Faster Gaussian Processes via Deep Embeddings

Constantinos Daskalakis [1] **Petros Dellaportas** [2 3] **Aristeidis Panos** [2]

Abstract

Gaussian processes provide a probabilistic framework for quantifying uncertainty of prediction and have been adopted in many applications in Statistics and Bayesian optimization. Unfortunately, they are hard to scale to large datasets as they necessitate inverting matrices whose size is linear in the number of observations. Moreover, they necessitate an a priori chosen functional form for their kernels with predetermined features. Our contribution is a framework that addresses both challenges. We use deep neural networks for automatic feature extraction, combined with explicit functional forms for the eigenspectrum of Gaussian processes with Gaussian kernels, to derive a Gaussian process inference and prediction framework whose complexity scales linearly in the number of observations and which accommodates automatic feature extraction. On a series of datasets, our method outperforms state of the art scalable Gaussian process approximations.

1. Introduction

Gaussian processes have long been studied in probability and statistics. In Bayesian inference, they provide a canonical way to define a probability distribution over functions, which can be used as a prior to build probabilistic frameworks for quantifying uncertainty in prediction. Among many applications, they have been a method of choice for hyperparameter tuning in deep learning.

For an introduction to Gaussian processes, their use in Bayesian Inference, and some of their many applications, see e.g. (Rasmussen & Williams, 2006; Liu et al., 2011). In the simplest setting, a probability distribution over functions $f : \mathbf{x} \mapsto y$ is defined as follows. Given values $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ for the independent variables, it is assumed that the dependent variables $\mathbf{y} = \{y_1, \ldots, y_N\}$ are jointly Gaussian, with mean vector $\boldsymbol{\mu} = (\mu(\mathbf{x}_i))_i$ and covariance matrix $K(X, X) = (k(\mathbf{x}_i, \mathbf{x}_j))_{ij}$, where $\mu(\cdot)$ is some function, and $k(\cdot, \cdot)$ is a positive semidefinite kernel. We often assume that we do not observe the Gaussian sample directly but additional noise is added to it prior to our observation.

Given this framework for defining distributions over functions, in Bayesian inference, we assume that we are given the value of the dependent variables on a subset $\mathcal{I} \subset \{1, \ldots, N\}$ of indices, and the goal is to compute the posterior distribution of the dependent variables over the remaining indices $\overline{\mathcal{I}}$. Oftentimes, $\mu(\cdot)$ and $k(\cdot, \cdot)$ are parametric functions, and the noise added to the Gaussian sample prior to our observation is also sampled from a parametric distribution, and the goal is to estimate the parameters of $\mu(\cdot)$, $k(\cdot, \cdot)$ and the noise distribution as well as to compute the posterior distribution.

While flexible and widely used, the aforedescribed framework has important limitations. Even when $\mu(\cdot)$ and $k(\cdot, \cdot)$ are assumed perfectly known, and even when there is no noise, computing the posterior distribution of the values $Y_{\overline{\mathcal{I}}}$ requires inverting matrices whose size is linear in $|\mathcal{I}|$, the number of observed values. Relatedly, when $\mu(\cdot)$, $k(\cdot, \cdot)$ or the noise distribution are parametric, estimating their parameters requires writing down the likelihood of Y_I, which also involves the inverse of the covariance matrix $K(X_{\mathcal{I}}, X_{\mathcal{I}})$. All in all, the emergence of inverse covariance matrices for both model parameter estimation and prediction makes the framework hard to scale computationally beyond a few thousand observations.

Given the aforedescribed computational limitations various approximation approaches have been proposed aiming at circumventing the inversion of large or dense covariance matrices, or performing this inversion approximately. A recent survey of scalable GP-based Bayesian Inference frameworks is provided in (Liu et al., 2020). These methods are roughly clustered into global approximations, which do approximate inference by distilling the entire dataset, and local approximations, which focus on subsets of data close

[1]Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA [2]Department of Statistical Science, University College London [3]Department of Statistics, Athens University of Economics and Business. Correspondence to: Aristeidis Panos <aristeidis.panos.15@ucl.ac.uk>, Petros Dellaportas <p.dellaportas@ucl.ac.uk>, Constantinos Daskalakis <costis@csail.mit.edu>.

to where predictions are to be made. Examples of global approximations include keeping a representative subset of $M \ll N$ training points resulting in a smaller kernel matrix (see e.g. Chalupka et al. (2013)), approximating the kernel matrix via a sparse matrix (see e.g. Gneiting (2002)), or performing low-rank approximations to the kernel matrix using inducing points (see e.g. Quiñonero-Candela & Rasmussen (2005); Titsias (2009); Hensman et al. (2013); Wilson & Nickisch (2015)). Local approximation methods include mixture of experts methods, which conduct model averaging from multiple experts built on local subsets of the data to boost predictions; see e.g. (Hinton, 2002; Rasmussen & Ghahramani, 2002; Gramacy & Lee, 2008; Sun & Xu, 2010; Yuksel et al., 2012; Masoudnia & Ebrahimpour, 2014; Deisenroth & Ng, 2015; Gramacy et al., 2016; Rullière et al., 2018; Liu et al., 2018).

A second limitation facing the vanilla application of Gaussian processes in Bayesian inference is the necessity to commit a priori to a specific functional form for the kernel, such as the Gaussian, Exponential, and Matérn kernels. Commonly used kernels such as these have a few tunable parameters but they use an a priori chosen distance function $d(\cdot, \cdot)$ measuring the affinity between pairs of $\mathbf{x}_i$ and $\mathbf{x}_j$. As such, standard Gaussian Process-based Bayesian Inference does not directly permit learning relevant features as part of the inference process.

Motivated by this limitation of the standard framework, several recent works have proposed combining the expressive power of deep neural networks (DNNs) with the uncertainty quantification power of Gaussian Processes. Quite naturally, these works propose mapping the original input space to some latent space using a neural network, then using a Gaussian Process over the latent space for prediction; see e.g. (Wilson et al., 2016a;b; Calandra et al., 2016; Al-Shedivat et al., 2017; Bradshaw et al., 2017). See also (Cremanns & Roos, 2017; Iwata & Ghahramani, 2017) for variants of this idea. In these works, the parameters of both the Neural Network used for embedding to latent space, and the parameters of the Gaussian process are jointly estimated by maximizing likelihood. Unfortunately, the resulting optimization problem faces a stronger version of the computational challenges facing the vanilla GP-based Bayesian inference mentioned earlier, as the likelihood function involves the inverse of the covariance matrix, which is now the composition of the kernel with the neural network. Backpropagation through the resulting function becomes even more challenging than in the vanilla setting.

The contribution of this work is a Gaussian process-based inference framework that addresses both limitations of the standard framework discussed above. Addressing the second limitation, we use DNNs to map the input space to some latent space, defining a Gaussian Process on the latent space, as proposed in the aforedescribed recent works. In contrast to prior work, however, we propose an approach for training the weights of the Neural Network and the parameters of the Gaussian Process, as well as for making predictions, which circumvents the need for inverting (for prediction) and backpropagating through (for training) inverses of large covariance matrices.

The main idea behind our framework is quite simple. Recall that the Gaussian kernel (in one- or multiple dimensions) has an infinite Mercer expansion in terms of explicitly known eigenfunctions involving Hermite polynomials. These eigenfunctions are orthonormal with respect to the inner product defined by an appropriately chosen Gaussian measure, and their corresponding eigenvalues are also known explicitly, and decrease exponentially fast in common regimes of parameters. As such, the kernel is well approximated by keeping the first few terms of its Mercer expansion.

How can we exploit these facts to speed up DNN+Gaussian process-based inference? We propose the following natural approach. First, we hypothesize that there is a DNN-expressible feature extraction map $g_w : \mathbf{x} \mapsto \mathbf{z}$, which maps the (potentially non-Euclidean and potentially high-dimensional) inputs $\mathbf{x}$ to low-dimensional Euclidean latent vectors $\mathbf{z} = g_w(\mathbf{x}) \in \Re^d$ such that the distribution over $\mathbf{x}$'s maps to a Gaussian distribution over $\mathbf{z}$'s. This means that a large sample $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of $\mathbf{x}$'s will map to a sample $Z = \{\mathbf{z}_1 \equiv g_w(\mathbf{x}_1), \ldots, \mathbf{z}_N \equiv g_w(\mathbf{x}_N)\}$ of latent vectors whose empirical distribution converges in distribution to a Gaussian distribution over $\Re^d$. In turn, this means that the vectors defined by evaluating on Z the eigenfunctions of the Mercer expansion of the Gaussian kernel will be approximately orthogonal, and that the covariance matrix $K(Z, Z) = (k(\mathbf{z}_i, \mathbf{z}_j))_{ij}$ defined by evaluating the Gaussian kernel on pairs of samples from Z, will be well-approximated by the sum of eigenvalue-scaled outer-products of the vectors computed by evaluating on Z the top few eigenfunctions of the Mercer expansion. This provides an analytically computed low-rank approximation to the covariance matrix $K(Z, Z)$. We use the matrix inversion lemma to obtain an explicit form for its inverse, using this explicit form in our maximum likelihood objective to train the parameters of the Neural Network, the Gaussian kernel, and the noise distribution, as well as to make predictions after training. As we have already noted, the eigenvalues in the Mercer expansion decrease exponentially fast in common regimes of parameters, so we may keep a small number ℓ of eigenfunctions from the Mercer expansion to obtain a good approximation to the Gaussian kernel, as described above. It follows that computing the likelihood function requires inverting a single $\ell \times \ell$ matrix, and computing matrix-vector products between $\ell \times N$ matrices and N dimensional vectors or $\ell \times \ell$ matrices and ℓ dimensional vectors, i.e. total time linear in N when ℓ is a constant. Using a trained model

for making a prediction similarly takes time linear in N.

2. Methodology

2.1. Gaussian process basics

Assume that we have a data vector $\mathbf{y}$ with entries y_i that are noisy observations of the function $f(\mathbf{x}_i)$ for all D-dimensional vectors in $X = \{\mathbf{x}_i\}_{i=1}^N$. We take the noise for each data entry to be independent Gaussian with variance σ^2. We place a Gaussian process prior with mean function $\mu(\cdot)$ and covariance kernel $k_\theta(\cdot,\cdot)$ over $f(.)$, so that the collection of function values $f(X)$ has a joint Gaussian distribution

$$[f(\mathbf{x}_1),\ldots,f(\mathbf{x}_N)]^T \sim \mathcal{N}(\boldsymbol{\mu}, K(X,X))$$

where $\boldsymbol{\mu}_i = \mu(\mathbf{x}_i)$ and $K(X,X)_{ij} = k_\theta(\mathbf{x}_i,\mathbf{x}_j)$. We set $A = K(X,X) + \sigma^2 I_N$. The log-marginal likelihood of the data is given by

$$\log p(\mathbf{y}|X) = -\frac{1}{2}(\mathbf{y}-\mu(X))^T A^{-1}(\mathbf{y}-\mu(X)) - \frac{1}{2}\log|A| - \frac{N}{2}\log(2\pi)$$

and the future observations y^* with covariates X^* have a normal conditional distribution with mean and variance given by

$$E(y^*|y) = \mu(X^*) + K(X^*,X)A^{-1}(y-\mu(X)) \quad (1)$$

$$V(y^*|y) = K(X^*,X^*) - K(X^*,X)A^{-1}K(X,X^*). \quad (2)$$

2.2. Deep embeddings meet Gaussian processes

In this section, we add a twist to the vanilla Gaussian process model presented in the previous section, in line with prior work combining DNNs and Gaussian processes (Wilson et al., 2016a;b; Calandra et al., 2016; Al-Shedivat et al., 2017; Bradshaw et al., 2017). We define a prior distribution over functions $f : \mathbf{x} \mapsto \mathbf{y}$ that results from the composition of a random function with a deterministic function, as follows. First, a deterministic function $g_w : \mathbf{x} \mapsto \mathbf{z}$ embeds D-dimensional input vectors $\mathbf{x}$ to d-dimensional vectors $\mathbf{z} = (z^1,\ldots,z^d)$; we assume that g_w is parametric, e.g. expressible by a DNN, and our inference framework will aim at inferring its parameters w. Next, a random function $h : \mathbf{z} \mapsto \mathbf{y}$ is sampled from a Gaussian process with noisy observations, as described in the previous subsection. The Gaussian process uses a squared exponential (a.k.a. Gaussian) kernel:

$$k_{\sigma_f^2,\Delta}(\mathbf{z}_i,\mathbf{z}_j) = \sigma_f^2 \exp(-0.5(\mathbf{z}_i-\mathbf{z}_j)^T\Delta(\mathbf{z}_i-\mathbf{z}_j))$$

with $\Delta = \mathrm{diag}(\epsilon_1^2,\ldots,\epsilon_d^2)$ comprising the length scales along d input dimensions and σ_f^2 being the signal variance.

For reasons that will become apparent shortly we will also make the following modeling assumption. We will assume that, with respect to the randomness of $\mathbf{x}$, the distribution of $\mathbf{z} = g_w(\mathbf{x})$ satisfies $z^j \sim N(0, 1/2\alpha_j^2)$, for each coordinate $j = 1,\ldots,d$, and some $\alpha_j > 0$.

2.3. Our inference framework

Our main contribution is a linear-time approximate inference and prediction framework for the model outlined in the previous section. Our starting point is Mercer's expansion (Mercer, 1909) of the Gaussian kernel:

$$k_{\sigma_f^2,\Delta}(\mathbf{z}_i,\mathbf{z}_j) = \sum_{\mathbf{n}\in\mathbb{N}^d} \lambda_{\mathbf{n}}\phi_{\mathbf{n}}(\mathbf{z}_i)\phi_{\mathbf{n}}(\mathbf{z}_j). \quad (3)$$

We take the $\phi_{\mathbf{n}}$'s to be an orthonormal basis of $L_2(\Re^d,\rho)$, where inner products of functions are computed with respect to the Gaussian measure, $\rho(\mathbf{z})$, with independent coordinates sampled by

$$\rho_j(z^j) = \alpha_j\pi^{-1/2}exp(-\alpha_j^2(z^j)^2),\ \forall j = 1,\ldots,d, \quad (4)$$

for the same α_j's from the previous section. It is standard (see e.g. (Zhu et al., 1997b; Rasmussen & Williams, 2006; Fasshauer, 2012; Fasshauer & McCourt, 2012)) that the orthonormal basis $(\phi_{\mathbf{n}})_{\mathbf{n}\in\mathbb{N}^d}$ can be constructed as a tensor product of the orthonormal bases of $L_2(\Re^d,\rho_j)$ for all j, as follows. Setting

$$\beta_j = \left(1+(2\epsilon_j/\alpha_j)^2\right)^{1/4},\ \ \gamma_{n_j} = \sqrt{\frac{\beta_j}{2^{n_j-1}\Gamma(n_j)}},$$

$$\delta_j^2 = \frac{\alpha_j^2}{2}(\beta_j^2-1)$$

the orthonormal eigenvectors are defined as follows

$$\phi_{\mathbf{n}}(\mathbf{z}) = \sigma_f\prod_{j=1}^d \phi_{n_j}(z^j) = \prod_{j=1}^d \left\{\gamma_{n_j}exp(-\delta_j^2(z^j)^2)H_{n_j-1}(\alpha_j\beta_j z^j)\right\}, \quad (5)$$

where H_n are the Hermite polynomials of degree n; and the corresponding eigenvalues are

$$\lambda_{\mathbf{n}} = \prod_{j=1}^d \lambda_{n_j} = \prod_{j=1}^d \left\{\left(\frac{\alpha_j^2}{\alpha_j^2+\delta_j^2+\epsilon_j^2}\right)^{1/2}\left(\frac{\epsilon_j^2}{\alpha_j^2+\delta_j^2+\epsilon_j^2}\right)^{n_j-1}\right\}. \quad (6)$$

Note that $\lambda_{n_j} \to 0$ as $n_j \to \infty$. Indeed, as long as α_j^2/ϵ_j^2 is bounded away from 0, this decay is exponentially fast.

Next, our modeling assumption from the previous section that, with respect to the randomness of $\mathbf{x}$, the distribution of $\mathbf{z} = g_w(\mathbf{x})$ satisfies $z^j \sim N(0, 1/2\alpha_j^2)$, for each coordinate $j = 1, \ldots, d$, becomes crucial. It implies that a large sample $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of $\mathbf{x}$'s will map to a sample $Z = \{\mathbf{z}_1 \equiv g_w(\mathbf{x}_1), \ldots, \mathbf{z}_N \equiv g_w(\mathbf{x}_N)\}$ of vectors whose empirical distribution converges in distribution to $\rho(\mathbf{z})$, as $N \to \infty$.

In turn, this means that the family of vectors $(\xi_{\mathbf{n}})_{\mathbf{n}\in\mathbb{N}^d}$ in $\Re^N$, defined as follows:

$$\xi_{\mathbf{n}} = \phi_{\mathbf{n}}(Z) \equiv (\phi_{\mathbf{n}}(\mathbf{z}_1), \phi_{\mathbf{n}}(\mathbf{z}_2), \ldots, \phi_{\mathbf{n}}(\mathbf{z}_N))$$

satisfy

$$\xi_{\mathbf{n}}^{\mathrm{T}} \cdot \xi_{\mathbf{n}'} \to \begin{cases} 0, \text{if } \mathbf{n} \neq \mathbf{n}' \\ 1, \text{otherwise} \end{cases},$$

as $N \to \infty$, i.e. "the vectors $(\xi_{\mathbf{n}})_{\mathbf{n}\in\mathbb{N}^d}$ become more and more orthonormal" as $N \to \infty$.

Note that, it follows from Equation (3) and the definition of the vectors $(\xi_{\mathbf{n}})_{\mathbf{n}}$ that the kernel matrix $K(Z, Z) = (k_{\sigma_f^2,\Delta}(\mathbf{z}_i, \mathbf{z}_j))_{ij}$ satisfies

$$K(Z, Z) \equiv \sum_{\mathbf{n}\in\mathbb{N}^d} \lambda_{\mathbf{n}} \xi_{\mathbf{n}} \xi_{\mathbf{n}}^{\mathrm{T}}. \tag{7}$$

It follows from (i) Eq. (7), (ii) the approximate orthonormality of the vectors in $(\xi_{\mathbf{n}})_{\mathbf{n}}$ as $N \to \infty$, and (iii) the fact that $\lambda_{n_j} \to 0$ as $n_j \to \infty$ exponentially fast, as long as α_j^2/ϵ_j^2 is bounded away from 0, that $K(Z, Z)$ is well-approximated by the sum of the first few terms of (7), i.e. for some m:

$$K(Z, Z) \approx \sum_{\mathbf{n}\in\mathbb{N}^d, \mathbf{n}\leq(m,m,\ldots,m)} \lambda_{\mathbf{n}} \xi_{\mathbf{n}} \xi_{\mathbf{n}}^{\mathrm{T}}. \tag{8}$$

Compactly, we represent the right hand side of (8) by $\Xi\Lambda\Xi^T$, where Ξ is the $N \times m^d$ matrix whose rows are the vectors $\xi_{\mathbf{n}}$, for $\mathbf{n} \leq (m, \ldots, m)$ and Λ is the $m^d \times m^d$ diagonal matrix with the corresponding eigenvalues. As such, we have an analytically computed low-rank approximation to the kernel matrix $K(Z, Z)$.

The resulting approximation to the covariance matrix of the Gaussian process is $B = \Xi\Lambda\Xi^T + \sigma^2 I_N$. Next, we use the Woodbury matrix inversion lemma and the Sylvester determinant theorem to obtain an explicit form for the inverse of B and its determinant:

$$B^{-1} = \sigma^{-2} I_N - \sigma^{-2}\Xi(\sigma^2\Lambda^{-1} + \Xi^T\Xi)^{-1}\Xi^T;$$
$$|B| = \sigma^{2N}|\Lambda||\Lambda^{-1} + \sigma^{-2}\Xi^T\Xi|.$$

Note that the right hand sides of both identities involve the inversion of an $m^d \times m^d$ matrix. In turn, these identities can be plugged directly into the expressions for the log-marginal likelihood, and mean and variance of the predictive density of future observations $\mathbf{y}^*$:

$$\log p(\mathbf{y}|Z) \approx -\frac{1}{2}(\mathbf{y} - \mu(Z))^T B^{-1}(\mathbf{y} - \mu(Z)) - \frac{1}{2}\log|B| - \frac{N}{2}\log(2\pi) \tag{9}$$

and

$$E(y^*|y) \approx \mu(Z^*) + K(Z^*, Z)B^{-1}y \tag{10}$$
$$V(y^*|y) \approx K(Z^*, Z^*) - K(Z^*, Z)B^{-1}K(Z, Z^*). \tag{11}$$

It follows that computing the likelihood function requires inverting a single $m^d \times m^d$ matrix, and computing matrix-vector products between $m^d \times N$ matrices and N dimensional vectors or $m^d \times m^d$ matrices and m^d dimensional vectors, i.e. total time linear in N when m^d is a constant. Making predictions using the trained model similarly takes time linear in N.

2.4. Learning

A key ingredient in our inference framework is that we learn all parameters of our model simultaneously. Assuming, for simplicity, that $\mu(Z) = 0$ in (9), we require inference for the parameters w of the DNN, the parameters σ_f^2, Δ of the Gaussian kernel and the noise variance σ^2. This is achieved by training the DNN with the loss function given by Eq (9). The normality of the output Z is enforced approximately by standardizing it before it enters (9). Note that this loss function is non-decomposable (in the sense e.g. of Kar et al. (2014)), so we opt not to perform the optimization with batches. However, this is greatly compensated by the linear computational and memory requirements of our method.

3. Related work

We have already presented in Section 1 a brief account of prior work related to our research. The computational complexity of Gaussian processes has been the major impediment to their widespread application in big data problems. Several review articles provide an account of the state-of-the-art. Besides the recent review by Liu et al. (2020), additional reviews can be found in Quiñonero-Candela & Rasmussen (2005); Camps-Valls et al. (2016); Rivera & Burnaev (2017). Finally, we note here deep Gaussian processes, introduced by Damianou & Lawrence (2013), are very much different in flavor and scope to our method, as the idea there is to model the output of a multivariate Gaussian process as input to another Gaussian Process, stacking several Gaussian Processes on top of each other in this way.

4. Experiments

The performance of our Deep Mercer Gaussian process method (DMGP) was compared against commonly used approximate but scalable Gaussian process methods on real-world datasets that are publicly available at the UCI dataset repository (Asuncin & Newman, 2007) and the official site of Rasmussen & Williams (2006). Simulated datasets were also used to illustrate some features of DMGP. Our experiments demonstrated that our method i) outperformed approximate Gaussian process methods on nearly all seven real-world datasets; ii) could be trained on datasets comprised of more than 1.8 million data points even by using a conventional, local machine; iii) achieved significant speedups against the baselines in both training and prediction time; iv) made predictions for all test points at the same time in no more than a second even when their number exceeds 10^5 data points; and v) was able to attain very similar performance to exact Gaussian process regression models or even surpass them on small simulated datasets.

All the experiments were carried out on a 72-core Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz Linux server. The implementation of our code was based on both Tensorflow (Abadi et al., 2015) and GPFlow (Matthews et al., 2017) and is avalaible at `https://github.com/aresPanos/DMGP_regression`. Finally, in all our experiments on real-world datasets, we standardized the input and target vectors (i.e., in the notation of Section 2, we set $\mu(Z) = 0$ throughout), we chose the output of our deep neural network g_w to always be one-dimensional (i.e. $d = 1$), and the activation function of each layer of the neural network to be $\tanh(.)$.

4.1. Simulated datasets

We generated two small-scale datasets to demonstrate the flexibility of DMGP and its performance against exact Gaussian Process regression. DMGP used, in both cases, a fully-connected DNN with a $[D\text{-}1]$ architecture, i.e. a single fully connected layer with D inputs and 1 output with a $\tanh(\cdot)$ activation function.

Table 1. Root-mean-squared error (RMSE) and negative log predictive density (NLPD) of DMGP and exact Gaussian Process for the two synthetic datasets. Best results are in bold. (Lower is better.)

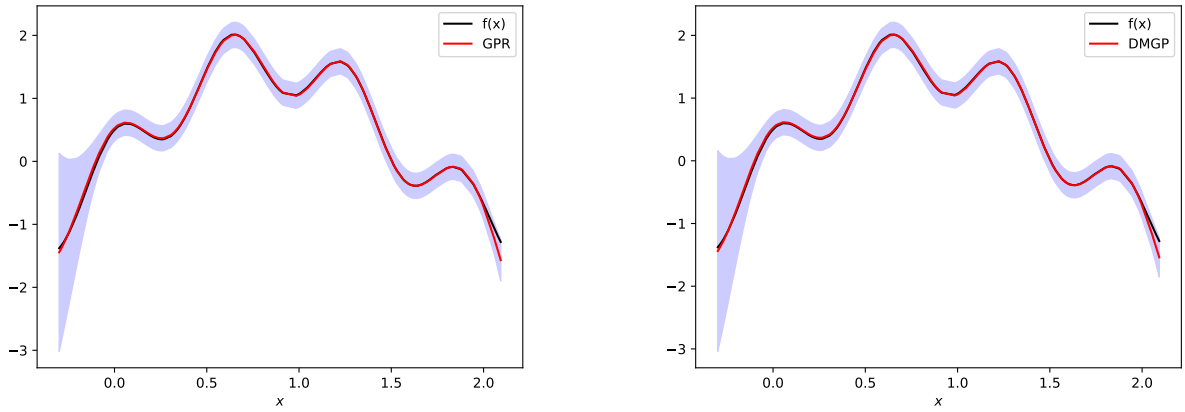
DATASET		RMSE	NLPD
DATA-1D	EXACT GP	0.0445	-1.1888
	DMGP	**0.0393**	**-1.1928**
DATA-4D	EXACT GP	2.0153	119.7685
	DMGP-1d	1.6052	1.9889
	DMGP-2d	1.6054	**1.9822**
	DMGP-3d	**1.6051**	1.9872

Data-1D. The first dataset, called Data-1D, includes one-dimensional training points $\{x_i\}_{i=1}^{1500}$ sampled uniformly on $[0, 2]$ and 200 test points sampled uniformly on $[-0.3, 2.1]$, with corresponding targets given by $y_i = f(x_i) + \epsilon$, where $f(x_i) = 1.5\sin(2x_i) + 0.5\cos(10x_i) + x_i/8$ and $\epsilon \sim \mathcal{N}(0, 0.01)$. We first train an exact Gaussian process model, with zero mean and a Gaussian kernel, using GPflow. Second, we use DMGP with $m = 20$. Figure 1 illustrates that both methods were able to correctly learn $f(x)$ (black line) on the test points with similar error guarantees. Table 1 shows the test root mean squared error (RMSE) and the negative log predictive density (NLPD). These results indicate that DMGP not only achieves similar performance to the exact GP model but slightly outperforms it on this data.

In order to gain better insight into the dependence of the choice of m and the accuracy of DMGP, we compute the test RMSE and NLPD as a function of m. Figures 3(a) and 3(b) demonstrate that in this simple dataset, we can match the performance of an exact Gaussian process using about $m = 20$ eigenfunctions. This reflects the fast decay of the eigenvalues λ_n, as depicted in Figure 3(d). Finally, Figure 3(c) gives the value of the optimized negative log-marginal likelihood as a function of m, while Figure 2 provides the training and test times for both exact Gaussian process and DMGP. Notice the training time gap between the exact Gaussian process and DMGP remains about the same for all small values of m that we tried. Additional plots of the DMGP model for different values of m are provided in Figure 4 in Appendix.

Data-4D. In this example we aim to show the feature engineering power of DMGP via its DNN embedding. We construct 4-dimensional input vectors $\{\mathbf{x}_i\}_{i=1}^{1300}$ as follows. For all i, we sample the coordinates of $\mathbf{x}_i = (x_i^1, x_i^2, x_i^3, x_i^4)$ as follows: $x_i^1 \sim \mathcal{N}(0, 1)$, $x_i^2 \sim \mathcal{N}(5, 4)$, $x_i^3 = \tau_i b_i$, and $x_i^4 = (1 - \tau_i)b_i$, where $b_i \sim \mathcal{N}(5, 4)$, and $\tau_i \sim \mathcal{U}[0, 1]$. The target vector $\mathbf{y}$ is sampled from a Gaussian process with 3-dimensional input vectors $\mathbf{z}_i = (x_i^1, x_i^2, x_i^3 + x_i^4)$, a Gaussian kernel with a shared $\epsilon^2 = 0.1$ across the three dimensions and $\sigma_f^2 = 1.5$, and noise variance $\sigma^2 = 0.01$. The train/test datasets consist of a random 1000/300 split of the $(\mathbf{x}_i, y_i)_{i=1}^{1300}$ data generated as described above. We train an exact Gaussian Process and DMGP with $d = 1$ and $m = 20$ on the training set. The performance of both methods on the test set is presented in Table 1. We find that the exact Gaussian process has larger RMSE and a very large NLPD value, because it is unable to "simulate" the true feature vector $(x_i^1, x_i^2, x_i^3 + x_i^4)$. In contract, DMGP has the expressive power to implement the right feature vector, achieving low RMSE and NLPD value.

Next, on the same dataset, we explore the effectiveness of increasing the output dimension d of the DNN. We attempt both $d = 2$ and $d = 3$ in tandem with $m = 5$ in both



(a) (b)

Figure 1. Comparison of the exact Gaussian process and DMGP on the Data-1D dataset. The black curve depicts the function $f(x) = 1.5\sin(2x) + 0.5\cos(10x) + x/8$ which is noisily observed in the train set. Panel (a) shows how the exact GP (red line) predicts the value of $f(x)$ on the test inputs, while Panel (b) shows how the DMGP (red line) predicts the value of $f(x)$ on the test inputs. For clarity, in both panels we have interpolated the predicted values on the test inputs to obtain a continuous curve. In both panels the blue shaded area corresponds to 95% confidence intervals.

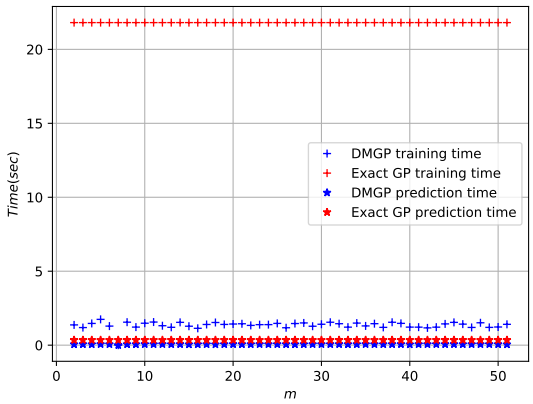


Figure 2. Training and prediction time comparisons between exact Gaussian Process (red) and DMGP (blue) on the Data-1D dataset as a function of the number of eigenfunctions m. Gaussian Process inference is conducted with a Gaussian kernel while DMGP utilizes a fully-connected DNN with architecture [D-1].

cases, resulting in 25 and 125 eigenvectors respectively. The results in Table 1 indicate that the combination $d = 1$ and $m = 20$ provides similar performance in terms of both RMSE and NLPD value.

4.2. Real-world datasets

Six UCI regression datasets and the Sarcos dataset (Rasmussen & Williams, 2006) are utilized for our experimental study on real-world data. Information about these datasets is provided in Table 2.

Table 2. Datasets used for our real-world data experiments. For each dataset, N and N^* are respectively the sizes of the training and test sets, and D is the number of features.

DATASET	D	N	N^*
ELEVATORS	18	14939	1660
PROTEIN	9	41157	4573
SARCOS	21	44039	4894
3DROAD	3	391386	43488
SONG	90	463810	51535
BUZZ	77	524925	58325
ELECTRIC	19	1844352	204928

Baselines. DMGP is tested against two widely-used and scalable approximate GP methods, namely Sparse Gaussian Process Regression (SGPR) (Titsias, 2009) and Stochastic Variational Gaussian Process (SVGP) (Hensman et al., 2013). We choose these methods because of their wide applicability and competitive performance on datasets of various scales. Both SGPR and SVGP rely on optimizing a lower bound of the exact log-marginal likelihood with respect to both kernel hyperparameters and function values at inducing points. SGPR can be seen as a special case of SVGP where this lower bound is optimized without stochastic estimation using minibatches and the used likelihood is Gaussian. The optimization of SVGP over minibatches allows it to scale well to large datasets of more than $500,000$ data points (Hensman et al., 2013). We use 500 inducing inputs for SGPR and 1000 for SVGP since these are common values for these methods in the literature (Matthews et al., 2017; Wang et al., 2019) and, equally importantly, because

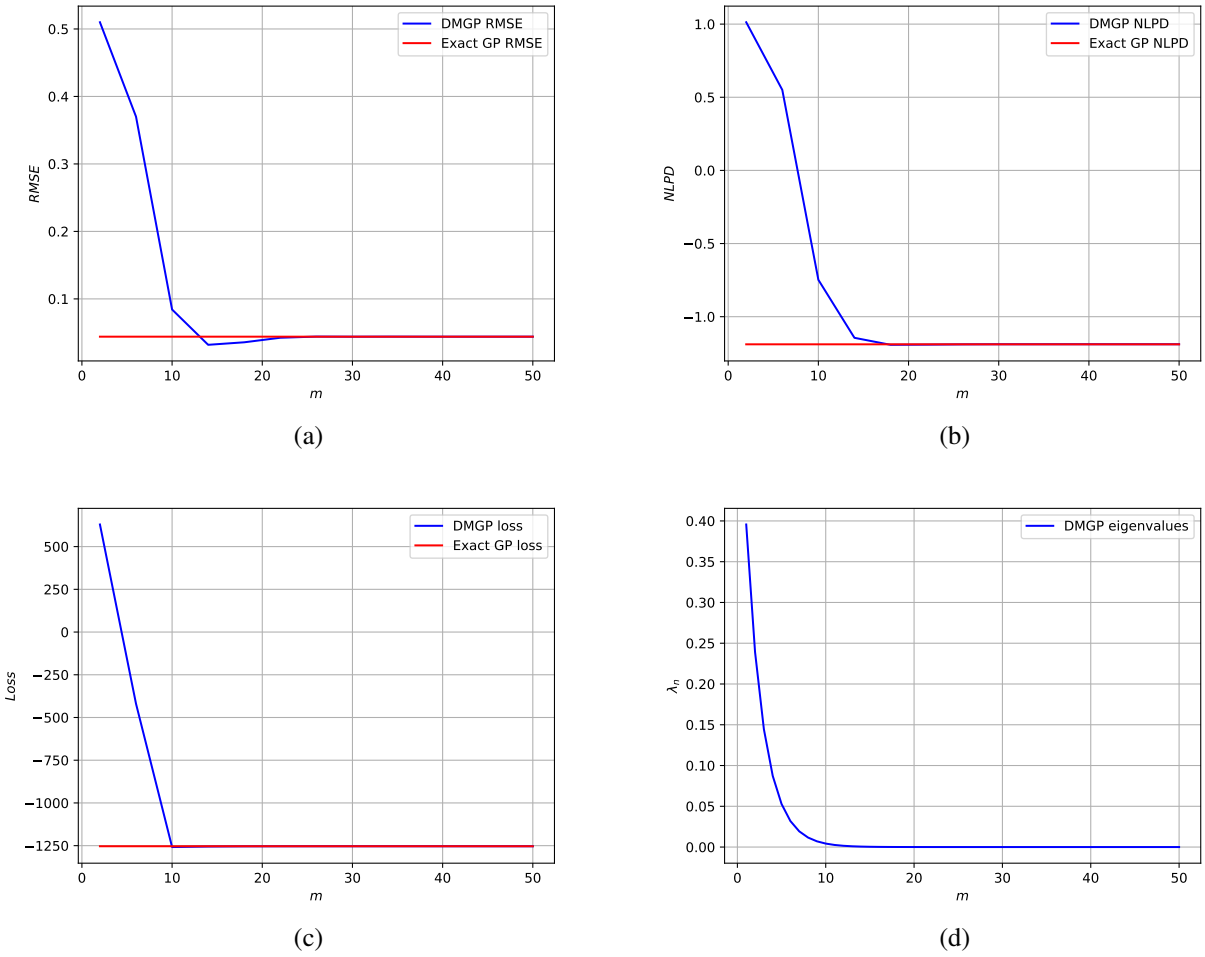


Figure 3. Several comparisons between exact Gaussian Process (red) and DMGP (blue) on the Data-1D dataset, as a function of the number of eigenfunctions m used in DMGP. Gaussian Process inference is conducted with a Gaussian kernel, while DMGP utilizes a fully-connected DNN with architecture [D-1]. Panel (a) presents the test RMSE as a function of m; (b) shows the test NLPD as a function of m; (c) presents the negative log marginal likelihood on the training set as a function of m; and (d) shows the decay of the first 50 eigenvalues λ_n of the optimized DMGP.

these values allow us to train these methods in a reasonable amount of time.

Experimental details. For all datasets with up to 10^5 data points we randomly split each dataset in ten equal partitions and then keep one of those for testing the rest nine partitions for training. We repeat this process for all partitions and report the averaged, across ten partitions, test RMSE and NLPD $\pm$ one standard deviation. For the larger datasets, we fail to run more than one partitions due to the high training time required for both SGPR and SVGP (see Table 4). Moreover, we have to omit accuracy results for both SGPR and SVGP on the largest, ELECTRIC dataset altogether, since both the training time and memory footprints were highly demanding for our available hardware. For both SGPR and SVGP, we use a zero-mean function and a Matérn 3/2 kernel with a shared lengthscale which is a common choice for these problems (Wang et al., 2019). For SVGP and SGPR we follow the training process of (Wang et al., 2019), which suggests running SVGP for 100 epochs with a minibatch of size 1000 and running SGPR for 1000 iterations. The Adam optimizer (Kingma & Ba, 2014) is our default optimizer with learning rates set to 0.1 and 0.01 respectively for SGPR and SVGP. Learning rates for DMGP are set by validation on each dataset separately; see supplementary material for details. Furthermore, we opt for a common DNN architecture across all datasets for DMGP, using in particular the [D-256-128-64-32-1] architecture, and we use $m = 25$ eigenfunctions throughout.

Predictive performance. Table 3 displays the predictive performance of DMGP, SGPR, and SVGP in terms of both test RMSE and NLPD. We see that DMGP consistently outperforms the two other Gaussian process approximations

Table 3. Root-mean-squared error (RMSE) and negative log predictive density (NLPD) of DMGP and the approximate Gaussian Process methods we use as baselines, on the SARCOS and UCI regression datasets. Best results are in bold. (Lower is better.)

	RMSE			NLPD		
DATASET	DMGP	SGPR	SVGP	DMGP	SGPR	SVGP
ELEVATORS	$\mathbf{0.360} \pm 0.001$	0.376 ± 0.008	0.391 ± 0.009	$\mathbf{0.397} \pm 0.030$	0.443 ± 0.019	0.490 ± 0.018
PROTEIN	$\mathbf{0.568} \pm 0.008$	0.620 ± 0.005	0.624 ± 0.011	$\mathbf{0.887} \pm 0.017$	0.946 ± 0.007	0.953 ± 0.014
SARCOS	$\mathbf{0.107} \pm 0.004$	0.160 ± 0.003	0.167 ± 0.003	$\mathbf{-0.763} \pm 0.050$	-0.385 ± 0.011	-0.332 ± 0.009
3DROAD	$\mathbf{0.369} \pm 0.047$	0.509 ± 0.004	0.471 ± 0.004	$\mathbf{0.452} \pm 0.053$	0.750 ± 0.008	0.676 ± 0.006
SONG	$\mathbf{0.794} \pm 0.017$	0.807 ± 0.003	0.817 ± 0.002	1.537 ± 0.051	$\mathbf{1.205} \pm 0.004$	1.217 ± 0.002
BUZZ	$\mathbf{0.234} \pm 0.001$	0.259 ± 0.001	0.261 ± 0.001	$\mathbf{-0.027} \pm 0.010$	0.052 ± 0.001	0.076 ± 0.005
ELECTRIC	$\mathbf{0.350} \pm 0.005$	0.575 ± 0.001	0.604 ± 0.001	$\mathbf{0.370} \pm 0.015$	0.869 ± 0.002	0.918 ± 0.001

Table 4. Training and prediction timing (in seconds) results of DMGP and approximate Gaussian processes based on the experimental details of Table 3. The prediction time is calculated over the full test dataset. Best results are in bold (lower the better).

	TRAINING			PREDICTION		
DATASET	DMGP	SGPR	SVGP	DMGP	SGPR	SVGP
ELEVATORS	$\mathbf{813.2} \pm 17.7$	859.5 ± 15.4	868.0 ± 25.4	$\mathbf{0.0} \pm 0.0$	0.4 ± 0.0	0.2 ± 0.0
PROTEIN	$\mathbf{913.4} \pm 14.6$	2014.4 ± 18.4	2737.4 ± 34.8	$\mathbf{0.1} \pm 0.0$	1.0 ± 0.0	0.6 ± 0.2
SARCOS	$\mathbf{923.2} \pm 14.3$	2091.3 ± 20.6	3995.8 ± 80.8	$\mathbf{0.1} \pm 0.0$	1.0 ± 0.0	0.8 ± 0.4
3DROAD	$\mathbf{4083.7} \pm 18.8$	17819.1 ± 115.4	34245.8 ± 160.0	$\mathbf{0.3} \pm 0.0$	8.8 ± 0.0	4.5 ± 0.0
SONG	$\mathbf{5341.2} \pm 19.3$	31915.8 ± 589.8	39103.9 ± 163.1	$\mathbf{0.4} \pm 0.0$	10.6 ± 0.0	7.5 ± 2.1
BUZZ	$\mathbf{5865.1} \pm 21.5$	34239.0 ± 486.3	42688.8 ± 171.8	$\mathbf{0.6} \pm 0.2$	13.3 ± 0.7	7.3 ± 0.5
ELECTRIC	$\mathbf{15337.1} \pm 18.0$	49452.7 ± 370.4	200502.9 ± 400.0	$\mathbf{1.4} \pm 0.1$	42.3 ± 0.7	23.3 ± 1.9

in almost every dataset in both RMSE and NLPD values.

Training and prediction times. The training and prediction times for each real-world dataset used in our study are shown in Table 4. The linear computational complexity of DMGP in the size N of the dataset results in significant advantages compared to the other methods in both training and prediction times. Notice that we run our method for 5000 iterations, compared to 1000 iterations used in SGPR, but still receive a considerable reduction in time. In particular, 1000 iterations of DMGP on the ELECTRIC dataset requires 3067 seconds while SGPR needs 49453 seconds. Moreover, SGPR needs more than 95 Gb memory to run while DMGP's memory footprint does not exceed 12 Gb. Hence, we could easily run our method on a conventional local machine without needing access to more sophisticated hardware such as that employed in Wang et al. (2019). The prediction time of DMGP demonstrates similar benefits where we attain more than $30\times$ and $20\times$ speedups over SGPR and SVGP, respectively, on the ELECTRIC dataset. Finally, it is worth noting the potential for additional speedups that our method can exhibit by utilizing more advanced hardware, since our objective function involves mainly[1] matrix-vector multiplications which can be very efficiently carried out by modern GPUs to further reduce the computation time significantly.

[1]There is also a Cholesky decomposition of the $m \times m$ matrix $\Xi^\top \Xi$, however this computation is negligible and does not affect the overall computational time of the log-marginal likelihood.

5. Conclusions

We contribute a new approximate Gaussian Process framework to address the important computational barrier facing Gaussian Processes. We use a deep neural network for feature extraction combined with a sparse covariance matrix approximation based on analytical expressions for the eigenvalues and eigenfunctions of the Gaussian Process kernel. Both the DNN and the kernel parameters are jointly trained. Training the DNN allows automatic feature engineering, accommodating datasets wherein the features of the regression inputs might have mixed types. The covariance matrix approximation allows jointly training the DNN and the kernel parameters in time linear in the size of the dataset. The use of the DNN for feature extraction alleviates the need to handcraft the metric on the feature space and the functional form of the kernel.

We perform an extensive series of experiments on simulated and real-world data of large and very large sizes. Even in the very large datasets we show that our method allows probabilistic predictions in moderate computation time and without need for sophisticated hardware. Our method consistently outperforms state-of-the-art scalable Gaussian Pro-

cess algorithms in both predictive accuracy and computation time. It does so even when using the same DNN architecture across all datasets, as well as the same values for m and d. We expect that more careful choices of the DNN architecture, specific to each dataset, will give even better results.

Although our inference framework allows the DNN to embed the inputs to d dimensions, the excellent results obtained with $d = 1$ across all real-world dataset problems, together with the results in our simulated datasets, provide experimental indication that $d = 1$ might be adequate in many applications. Of course, we can handcraft data where the one-dimensional embedding is inadequate and one should use larger d. By increasing d, one has to decrease m so that the resulting computational effort remains the same.

In this study, we confined ourselves to standard Gaussian process regression problems. Nonetheless, there is clearly scope to extend our work to Gaussian process classification and multi-task related problems in the future. We believe that probabilistic predictions will, and should, play an important role in machine learning applications, and a natural way to achieve this is via Gaussian Processes.

Acknowledgements

CD and PD acknowledge partial financial support by the Alan Turing Institute under the EPSRC grant EP/N510129/1.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

Al-Shedivat, M., Wilson, A. G., Saatchi, Y., Hu, Z., and Xing, E. P. Learning scalable deep kernels with recurrent structure. *The Journal of Machine Learning Research*, 18 (1):2850–2886, 2017.

Asuncin, A. and Newman, D. UCI Machine Learning Repository. http://www.ics.uci.edu/~mlearn/MLRepository.html, 2007. URL http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html.

Bradshaw, J., Matthews, A. G. d. G., and Ghahramani, Z. Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks. *arXiv preprint arXiv:1707.02476*, 2017.

Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. Manifold gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 3338–3345. IEEE, 2016.

Camps-Valls, G., Verrelst, J., Munoz-Mari, J., Laparra, V., Mateo-Jimenez, F., and Gomez-Dans, J. A survey on gaussian processes for earth-observation data analysis: A comprehensive investigation. *IEEE Geoscience and Remote Sensing Magazine*, 4(2):58–78, 2016.

Chalupka, K., Williams, C. K., and Murray, I. A framework for evaluating approximation methods for Gaussian process regression. *Journal of Machine Learning Research*, 14(Feb):333–350, 2013.

Cremanns, K. and Roos, D. Deep gaussian covariance network. *arXiv preprint arXiv:1710.06202*, 2017.

Damianou, A. and Lawrence, N. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pp. 207–215, 2013.

Deisenroth, M. and Ng, J. W. Distributed gaussian processes. In *International Conference on Machine Learning*, pp. 1481–1490, 2015.

Fasshauer, G. E. Greens functions: Taking another look at kernel approximation, radial basis functions, and splines. In *Approximation Theory XIII: San Antonio 2010*, pp. 37–63. Springer, 2012.

Fasshauer, G. E. and McCourt, M. J. Stable evaluation of gaussian radial basis function interpolants. *SIAM Journal on Scientific Computing*, 34(2):A737–A762, 2012.

Gneiting, T. Compactly supported correlation functions. *Journal of Multivariate Analysis*, 83(2):493–508, 2002.

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

Gramacy, R. B. et al. laGP: large-scale spatial modeling via local approximate Gaussian processes in R. *Journal of Statistical Software*, 72(1):1–46, 2016.

Hensman, J., Fusi, N., and Lawrence, N. D. Gaussian Processes for Big Data. In *Uncertainty in Artificial Intelligence*, 2013.

Hensman, J., Matthews, A., and Ghahramani, Z. Scalable variational gaussian process classification. 2015.

Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Iwata, T. and Ghahramani, Z. Improving output uncertainty estimation and generalization in deep learning via neural network gaussian processes. *arXiv preprint arXiv:1707.05922*, 2017.

Kar, P., Narasimhan, H., and Jain, P. Online and stochastic gradient methods for non-decomposable loss functions. In *Advances in Neural Information Processing Systems*, pp. 694–702, 2014.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Liu, H., Cai, J., Wang, Y., and Ong, Y. S. Generalized robust bayesian committee machine for large-scale gaussian process regression. In *International Conference on Machine Learning*, pp. 3131–3140, 2018.

Liu, H., Ong, Y.-S., Shen, X., and Cai, J. When Gaussian process meets big data: A review of scalable GPs. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Liu, W., Principe, J. C., and Haykin, S. *Kernel adaptive filtering: a comprehensive introduction*, volume 57. John Wiley & Sons, 2011.

Masoudnia, S. and Ebrahimpour, R. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42(2): 275–293, 2014.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017. URL http://jmlr.org/papers/v18/16-537.html.

Mercer, J. Functions of positive and negative type and their connection with the theory of integral equations, philosophical transsaction of the royal society of london, ser, 1909.

Quiñonero-Candela, J. and Rasmussen, C. E. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

Rasmussen, C. E. and Ghahramani, Z. Infinite mixtures of Gaussian process experts. In *Advances in neural information processing systems*, pp. 881–888, 2002.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Rivera, R. and Burnaev, E. Forecasting of commercial sales with large scale gaussian processes. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 625–634. IEEE, 2017.

Rullière, D., Durrande, N., Bachoc, F., and Chevalier, C. Nested kriging predictions for datasets with a large number of observations. *Statistics and Computing*, 28(4): 849–867, 2018.

Sculley, D. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pp. 1177–1178, 2010.

Sun, S. and Xu, X. Variational inference for infinite mixtures of Gaussian processes with applications to traffic flow prediction. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):466–475, 2010.

Titsias, M. Variational learning of inducing variables in sparse Gaussian processes. In *Artificial Intelligence and Statistics*, pp. 567–574, 2009.

Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K. Q., and Wilson, A. G. Exact gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, pp. 14622–14632, 2019.

Wilson, A. and Nickisch, H. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *International Conference on Machine Learning*, pp. 1775–1784, 2015.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. Deep kernel learning. In *Artificial Intelligence and Statistics*, pp. 370–378, 2016a.

Wilson, A. G., Hu, Z., Salakhutdinov, R. R., and Xing, E. P. Stochastic variational deep kernel learning. In *Advances in Neural Information Processing Systems*, pp. 2586–2594, 2016b.

Yuksel, S. E., Wilson, J. N., and Gader, P. D. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems*, 23(8):1177–1193, 2012.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997a.

Zhu, H., Williams, C. K., Rohwer, R., and Morciniec, M. Gaussian regression and optimal finite dimensional linear models. 1997b.

Appendix

A. Extra experimental details

In this section, we provide extra information about the setup for the experiments of the main paper. All the datasets used had each of their features standardized. Similarly, standardization was applied to the target vector $\mathbf{y}$. The ELECTRIC dataset was the only one wherein we did further feature engineering due to the time related features it involved. More precisely, we used one-hot encoding for the four distinctive years of the electric power consumption measurements. Moreover, features such as days, months, hours, and minutes were converted to $\sin/\cos$ values due to their cyclical nature. The remaining (seven) features were standardized. As target for this experiment we set the natural logarithm of the "voltage" feature.

Our DMGP method used the Adam optimizer on all the real-world datasets, and used the L-BFGS-B optimizer (Zhu et al., 1997a) on the two simulated datasets given the small number of parameters that these optimizations involved.

For each of our real-world dataset experiments, we did hyperparameter tuning of our DMGP model to choose the learning rate of the Adam optimizer and the initialization of the two kernel parameters (ϵ^2 and σ_f^2) and the noise variance (σ^2). To do so, we randomly kept 90% of the dataset for training and the remaining 10% of it for validation. We grid-searched over our parameter space, running DMGP for 1000 iterations for each combination of parameters, and ultimately selected the combination that did best in terms of NLPD on the validation set. The linear time complexity of our method makes it cheap to run each of these 1000 iterations during grid search, and does not impose a big cost even for the large datasets. The random split for hyperparameter tuning was different from the 10 splits described in the main paper, which were employed for the evaluation of our method in the reported results. The results of hyperparameter tuning are shown in Table 5, which shows the learning rate and how the kernel parameters ϵ, σ_f and the noise variance σ were initialized for each dataset.

Table 5. Experimental setup used for running DMGP on the real-world datasets of the main paper. For each dataset, we show the learning rate used in the Adam optimizer and the initialization of the kernel parameters and the noise variance.

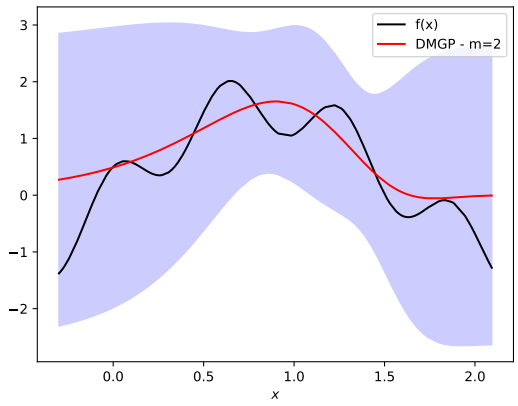
Dataset	Elevator	Protein	Sarcos	3DRoad	Song	Buzz	Electric
learning rate	0.0001	0.002	0.001	0.01	0.00001	0.001	0.005
ϵ^2	1	1	1	1	1	1	1
σ_f^2	1	1	1	8	0.2	7	10
σ^2	0.1	0.1	0.1	1	3	0.5	1

We note that the inducing inputs for each of the baselines SGPR and SVGP were initialized by mini-batch k-means (Sculley, 2010), which is a common strategy (Hensman et al., 2013; 2015).

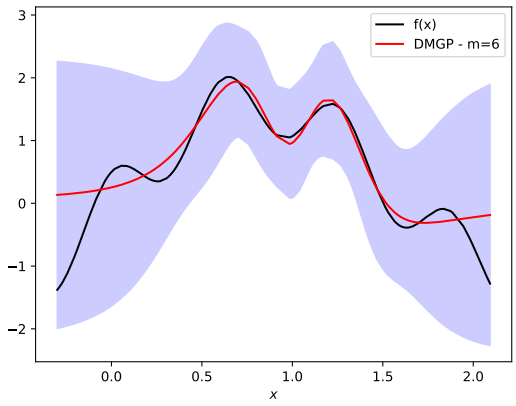
Finally, all reported training times of Table 4 in the main paper were computed by multiplying the total number of iterations we run each method for with the average time that the first ten iterations of each method needed to complete.

B. Additional plots

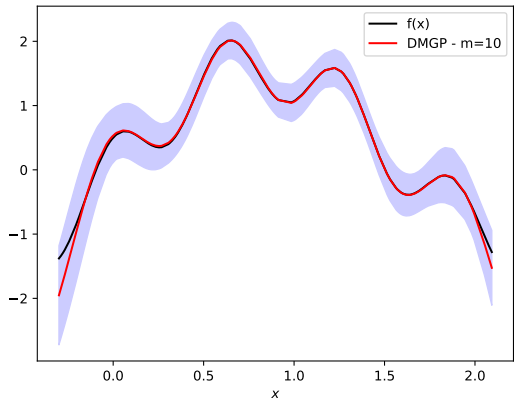
We also provide some extra plots based on the Data-1D dataset of the main paper in Figure 4 where a different number of eigemfunctions has been used for each case.



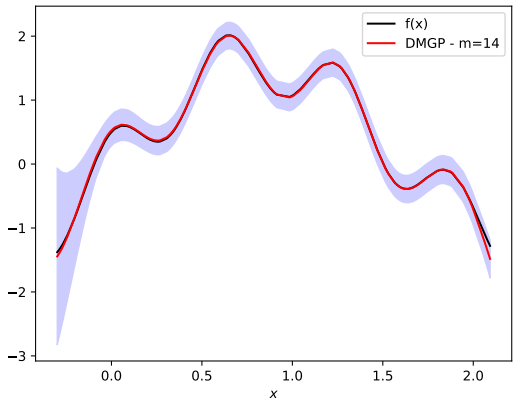
$m = 2$



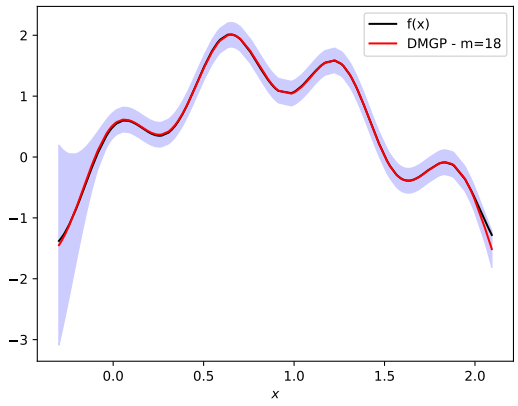
$m = 6$



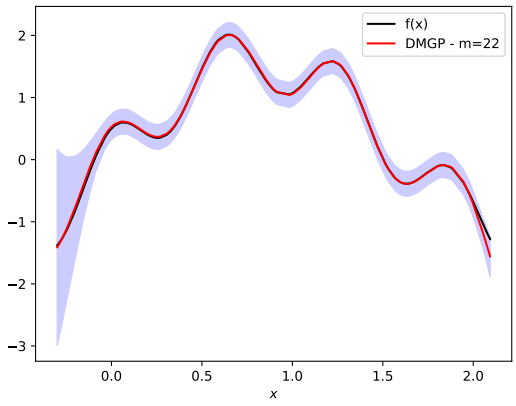
$m = 10$



$m = 14$



$m = 18$



$m = 22$

Figure 4. Each panel depicts $f(x) = 1.5\sin(2x) + 0.5\cos(10x) + x/8$ (black) and DMGP (blue) on the Data-1D dataset using different number of eigenfunctions m. The shaded area demonstrates 95% mass of the predictive density. More results regarding this dataset can be found in Figure 3 of the main paper.