 3rd layer).

Formally, we aim to do inference about some function \( f \) with domain \( \mathcal{X} \). \( \mathcal{X} = \prod_{i=1}^{D} \mathcal{X}_i \) is a \( D \)-dimensional input space, where each individual dimension is bounded real, that is, \( \mathcal{X}_i = [l_i, u_i] \subset \mathbb{R} \) (with lower and upper bounds \( l_i \) and \( u_i \), respectively). We define functions \( \delta_i : \mathcal{X} \to \{ \text{true}, \text{false} \} \), for \( i \in \{1, \ldots, D \} \). \( \delta_i(x) \) stipulates the relevance of the \( i \)th feature \( x_i \) to \( f(x) \).

2.1 The problem

As an example, imagine trying to model the performance of a neural network having either one or two hidden layers, with respect to the regularization parameters for each layer, \( x_1 \) and \( x_2 \). If \( y \) represents the performance of a one layer-net with regularization parameters \( x_1 \) and \( x_2 \), then the value \( x_2 \) doesn’t matter, since there is no second layer to the network. Below, we’ll write an input triple as \((x_1, \delta_2(x), x_2)\) and assume that \( \delta_1(x) = \text{true} \); that is, the regularization parameter for the first layer is always relevant.

In this setting, we want a kernel \( k \) to be dependent on which parameters are relevant, and the values of relevant parameters for both points. For example, consider first-layer parameters \( x_1 \) and \( x'_1 \):

- If we are comparing two points for which the same parameters are relevant, the value of any unused parameters shouldn’t matter,
  \[ k((x_1, \text{false}, x_2), (x'_1, \text{false}, x'_2)) = k((x_1, \text{false}, x_2'), (x'_1, \text{false}, x'_2')), \quad \forall x_2, x'_2, x_2', x'_2'; \]  

- The covariance between a point using both parameters and a point using only one should again only depend on their shared parameters,
  \[ k((x_1, \text{false}, x_2), (x'_1, \text{true}, x'_2)) = k((x_1, \text{false}, x_2'), (x'_1, \text{true}, x'_2')), \quad \forall x_2, x'_2, x_2', x'_2'. \]  

2.2 Cylindrical Embedding

We can build a kernel with these properties for each possibly irrelevant input dimension \( i \) by embedding our points into a Euclidean space. Specifically, the embedding we use is

\[
g_i(x) = \begin{cases} 
0, & \text{if } \delta_i(x) = \text{false} \\
\omega_i \sin \pi \rho_i \frac{x_i - l_i}{u_i - l_i}, & \text{if } \delta_i(x) = \text{true}
\end{cases}
\]

Where \( \omega_i \in \mathbb{R}^+ \) and \( \rho_i \in [0, 1] \).

Figure 1 shows a visualization of the embedding of points \((x_1, \delta_2(x), x_2)\) into \( \mathbb{R}^3 \). In this space, we have the Euclidean distance,

\[
d_i(x, x') = ||g_i(x) - g_i(x')||_2 = \begin{cases} 
0, & \text{if } \delta_i(x) = \delta_i(x') = \text{false} \\
\omega_i \sqrt{2 \left( 1 - \cos(\pi \rho_i \frac{x_i - x'_i}{u_i - l_i}) \right)}, & \text{if } \delta_i(x) = \delta_i(x') \\
\omega_i \sqrt{2 \left( 1 - \cos(\pi \rho_i \frac{x_i - x'_i}{u_i - l_i}) \right)}, & \text{if } \delta_i(x) = \delta_i(x') = \text{true}
\end{cases}
\]

We can use this to define a covariance over our original space. In particular, we consider the class of covariances that are functions only of the Euclidean distance \( \Delta \) between points. There are many
examples of such covariances. Popular examples are the exponentiated quadratic, for which \( \kappa(\Delta) = \sigma^2 \exp(-\frac{1}{2} \Delta^2) \), or the rational quadratic, for which \( \kappa(\Delta) = \sigma^2 (1 + \frac{1}{2\alpha} \Delta^2)^{-\alpha} \). We can simply take (4) in the place of \( \Delta \), returning a valid covariance that satisfies all desiderata above.

Explicitly, note that as desired, if \( i \) is irrelevant for both \( g \) and \( g' \), \( d_i \) specifies that \( g(x) \) and \( g(x') \) should not differ owing to differences between \( x_i \) and \( x'_i \). Secondly, if \( i \) is relevant for both \( g \) and \( g' \), the difference between \( f(g) \) and \( f(g') \) due to \( x_i \) and \( x'_i \) increases monotonically with increasing \( |x_i - x'_i| \). The parameter \( \rho_i \) controls whether differing in the relevance of \( i \) contributes more or less to the distance than differing in the value of \( x_i \), should \( i \) be relevant. Hyperparameter \( \omega_i \) defines a length scale for the \( i \)-th feature.

Note that so far we only have defined a kernel for dimension \( i \). To obtain a kernel for the entire \( D \)-dimensional input space, we simply embed each dimension in \( \mathbb{R}^2 \) using Equation (3) and then use the embedded input space of size \( 2D \) within any kernel that is defined in terms of Euclidean distance. We dub this new kernel the \emph{arc kernel}. Its parameters, \( \omega_i \) and \( \rho_i \) for each dimension, can be optimized using the GP marginal likelihood, or integrated out using Markov chain Monte Carlo.

3 Experiments

We now show that the arc kernel yields better results than other alternatives. We perform two types of experiments: first, we study model quality in isolation in a regression task; second, we demonstrate that the improved model indeed yields improved BO performance.

Data. We use two different datasets, both of which are common in the deep learning literature. The first is the canonical MNIST digits dataset [15] where the task is to classify handwritten digits. The second is the CIFAR-10 object recognition dataset [16]. We pre-processed CIFAR-10 by extracting features according to the pipeline given in [17].

3.1 Model Quality Experiments

Models. Our first experiments concern the quality of the regression models used to form the response surface for Bayesian optimization. We generated data by performing 10 independent runs of Bayesian optimization on MNIST and then treat this as a regression problem. We compare the GP with arc kernel (Arc GP) to several baselines: the first baseline is a simple linear regression model, the second is a GP using a Matérn 5/2 kernel. Irrelevant dimensions are simply filled in randomly for each input. We also compare to the case where each architecture gets its own separate model, as in [5]. The results are averaged over 10-fold train/test splits. Kernel parameters were inferred using slice sampling [18]. As the errors lie between 0 and 1 with many distributed toward the lower end, it can be beneficial to take the log of the outputs before modelling them with a GP. We experiment with both the original and transformed outputs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Original data</th>
<th>Log outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate Linear</td>
<td>0.812 ± 0.045</td>
<td>0.737 ± 0.049</td>
</tr>
<tr>
<td>Separate GP</td>
<td>0.546 ± 0.038</td>
<td>0.446 ± 0.041</td>
</tr>
<tr>
<td>Separate Arc GP</td>
<td>0.535 ± 0.030</td>
<td>0.440 ± 0.031</td>
</tr>
<tr>
<td>Linear</td>
<td>0.876 ± 0.043</td>
<td>0.834 ± 0.047</td>
</tr>
<tr>
<td>GP</td>
<td>0.481 ± 0.031</td>
<td>0.401 ± 0.028</td>
</tr>
<tr>
<td>Arc GP</td>
<td><strong>0.421 ± 0.033</strong></td>
<td><strong>0.335 ± 0.028</strong></td>
</tr>
</tbody>
</table>

Table 1: Normalized Mean Squared Error on MNIST Bayesian optimization data
3.2 Bayesian Optimization Experiments

In this experiment, we test the ability of Bayesian optimization to tune the hyperparameters of each layer of a deep neural network. We allow the neural networks for these problems to use up to 5 hidden layers (or no hidden layer). We optimize over learning rates, L2 weight constraints, dropout rates [19], and the number of hidden units per layer leading to a total of up to 23 hyperparameters and 6 architectures. On MNIST, most effort is spent improving the error by a fraction of a percent, therefore we optimize this dataset using the log-classification error. For CIFAR-10, we use classification error as the objective. We use the Deepnet2 package, and each function evaluation took approximately 1000 to 2000 seconds to run on NVIDIA GTX Titan GPUs.

Experimental Setup. For Bayesian optimization, we follow the methodology of [10], using slice sampling and the expected improvement heuristic. In this methodology, the acquisition function is optimized by first selecting from a pre-determined grid of points lying in $[0, 1]^{23}$, distributed according to a Sobol sequence. Our baseline is a standard Gaussian process that is agnostic to whether particular dimensions are irrelevant for a given point.

Results. Figure 2 shows that on these datasets, using the arc kernel consistently reaches good solutions faster than the naive baseline, or it finds a better solution. In the case of MNIST, the best discovered model achieved 1.19% test error using 50000 training examples. By comparison, [20] achieved 1.28% test error using a similar model and 60000 training examples. Similarly, our best model for CIFAR-10 achieved 21.1% test error using 45000 training examples and 400 features. For comparison, a support vector machine using 1600 features with the same feature pipeline and 50000 training examples achieves 22.1% error. Figure 2c shows the proportion of function evaluations spent on each architecture size for the CIFAR-10 experiments. Interestingly, the baseline tends to favour smaller models while a GP using the arc kernel distributes it’s efforts amongst deeper architectures that tend to yield better results.

4 Conclusion

We introduced the arc kernel for conditional parameter spaces that facilitates modelling the performance of deep neural network architectures by enabling the sharing of information across architectures where useful. Empirical results show that this kernel improves GP model quality and GP-based Bayesian optimization results over several simpler baseline kernels. Allowing information to be shared across architectures improves the efficiency of Bayesian optimization and removes the need to manually search for good architectures. The resulting models perform favourably compared to established benchmarks by domain experts.

5 Acknowledgements

The authors would like to thank Ryan P. Adams for helpful discussions.

---

2https://github.com/nitishsrivastava/deepnet
References


