Training Deep Gaussian Processes with Sampling

Keyon Vafa
Columbia University
keyon.vafa@columbia.edu

Abstract

In this workshop paper, we explore deep Gaussian processes (deep GPs), a class of models for regression that combines Gaussian processes (GPs) with deep architectures. Exact inference on deep GPs is intractable, and while variational approximation methods have been proposed, these models are difficult to implement and do not extend easily to arbitrary kernels. We propose a stochastic gradient algorithm which relies on sampling to circumvent the intractability hurdle and uses pseudo data to ease the computational burden. To illustrate the model properties, we train various deep architectures on a noisy step-function and toy non-stationary data. By comparing the predictive distributions of each model, we show that deep GPs are well-suited to fit non-stationary functions.

1 Introduction

In recent years, Gaussian processes (GPs) have become one of the most popular nonparametric Bayesian models for supervised learning [10]. There are many advantages of using a probabilistic model like a GP for regression. For one, we can evaluate the marginal likelihood of the model, making model selection a numerically tractable task. Additionally, overfitting is seldom a problem due to the lack of parameters that need to be estimated [10]. Finally, by using an appropriate kernel, we can approximate a wide range of functions.

Under certain conditions, a GP can be viewed as a neural network with a single infinite-dimensional layer of hidden values [8]. Thus, it seems natural to combine the probabilistic advantages of GPs with the deep architectures of neural networks. Deep Gaussian processes (deep GPs) are intuitively an extension of GPs to multiple layers, and thus can be viewed as a deep neural network with multiple infinite-dimensional hidden layers [1].

2 Deep Gaussian Processes

Formally, we define a deep Gaussian Process as the distribution over functions constructed by composing GPs. We assume that each function is drawn independently, and that data is generated by the following process [4]:

$$\boldsymbol{f}^{(1:L)}(\boldsymbol{x}) = \boldsymbol{f}^{(L)}(\boldsymbol{f}^{(L-1)}(\dots \boldsymbol{f}^{(2)}(\boldsymbol{f}^{(1)}(\boldsymbol{x}))\dots))$$
$$\text{where } f_d^{(l)} \sim \mathcal{GP}\left(0, k_d^{(l)}(\boldsymbol{x}, \boldsymbol{x}')\right) \text{ for } f_d^{(l)} \in \boldsymbol{f}^{(l)}.$$

As a concrete example, we focus on the simplest type of deep GP, with a one-dimensional input, x_n, a one-dimensional hidden unit, h_n, and a one-dimensional output, y_n. This two-layer network consists of two GPs, f and g. Specifically, $h_n = f(x_n)$ and $y_n = g(h_n)$, where $f \sim \mathcal{GP}(0, k^{(1)}(x, x'))$ and $g \sim \mathcal{GP}(0, k^{(2)}(h, h'))$. Because each GP is allowed to have different hyperparameters, note that the covariance matrices correspond to different kernels. The graphical representation of this model is depicted in Figure 1.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

$$\begin{tikzcd} x_n \arrow[r, "f"] & h_n \arrow[r, "g"] & y_n \end{tikzcd}$$

Figure 1: A graphical representation of a simple two-layer deep GP where every layer has one dimension.

One advantage of using a deep GP prior over a standard GP is that deep GPs can model non-stationary functions without the use of a non-stationary kernel. If the shape of a function changes along the input space, a standard GP would require a non-stationary kernel to capture these differences. However, because a deep GP is the composition of GPs with different kernel parameters, it can model non-stationary functions.

3 Inference

Ideally, a Bayesian treatment of a deep GP would allow us to integrate out the hidden function values to evaluate $P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})$. Unfortunately, this computation, which involves the integrals of Gaussians with respect to kernel functions, is intractable [3].

We introduce a stochastic gradient descent algorithm, which trains a deep GP by relying on two central ideas. First, because we cannot integrate out hidden function values, we rely on sampling predictive means and covariances to approximate the marginal likelihood. Using these samples, we use automatic differentiation techniques and the reparameterization trick from Kingma et al. [7] to evaluate the gradients and optimize our objective.

Second, if we fit every GP in our model without any approximations, the computational complexity for L layers and H hidden units per layer is $\mathcal{O}(N^3LH)$. To ease the computational burden of our model, we use the Fully Independent Training Conditional (FITC) approximation, which incorporates training samples that may not be in the original data, known as pseudo data [11]. We introduce M pseudo inputs $\bar{\boldsymbol{X}} = \{\bar{\boldsymbol{x}}_m\}_{m=1}^M$ and the corresponding pseudo outputs $\bar{\boldsymbol{y}} = \{\bar{y}_m\}_{m=1}^M$, to every GP in the cascade. We assume that, conditioning on the pseudo inputs and outputs, every output value y_n is generated from the corresponding input $\boldsymbol{x}_n$ by conditioning a multivariate normal on the pseudo data:

$$P(y_n|\boldsymbol{x}_n, \bar{\boldsymbol{X}}, \bar{\boldsymbol{y}}) = \mathcal{N}(\boldsymbol{k}_{\boldsymbol{x}_n}^T \boldsymbol{K}_{\bar{\boldsymbol{x}}\bar{\boldsymbol{x}}}^{-1} \bar{\boldsymbol{y}}, k_{\boldsymbol{x}_n} - \boldsymbol{k}_{\boldsymbol{x}_n}^T \boldsymbol{K}_{\bar{\boldsymbol{x}}_n\bar{\boldsymbol{x}}_n}^{-1} \boldsymbol{k}_{\boldsymbol{x}_n} + \sigma^2) \tag{1}$$

for $k_{\boldsymbol{x}_n} = k(\boldsymbol{x}_n, \boldsymbol{x}_n), \boldsymbol{k}_{\boldsymbol{x}_n} = [k(\boldsymbol{x}_n, \bar{\boldsymbol{x}}_1), \ldots, k(\boldsymbol{x}_n, \bar{\boldsymbol{x}}_m)]$, and $\boldsymbol{K}_{\bar{\boldsymbol{x}}\bar{\boldsymbol{x}}}$ the kernel matrix evaluated at our pseudo inputs [11]. The key assumption is that, given the pseudo data, each component of y_i is generated independently. The $\mathcal{O}(N^3)$ matrix inversion at every GP is now removed, so our overall complexity becomes $\mathcal{O}(N^2MLH)$ where $M \ll N$. We add a normal prior on the location of the pseudo outputs: $P(\bar{\boldsymbol{y}}|\bar{\boldsymbol{X}}) = \mathcal{N}(\boldsymbol{0}, K_{\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}})$, and learn the pseudo parameters at every layer.

As an example, consider a single-dimensional 2-layer deep GP, as in Figure 1. We would like to learn $\{(\bar{\boldsymbol{X}}^{(l)}, \bar{\boldsymbol{y}}^{(l)})\}_{l=1}^2$, the pseudo data for each layer, along with the kernel parameters for f and g. To evaluate the log-likelihood at our current parameter estimates, we first sample values from the hidden layer, $\boldsymbol{H}$, given our inputs, $\boldsymbol{X}$. We use the FITC approximation, so

$$\boldsymbol{H} \sim \mathcal{N}\left(\boldsymbol{K}_{\boldsymbol{X}\bar{\boldsymbol{X}}^{(1)}} \boldsymbol{K}_{\bar{\boldsymbol{X}}^{(1)}\bar{\boldsymbol{X}}^{(1)}}^{-1} \bar{\boldsymbol{y}}^{(1)}, \text{diag}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} - \boldsymbol{K}_{\boldsymbol{X}\bar{\boldsymbol{X}}^{(1)}} \boldsymbol{K}_{\bar{\boldsymbol{X}}^{(1)}\bar{\boldsymbol{X}}^{(1)}}^{-1} \boldsymbol{K}_{\bar{\boldsymbol{X}}^{(1)}\boldsymbol{X}}\right)\right).$$

After obtaining J samples of the hidden layer values, we use each sample $\tilde{\boldsymbol{H}}^j$ to approximate the marginal log-likelihood (note that this departs from standard applications of the reparameterization trick, where one sample is often used). After incorporating the priors over the pseudo outputs, we have:

$$\log P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\Theta}) \approx \frac{1}{J} \sum_{j=1}^{J} \log P\left(\boldsymbol{y}|\tilde{\boldsymbol{H}}^j, \bar{\boldsymbol{X}}^{(2)}, \bar{\boldsymbol{y}}^{(2)}\right) + \sum_{l=1}^{2} \log P\left(\bar{\boldsymbol{y}}^{(l)}|\bar{\boldsymbol{X}}^{(l)}\right). \tag{2}$$

Finally, we can use automatic differentiation to evaluate gradients. Thus, by maximizing the above function with respect to $\boldsymbol{\Theta}$, we can learn the model parameters.

The advantages of this algorithm over existing methods to train deep GPs are that it allows for automatic differentiation, it can extend to arbitrary kernels, and it is comparatively straightforward.

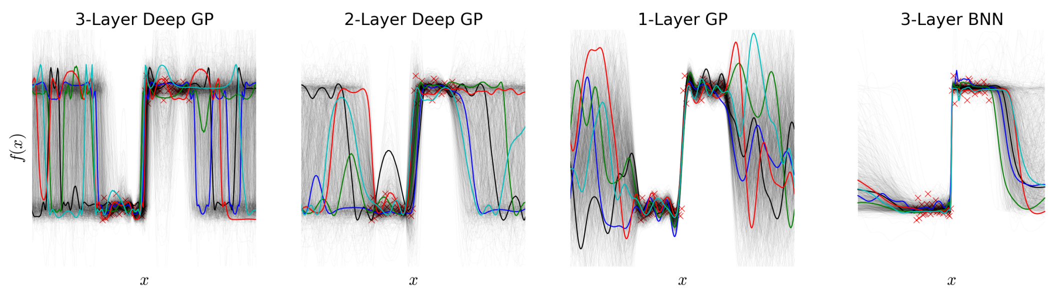

Figure 2: For a noisy step function with $N = 60$ data points, we train a 2- and 3-layer deep GP with 1 hidden unit and 10 pseudo parameters for each function, a standard GP for regression, and a 3-layer BNN with 20 hidden units in each hidden layer and a radial nonlinearity. The red 'x's denote the data points, and the black curves denote 2000 posterior draws for each model, 5 of which are randomly colored for emphasis.

Damianou and Lawrence [3] also use pseudo parameters at every layer to fit a deep GP, but rather than learning all the pseudo outputs, they perform inference using approximate variational marginalization and integrate them out. Subsequent methods implemented to train deep GPs for regression [6, 2, 1] also use variational approximations. While the variational methods benefit from being able to integrate out the pseudo outputs, they rely on integral approximations that depend on the particular kernel being used [6, 1]. Thus, they do not extend easily to arbitrary kernels or to those that are hard to integrate, such as the Matérn kernel. As a result, our stochastic gradient algorithm has a "black-box" nature, since it can extend easily to most kernels by using automatic differentiation. This also has a cost; much can depend on the choice of kernel, which can affect the variance of the gradients. Additionally, depending on the architecture, optimization can be prone to local optima.

4 Experiments

4.1 Step Function

We illustrate the strengths and limitations of our proposed algorithm by training a deep GP on noisy step function data. The non-stationarity of a step function implies that a deep GP should perform better than a standard GP for regression [6]. For a step function with $N = 60$ data points, we train 2- and 3-layer deep GPs, each with single-dimensional hidden layers, 10 pseudo parameters per function, and the squared exponential kernel. In addition, we compare models by training a Bayesian Neural Network (BNN) with black-box stochastic variational inference (BBSVI) [9, 5], with 2 hidden layers and 20 hidden units in each layer with a radial basis nonlinearity.

Figure 2 illustrates the experimental results. Qualitatively, we observe that the predictive distribution for the single-layer GP is unable to capture the smoothness at both steps of the function. By comparison, the predictive draws for the 2- and 3-layer deep GPs are flatter at each step. The deep GPs are able to capture these fluctuations by composing multiple GPs with varying kernel parameters: For example, the recovered length-scale parameter for the single-layer GP is $l^2 = .327$, compared to .539 and .222 for the two functions (in that order) composed for the 2-layer deep GP. Additionally, compared to the 2-layer deep GP, the 3-layer model has a steeper incline between steps and puts more weight on the steps outside the input region.

It appears that the posterior draws for a BNN are flatter at the steps than for a deep GP, and the incline between steps is steeper. However, the draws for the BNN outside of the input range are not as concentrated at the steps, which does not match our intuition on the uncertainty. Additionally, the BNN depicted in Figure 2 is a more complex model than the deep GPs we trained, as there are 20 hidden units per layer (as opposed to 1). When we trained a 3-layer BNN with one hidden unit, the posterior draws provided poorer approximations to the data and displayed little variance. As a result, in the case of a noisy step function, a deep GP appears to be better suited to capture uncertainty.

To empirically test the success of our models for the noisy step function, we run experiments comparing prediction quality for 1-, 2-, and 3-layer deep GPs (where each hidden layer has one unit)

Number of Data Points	1 Layer	2 Layers	3 Layers
50	.44	$-.03$	**1.27**
100	.23	**.54**	$-.36$
200	$-.11$	.71	**.80**

Figure 3: The test log-likelihood per data corresponding to the highest training log-likelihood for each combination of layers and data points in the step function experiment.

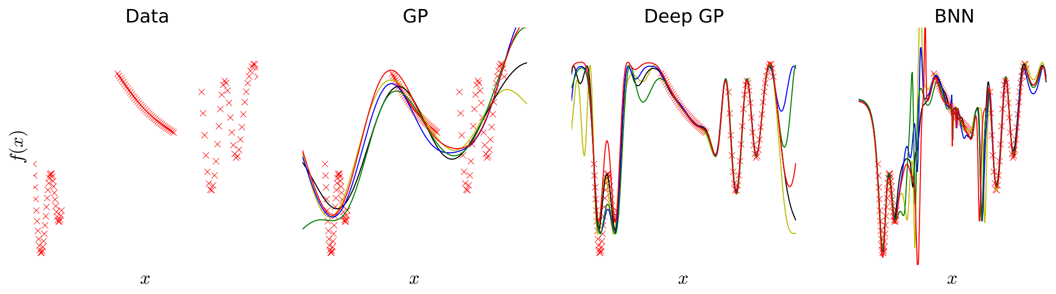

Figure 4: Predictive draws for toy non-stationary data from a standard GP, a 2-layer deep GP with one hidden unit, and a 3-layer BNN with 20 hidden units per layer.

with varying sizes of our data set: $N = 50, 100$, and 200. We train on 80% of the data and test on the remaining 20%, using 10 pseudo data points for each model. We discovered that deeper models were prone to poor local optima, so we performed 10 random restarts for each setting, using the model with the largest training log-likelihood for prediction. Our results are displayed in Figure 3. These results appear to coincide with the notion that, when successfully trained, deeper models are better suited to fit the step function.

4.2 Toy Non-Stationary Data

To evaluate a deep GP's ability to learn non-stationary functions, we create toy non-stationary data and fit it with various architectures. Specifically, we create data by dividing the input space into three regions, and sample outputs from a GP using varying hyper-parameters for each region. In Figure 4, we use a single-layer GP, a 2-layer deep GP with one hidden unit, and a 3-layer BNN with 20 hidden units per layer to fit the data. The deep GP and BNN yield predictive draws that are highly-varying in the outside region and flatter in the intermediate region, compared to the single-layer GP which has a constant curvature. Compared to the deep GP, the BNN appears to fit the data perfectly, although the uncertainty is more extreme in the transitions between regions and much smaller at the tails.

5 Conclusion

Deep GPs, which combine the deep architecture of BNNs with the probabilistic properties of standard GPs, are a powerful class of deep Bayesian models. While our proposed algorithm is a relatively simple inference technique and can extend to arbitrary kernels, it is difficult to optimize, as optimization over deeper models can result in local minima that do not provide good fits to the data. When deeper models successfully train, they achieve a higher predictive performance than shallower models, so it will be beneficial to explore methods that make inference for deep GPs more feasible.

Acknowledgments

This work would have been impossible without the generous help and support of David Duvenaud, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Alexander Rush. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-16-44869.

References

[1] Thang D Bui, Daniel Hernández-Lobato, Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. Deep Gaussian processes for regression using approximate expectation propagation. *33rd International Conference on Machine Learning*, 2016.

[2] Zhenwen Dai, Andreas Damianou, Javier González, and Neil Lawrence. Variational auto-encoded deep Gaussian processes. *International Conference on Learning Representations*, 2016.

[3] Andreas C Damianou and Neil D Lawrence. Deep Gaussian processes. In *16th International Workshop on Artificial Intelligence and Statistics*, 2013.

[4] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[5] David Duvenaud and Ryan P Adams. Black-box stochastic variational inference in five lines of Python. 2015.

[6] James Hensman and Neil D Lawrence. Nested variational compression in deep Gaussian processes. *arXiv preprint arXiv:1412.1370*, 2014.

[7] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems 28*, 2015.

[8] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[9] Rajesh Ranganath, Sean Gerrish, and David M Blei. Black box variational inference. In *Proceedings of the Seventeenth International Workshop on Artificial Intelligence and Statistics*, 2014.

[10] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[11] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, 2005.