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

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 (Brochu et al., 2010) 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 (Rasmussen & Williams, 2006)), as they are powerful and flexible models for which the marginal and conditional distributions can be computed efficiently.[1] 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 (Hutter, 2009; Bergstra et al., 2011). This is a general problem in algorithm configuration (Hutter, 2009) that occurs in many machine learning contexts, such as, e.g., in deep neural networks (Hinton et al., 2006); flexible computer vision architectures (Bergstra et al.); and the combined selection and hyperparameter optimization of machine learning algorithms (Thornton et al., 2013). We detail the case of deep neural networks below.

Bayesian optimization has recently been applied successfully to deep neural networks (Snoek et al., 2012; Bergstra et al., 2011) 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 (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012a), natural language processing (Mikolov et al., 2010) 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 of architecture or used non-GP models (Bergstra et al., 2011), or assumed a fixed architecture (Snoek et al., 2012). If there is any relation between networks with different architectures, separately modelling 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 the effectiveness of this kernel on a GP regression task and a Bayesian optimization task using a feed-forward classification neural network.

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

[1] 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 (Taddy et al., 2011; Hutter et al., 2011; Bergstra et al., 2011).

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

space that may have dimensions which are irrelevant under known conditions (further details are available in (Hutter & Osborne, 2013)). 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 $\delta_i \colon \mathcal{X} \to \{\text{true}, \text{false}\}$, for $i \in \{1, \ldots, D\}$. $\delta_i(\mathbf{x})$ stipulates the relevance of the ith feature x_i to $f(\mathbf{x})$.

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 $(x_1, \delta_2(\mathbf{x}), x_2)$ and assume that $\delta_1(\mathbf{x}) = \text{true}$; 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,

$$\begin{aligned} &k\big((x_1, \text{false}, x_2), (x_1', \text{false}, x_2')\big) \\ =&k\big((x_1, \text{false}, x_2''), (x_1', \text{false}, x_2''')\big),\ \forall x_2, x_2', x_2'', x_2''' ; \end{aligned} \tag{1}$$

- The covariance between a point using both parameters and a point using only one should again only depend on their shared parameters,

$$\begin{aligned} &k\big((x_1, \text{false}, x_2), (x_1', \text{true}, x_2')\big) \\ =&k\big((x_1, \text{false}, x_2''), (x_1', \text{true}, x_2''')\big),\ \forall x_2, x_2', x_2'', x_2''' . \end{aligned} \tag{2}$$

Put another way, in the absence of any other information, this specification encodes our prior ignorance about the irrelevant (missing) parameters while still allowing us to model correlations between relevant parameters.

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

$$g_i(\mathbf{x}) = \begin{cases} [0, 0]^\intercal & \text{if } \delta_i(\mathbf{x}) = \text{ false} \\ \omega_i [\sin \pi \rho_i \frac{x_i}{u_i - l_i}, \cos \pi \rho_i \frac{x_i}{u_i - l_i}]^\intercal & \text{otherwise.} \end{cases} \tag{3}$$

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

Figure 2.2 shows a visualization of the embedding of points $(x_1, \delta_2(\mathbf{x}), x_2)$ into $\mathbb{R}^3$. In this space, we have the Euclidean distance,

$$\begin{aligned} &d_i(\mathbf{x}, \mathbf{x}') = ||g_i(\mathbf{x}) - g_i(\mathbf{x}')||_2 \\ =&\begin{cases} 0 & \text{if } \delta_i(\mathbf{x}) = \delta_i(\mathbf{x}') = \text{false} \\ \omega_i & \text{if } \delta_i(\mathbf{x}) \neq \delta_i(\mathbf{x}') \\ \omega_i \sqrt{2} \sqrt{1 - \cos(\pi \rho_i \frac{x_i - x_i'}{u_i - l_i})} & \text{if } \delta_i(\mathbf{x}) = \delta_i(\mathbf{x}') = \text{true.} \end{cases} \end{aligned} \tag{4}$$

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 Δ 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 Δ, returning a valid covariance that satisfies all desiderata above.

Explicitly, note that as desired, if i is irrelevant for both $\mathbf{x}$ and $\mathbf{x}'$, d_i specifies that $g(\mathbf{x})$ and $g(\mathbf{x}')$ should not differ owing to differences between x_i and x_i'. Secondly, if i is relevant for both $\mathbf{x}$ and $\mathbf{x}'$, the difference between $f(\mathbf{x})$ and $f(\mathbf{x}')$ due to x_i and x_i' increases monotonically with increasing $|x_i - x_i'|$. The parameter ρ_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

Figure 1: A demonstration of the embedding giving rise to the pseudo-metric. All points for which $\delta_2(x) = \text{false}$ are mapped onto a line varying only along x_1. Points for which $\delta_2(x) = \text{true}$ are mapped to the surface of a semicylinder, depending on both x_1 and x_2. This embedding gives a constant distance between pairs of points which have differing values of δ but the same values of x_1.

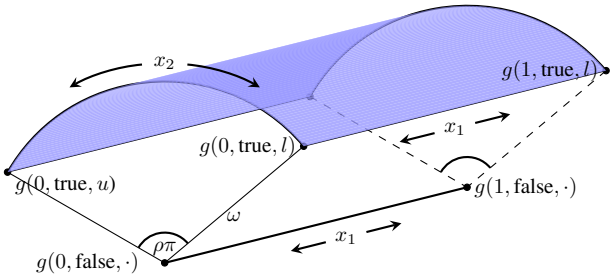
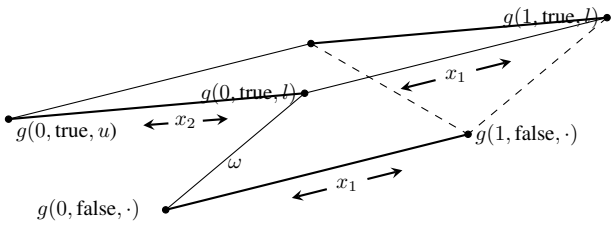


Figure 2: The box embedding (placeholder)



relevant. Hyperparameter ω_i defines a length scale for the ith 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. [Note from DD:This isn't quite accurate anymore, right Kev?] We dub this new kernel the *arc kernel*. Its parameters, ω_i and ρ_i for each dimension, can be optimized using the GP marginal likelihood, or integrated out using Markov chain Monte Carlo.

2.3 Box Kernel

When all the data have the same number of parameters, the box kernel has the original kernel as its special case.

3 Experiments

We now show that the arc kernel yields better results than other alternatives. Bayesian optimization requires building a surrogate model of the function being optimized, and better models can be expected to lead to more efficient optimization. However, because of the many interacting components of BO, optimizer performance might not correspond directly to the quality of the model. Thus, we perform two types of experiments: first, we study model quality in isolation in a regression task; second, we study the effect of the arc kernel on BO performance. All GP models use a Matérn $5/2$ kernel.

Data. We use two different datasets, both of which are common in the deep learning literature. The first is the canonical MNIST digits dataset (Lecun et al., 1998) where the task is to classify handwritten digits. The second is the CIFAR-10 object recognition dataset (Krizhevsky, 2009). We pre-processed CIFAR-10 by extracting features according to the pipeline given in (Coates et al., 2011).

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 where irrelevant dimensions are simply filled in randomly for each input. We also compare to the case where each architecture uses its own separate GP, as in (Bergstra et al., 2011). The results are averaged over 10-fold train/test splits. Kernel parameters were inferred using slice sampling (Murray & Adams, 2010). 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 1: Normalized Mean Squared Error on MNIST Bayesian optimization data

Method	Original data	Log outputs
Separate Linear	0.812 ± 0.045	0.737 ± 0.049
Separate GP	0.546 ± 0.038	0.446 ± 0.041
Separate Arc GP	0.535 ± 0.030	0.440 ± 0.031
Linear	0.876 ± 0.043	0.834 ± 0.047
GP	0.481 ± 0.031	0.401 ± 0.028
Arc GP	$\mathbf{0.421 \pm 0.033}$	$\mathbf{0.335 \pm 0.028}$

Results. Table 1 shows that a GP using the arc kernel performs favourably to a GP that ignores the relevance information of each point. The "separate" categories apply a different model to each layer and therefore do not take advantage of dependencies between layers. Interestingly, the separate Arc GP, which is effectively just a standard GP with additional embedding, performs comparably to a standard GP, suggesting that the embedding doesn't limit the expressiveness of the model.

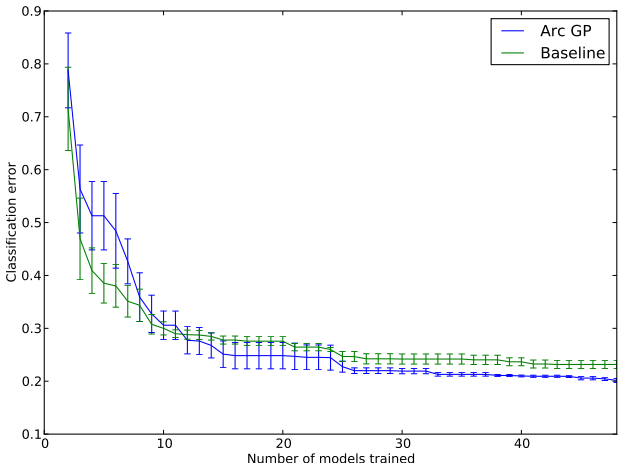
3.2 Bayesian Optimization Experiments

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 (Hinton et al., 2012b), 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 Deepnet[2] package, and each function evaluation took approximately 1000 to 2000 seconds to run on NVIDIA GTX Titan GPUs. Note that when a network of depth n is tested, all hyperparameters from layers $n+1$ onward are deemed irrelevant.

Experimental Setup. For Bayesian optimization, we follow the methodology of (Snoek et al., 2012), using slice

[2]https://github.com/nitishsrivastava/deepnet

(a) MNIST



(b) CIFAR-10

(c) Architectures searched

Figure 3: Bayesian optimization results using the arc kernel.

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 over this space that is agnostic to whether particular dimensions are irrelevant for a given point.

Results. Figure 3 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, (Wan et al., 2013) 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 3c 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

References

Bergstra, James, Yamins, Daniel, and Cox, David. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures.

Bergstra, James, Bardenet, Rémi, Bengio, Yoshua, Kégl, Balázs, et al. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, 2011.

Brochu, Eric, Brochu, Tyson, and de Freitas, Nando. A Bayesian interactive optimization approach to procedural animation design. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010.

Coates, Adam, Lee, Honglak, and Ng, Andrew Y. An analysis of single-layer networks in unsupervised feature learning. *Artificial Intelligence and Statistics*, 2011.

Hinton, Geoffrey E., Osindero, Simon, and Teh, Yee-Whye. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006. ISSN 0899-7667.

Hinton, Geoffrey E., Deng, Li, Yu, Dong, Dahl, George E., rahman Mohamed, Abdel, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N., and Kingsbury, Brian. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97, 2012a.

Hinton, Geoffrey E., Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012b.

Hutter, Frank. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University of British Columbia, Department of Computer Science, Vancouver, Canada, October 2009.

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

Hutter, Frank, Hoos, Holger H., and Leyton-Brown, Kevin. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, pp. 507–523, 2011.

Krizhevsky, Alex. Learning multiple layers of features from tiny images. *Technical report, Department of Computer Science, University of Toronto*, 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoff. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 2012.

Lecun, Yann, Bottou, Lon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. In *Proc. of the IEEE*, pp. 2278–2324, 1998.

Mikolov, Tomas, Karafiát, Martin, Burget, Lukas, Cernockỳ, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *Interspeech*, pp. 1045–1048, 2010.

Murray, Iain and Adams, Ryan P. Slice sampling covariance hyperparameters of latent Gaussian models. In *Advances in Neural Information Processing Systems*, 2010.

Rasmussen, Carl E. and Williams, Christopher K.I. Gaussian Processes for Machine Learning. *The MIT Press, Cambridge, MA, USA*, 2006.

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

Taddy, Matthew A., Gramacy, Robert B., and Polson, Nicholas G. Dynamic trees for learning and design. *Journal of the American Statistical Association*, 106 (493):109–123, 2011.

Thornton, Chris, Hutter, Frank, Hoos, Holger H., and Leyton-Brown, Kevin. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD'13*, pp. 847–855, 2013.

Wan, Li, Zeiler, Matthew, Zhang, Sixin, Cun, Yann L, and Fergus, Rob. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, 2013.