Scalable Gaussian Processes with Billions of Inducing Inputs via Tensor Train Decomposition

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

We propose a new method (TT-GP) for approximate inference in Gaussian process models. We build on previous results on scalable GPs including stochastic variational inference based on inducing inputs, kernel interpolation and structure exploiting algebra. We also utilize the Tensor Train decomposition which allows us to efficiently train GP models with billions of inducing inputs and improve the results of previous GP models on several benchmark datasets. Further, our approach allows us to train kernels based on deep neural networks without any modifications of the underlying GP model. Our model allows end-to-end training of both GP and Neural Network parameters without pretraining through maximization of GP marginal likelihood. In this approach DNN learns a multidimensional embedding for the data, which is used by the GP to make the final prediction. We show the efficiency of the proposed approach on several widely used classification benchmark datasets including CIFAR10, MNIST and Airline.

1 Introduction

Gaussian processes (GP) provide a prior over functions and allow finding complex regularities in data. The ability of GPs to adjust the complexity of the model to the size of the data makes it apealling to use them for big datasets. Unfortunately, standard methods for GP regression and classification scale as $\mathcal{O}(n^3)$ with the size of the data n and can not be applied when n exceeds several thousands.

Numerous approximate inference methods have been proposed in the literature. Many of these methods are based on the concept of inducing inputs (Quiñonero-Candela and Rasmussen [13], Snelson and Ghahramani [17], Williams and Seeger [19]). These methods build a smaller set Z of m points that serve to approximate the true posterior of the process and reduce the complexity to $\mathcal{O}(nm^2 + m^3)$. Titsias [18] proposed to consider the values u of the Gaussian process at the inducing inputs as latent variables and derived a variational inference procedure to approximate the posterior distribution of these variables. Hensman et al. [3] and Hensman et al. [4] extended this framework by using stochastic optimization to scale up the method and generalizing it to classification problems.

Inducing input methods allowed to use Gaussian processes on datasets containing million of examples. However, these methods are still limited with the number of inducing points m they can use. Wilson and Nickisch [20] proposed the KISS-GP framework, which exploits the Kronecker product structure in covariance matrices for inducing points placed on a multidimensional grid in the feature space. KISS-GP has complexity $\mathcal{O}(n + Dm^{1+1/D})$, where D is the dimensionality of the feature space. Note however, that m is the number of points in a D-dimensional grid and grows exponentially with D, which makes the method inapplicable when the number of features D is larger than 4.

In this paper we propose a new method TT-GP, that can use billions of inducing inputs and is not limited to very low-dimensional feature spaces. We achieve this by combining kernel interpolation and Kronecker algebra of KISS-GP with a scalable variational inference procedure. We restrict the family of variational distributions from Hensman et al. [3] to have parameters in special formats. Specifically, we use Kronecker product format for the covariance matrix Σ and Tensor Train format (Oseledets [12]) for the expectation μ of the variational distribution over the values u of the process at inducing inputs Z.

The proposed TT-GP method naturally allows to train expressive kernel functions on big datasets. Wilson et al. [22] and Wilson et al. [21] demonstrated the efficiency of Gaussian processes with kernels based on deep neural networks. They used subsets of the outputs of a neural network as inputs for the Gaussian process. As the authors were using KISS-GP, they were limited to using low dimensional Gaussian processes and had to pretrain the network before adding the GP layer. The proposed TT-GP method allows us to learn multidimensional embeddings and train the model end-to-end.

We demonstrate the predictive performance and scalability of the proposed TT-GP method both with standard RBF and deep kernels on a wide range of classification and regression tasks.

2 Background

2.1 Gaussian Processes

Here we provide a brief review of standard Gaussian process methods and their limitations.

A Gaussian process is a collection of random variables, any finite number of which have a joint normal distribution. A GP f taking place in $\mathbb{R}^D$ is fully defined by its mean $m : \mathbb{R}^D \to \mathbb{R}$ and covariance $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ functions. For every $t_1, t_2, \dots, t_n \in \mathbb{R}^D$

$$f(t_1), f(t_2), \dots, f(t_n) \sim \mathcal{N}(m, K),$$

where $m = (m(t_1), m(t_2), \dots, m(t_n))^T \in \mathbb{R}^n$, and $K \in \mathbb{R}^{n \times n}$ is the matrix comprised of pairwise values of covariance function k. Below we will use notation $K(A, B)$ for the matrix of pairwise values of covariance function k on points from sets A and B.

Consider regression task. The dataset consists of n objects $X = (x_1, \dots, x_n)^T \in \mathbb{R}^{n \times D}$, and target values $y = (y_1, y_2, \dots, y_n)^T \in \mathbb{R}^n$. We will assume that the data is generated by a latent Gaussian process f taking place in the feature space and denote the value of the process at data point x_i by f_i for all $i = 1, 2, \dots, n$. We will use a zero-mean GP prior with covariance function k for f. We will assume that the observed target variables y_i are a noisy version of f_i:

$$p(y_i|f_i) = \mathcal{N}(y_i|f_i, \nu^2 I),$$

where ν^2 is the noise variance.

Assume that we want to predict the values of the process f_* at a set of test points X_*. As the joint distribution of y and f_* is Gaussian, we can anlytically compute the conditional distribution

$$p(f_*|y, X, X_*) = \mathcal{N}(f_*|\hat{m}, \hat{K}),$$

where

$$\hat{m} = K(X_*, X)(K(X, X) + \nu^2 I)^{-1} y,$$
$$\hat{K} = K(X_*, X_*) - K(X_*, X)(K(X, X) + \nu^2 I)^{-1} K(X, X_*)).$$

Popular covariance functions usually have a set of hyper-parameters θ. For example, the RBF kernel

$$k_{\text{RBF}}(x, x') = \sigma_f^2 \exp\left(-0.5\|x - x'\|^2/l^2\right)$$

has two parameters l and σ_f. In order to fit the model to the data, we can maximize the marginal likelihood of the process with respect to these parameters. In case of GP regression we can compute the marginal likelihood analytically

$$\log p(y) = -\frac{1}{2} y^T (K_\theta(X, X) + \nu^2 I)^{-1} y - \frac{1}{2} \log |K_\theta(X, X) + \nu^2 I| - \frac{n}{2} \log 2\pi,$$

where $K_\theta(X, X)$ denotes the covariance matrix for the specific value of the hyper-parameters θ.

Note that the complexity of computing both predictive distribution and marginal likelihood is $\mathcal{O}(n^3)$. For classification we substitute the normal $p(y_i|f_i)$ distribution with a sigmoid

$$p(y_i|f_i) = \sigma(y_i f_i).$$

For classification both predictive distribution and marginal likelihood are intractable. For detailed description of GP regression and classification see Rasmussen and Williams [14]

2.2 Inducing Inputs

A number of approximate methods were developed to scale up Gaussian processes. Hensman et al. [3] proposed a variational lower bound that factorizes over observations for Gaussian process marginal likelihood. We rederive this bound here.

Consider a set $Z \in \mathbb{R}^{m \times D}$ of m points in the feature space. We will call points Z inducing inputs or inducing points. We will introduce latent variables $u \in \mathbb{R}^m$ representing the values of the Gaussian process at these points. Consider the augmented model

$$p(y, f, u) = p(y|f)p(f|u)p(u) = \prod_{i=1}^{n} p(y_i|f_i)p(f|u)p(u),$$

where

$$p(f|u) = \mathcal{N}(f|K_{nm}K_{mm}^{-1}u, K_{nn} - K_{nm}K_{mm}^{-1}K_{mn}), \quad p(u) = \mathcal{N}(u|0, K_{mm}). \tag{1}$$

Here and below we ommit the inputs X to simplify the notation.

The standard variational lower bound is given by

$$\log p(y) \geq \mathbb{E}_{q(u,f)} \log \frac{p(y, f, u)}{q(u, f)} = \mathbb{E}_{q(f)} \log \prod_{i=1}^{n} p(y_i|f_i) - \text{KL}(q(u, f)||p(u, f)), \tag{2}$$

where $q(u, f)$ is the variational distribution over latent variables. Consider the following family of variational distributions

$$q(u, f) = p(f|u)\mathcal{N}(u|\mu, \Sigma), \tag{3}$$

where $\mu \in \mathbb{R}^m$ and $\Sigma \in \mathbb{R}^{m \times m}$ are variational parameters.

Throughout the paper we will use the following notation.

$$K_{nn} = K(X, X), \quad K_{nm} = K(X, Z), \quad K_{mn} = K(Z, X) = K_{nm}^T, \quad K_{mm} = K(Z, Z).$$

We can then rewrite the KL-divergence tetm in (2) as follows

$$\text{KL}(q(u, f)||p(u, f)) = \text{KL}(p(f|u)q(u)||p(f|u)p(u)) = \text{KL}(q(u)||p(u)) = \\ = \frac{1}{2}\left(\log \frac{|K_{mm}|}{|\Sigma|} - m + \text{tr}(K_{mm}^{-1}\Sigma + \mu^T K_{mm}^{-1}\mu)\right).$$

The marginal distribution over f can be computed analytically

$$q(f) = \mathcal{N}\left(f|K_{nm}K_{mm}^{-1}\mu, K_{nn} + K_{nm}K_{mm}^{-1}(\Sigma - K_{mm})K_{mm}^{-1}K_{mn}\right).$$

We can then rewrite (2) as

$$\log p(y) \geq \sum_{i=1}^{n} \mathbb{E}_{q(f_i)} \log p(y_i|f_i) - \text{KL}(q(u)||p(u)). \tag{4}$$

Note that the lower bound (4) factorizes over observations and thus stochastic optimization can be applied to maximize this bound with respect to both kernel hyper-parameters θ and variational parameters μ and Σ. In case of regression we can rewrite (4) in the closed form

$$\log p(y) \geq \sum_{i=1}^{n}\left(\log \mathcal{N}(y_i|k_i^T K_{mm}^{-1}\mu, \nu^2) - \frac{1}{2\nu^2}\tilde{K}_{ii} - \frac{1}{2\nu^2}\text{tr}(k_i^T K_{mm}^{-1}\Sigma K_{mm}^{-1}k_i)\right) - \\ -\frac{1}{2}\left(\log \frac{|K_{mm}|}{|\Sigma|} - m + \text{tr}(K_{mm}^{-1}\Sigma) + \mu^T K_{mm}^{-1}\mu\right), \tag{5}$$

where $k_i \in \mathbb{R}^m$ is the i-th column of K_{mn} matrix and

$$\tilde{K} = K_{nn} - K_{nm}K_{mm}^{-1}K_{mn}.$$

At prediction time we can use the variational distribution as a substitute for posterior

$$p(f_*|y) = \int p(f_*|f,u)p(f,u|y)dfdu \approx \int p(f_*|f,u)q(f,u)dudf = \int p(f_*|u)q(u)du.$$

The complexity of computing the bound (5) is $\mathcal{O}(nm^2 + m^3)$. Hensman et al. [4] proposes to use Gauss-Hermite quadratures to approximate the expectation term in (4) for binary classification problem to obtain the same computational complexity $\mathcal{O}(nm^2 + m^3)$. This complexity allows to use Gaussian processes in tasks with millions of training samples, but these methods are limited to use small numbers of inducing points m, which hurts the predictive performance and doesn't allow to learn expressive kernel functions.

2.3 KISS-GP

Saatçi [15] noted that the covariance matrices computed at points on a multidimensional grid in the feature space can be represented as a Kronecker product if the kernel function factorizes over dimensions

$$k(x,x') = k_1(x^1, x'^1) \cdot k_2(x^2, x'^2) \cdot \ldots \cdot k_D(x^D, x'^D). \tag{6}$$

Note, that many popular covariance functions, including RBF, belong to this class. Kronecker structure of covariance matrices allows to perform efficient inference for full Gaussian processes with inputs X on a grid.

Wilson and Nickisch [20] proposed to set inducing inputs Z on a grid:

$$Z = Z^1 \times Z^2 \times \ldots \times Z^D, \quad Z^i \in \mathbb{R}^{m_i} \quad \forall i = 1, 2, \ldots, D.$$

The number m of inducing points is then given by

$$m = \prod_{i=1}^{D} m_i.$$

Let the covariance function satisfy (6). Then the covariance matrix K_{mm} can be represented as a Kronecker product over dimensions

$$K_{mm} = K^1_{m_1m_1} \otimes K^2_{m_2m_2} \otimes \ldots \otimes K^D_{m_Dm_D},$$

where

$$K^i_{m_im_i} = K_i(Z_i, Z_i) \in \mathbb{R}^{m_i \times m_i} \quad \forall i = 1, 2, \ldots, D.$$

Kronecker products allow efficient computation of inverse

$$(A_1 \otimes A_2 \otimes \ldots \otimes A_D)^{-1} = A_1^{-1} \otimes A_2^{-1} \otimes \ldots \otimes A_D^{-1},$$

and determinant

$$|A_1 \otimes A_2 \otimes \ldots \otimes A_D| = |A_1|^{c_1} \cdot |A_2|^{c_2} \cdot \ldots \cdot |A_D|^{c_D},$$

where

$$A_i \in \mathbb{R}^{k_i \times k_i}, \quad c_i = \prod_{j \neq i} k_j, \quad \forall i = 1, 2, \ldots, D.$$

Another major idea of KISS-GP is to use interpolation to approximate K_{mn}. Considering inducing inputs as interpolation points for the function $k(\cdot, z_i)$ we can write

$$K_{mn} \approx K_{mm}W, \quad k_i \approx K_{mm}w_i, \tag{7}$$

where $W \in \mathbb{R}^{m \times n}$ contains the coefficients of interpolation, and w_i is it's i-th column. Authors of KISS-GP suggest using cubic convolutional interpolation (Keys [6]), in which case the interpolation weights w_i can be represented as a Kronecker product over dimensions

$$w_i = w_i^1 \otimes w_i^2 \otimes \ldots \otimes w_i^D, \quad w_i \in \mathbb{R}^{m_i} \quad \forall i = 1, 2, \ldots, D.$$

Wilson and Nickisch [20] combine these ideas with SOR (Silverman [16]) in the KISS-GP method with $\mathcal{O}(n + Dm^{1+1/D})$ computational complexity. This complexity allows to use KISS-GP with a large number (possibly greater than n) of inducing points. Note, however, that m grows exponentially with the dimensionality D of the feature space, and the method becomes impractical when $D \gg 4$. We don't describe other details of KISS-GP here.

2.4 Tensor Train Decomposition

Tensor Train (TT) decomposition proposed in Oseledets [12] allow to efficiently store tensors (multidimensional arrays of data), large matrices and vectors. For matrices and vectors in TT-format linear algebra operations can be implemented efficiently. TT format was successfully applied for different machine learning tasks (see Novikov et al. [10], Novikov et al. [11]).

Consider a D-dimensional tensor $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times \ldots \times k_D}$. $\mathcal{A}$ is said to be in the Tensor Train format if

$$\mathcal{A}(i_1, i_2, \ldots, i_d) = G_1[i_1] \cdot G_2[i_2] \cdot \ldots \cdot G_D[i_D], \quad i_k \in \{1, 2, \ldots, n_k\} \ \forall k, \tag{8}$$

where

$$G_k[i_k] \in \mathbb{R}^{r_k \times r_{k+1}} \ \ \forall k, i_k, \quad r_0 = r_{D+1} = 1.$$

Matrices G_k are called TT-cores, and numbers r_k are called TT-ranks of tensor $\mathcal{A}$.

In order to represent a vector in TT-format, it is reshaped to a multidimensional tensor (possibly with zero padding) and then format (8) is used. We will use TT-format for the vector μ of expectations of the values u of the Gaussian process in points Z placed on a multidimensional grid. In this case, μ is naturally represented as a D-dimensional tensor.

For matrices TT format is given by

$$M(i_1, i_2, \ldots, i_d; j_1, j_2, \ldots, j_D) = G_1[i_1, j_1] \cdot G_2[i_2, j_2] \cdot \ldots \cdot G_D[i_D, j_D],$$

where

$$G_k[i_k, j_k] \in \mathbb{R}^{r_k \times r_{k+1}} \ \ \forall k, i_k, j_k, \quad r_0 = r_{D+1} = 1.$$

Note, that Kronecker product format is a special case of TT for TT-ranks $r_1 = r_2 = \ldots = r_{D+1} = 1$.

Many operations of linear algebra can be efficiently implemented for TT vectors and matrices. Let $u, v \in \mathbb{R}^{n_1 \cdot n_2 \cdot \ldots \cdot n_D}$ be vectors in TT-format with TT-ranks not greater than r.

$$u(i_1, i_2, \ldots, i_D) = u_1[i_1] \cdot u_2[i_2] \cdot \ldots \cdot u_D[i_D], \quad i_k \in \{1, 2, \ldots, n_k\} \ \forall k,$$

and the same for v. Let A and B be represented as a Kronecker product

$$A = A_1 \otimes A_2 \otimes \ldots \otimes A_D, \quad A_k \in \mathbb{R}^{n_k \times n_k} \ \ \forall k,$$

and the same for B. Let $n = \max_k n_k$. Then the computational complexity of computing the quadratic form $u^T A v$ is $\mathcal{O}(Dnr^3)$. The computational complexity of computing tr(AB) is $\mathcal{O}(Dn^2)$. We will need these two operations below.

3 TT-GP

In the previous section we described several methods for GP regression and classification. All these methods have different limitations. Standard methods can not be applied for big datasets, KISS-GP requires small dimensionality of the feature space and other methods based on inducing points are limited to use a small number m of these points. In this section we propose the TT-GP method that can be used with big datasets and can incorporate billions of inducing inputs. TT-GP can naturally be used for training expressive deep kernels to work with structured data (e.g. images).

3.1 Variational Parameters Approximation

In section 2.2 we derived the variational lower bound of Hensman et al. [3]. We will place the inducing inputs Z on a multidimensional grid in the feature space and we will assume the the covariance function satisfies (6). Let the number of inducing points in each dimension be m_0. Then

$$m = m_0^D.$$

As shown in section 2.3, in this case K_{mm} matrix can be rewritten as a Kronecker product over dimensions. Substituting the approximation (7) into the lower bound (5), we obtain

$$\begin{aligned}\log p(y) \geq \sum_{i=1}^{n} \left(\log \mathcal{N}(y_i | w_i^T \mu, \nu^2) - \frac{1}{2\nu^2} \tilde{K}_{ii} - \frac{1}{2\nu^2} \text{tr}(w_i^T \Sigma w_i) \right) - \\ -\frac{1}{2} \left(\log \frac{|K_{mm}|}{|\Sigma|} - m + \text{tr}(K_{mm}^{-1} \Sigma) + \mu^T K_{mm}^{-1} \mu \right),\end{aligned} \tag{9}$$

where $\tilde{K}_{ii} = k(x_i, x_i) - w_i^T K_{mm} w_i$.

Note that K_{mm}^{-1} and $|K_{mm}|$ can be computed with $\mathcal{O}(Dm_0^3) = \mathcal{O}(Dm^{3/D})$ operations due to the Kronecker product structure. Now the most computationally demanding terms are those containing variational parameters μ and Σ.

Let us restrict the family of variational distributions (3). Let Σ be a Kronecker product over dimensions, and μ be in TT-format with TT-ranks not greater than r. Then, according to section 2.4, we can then compute the lower bound with $\mathcal{O}(nDm_0r^2 + Dm_0r^3 + Dm_0^3) = \mathcal{O}(nDm^{1/D}r^2 + Dm^{1/D}r^3 + Dm^{1/D})$.

Thus, the proposed TT-GP method has linear complexity with respect to dimensionality D of the feature space, despite the exponential growth of the number of inducing inputs. Lower bound (9) can be maximized with respect to kernel hyper-parameters θ, TT-cores of μ and Kronecker multipliers of Σ. Note that stochastic optimization can be applied, as the bound (9) factorizes over data points. The total number of optimized parameters is $\mathcal{O}(\sharp\theta + Dm^{1/D} + Dm^{2/D})$, where $\sharp\theta$ is the number of kernel hyper-parameters.

3.2 Classification

In this section we describe a generalization of the proposed method for multiclass classification. In this case the dataset consists of features $X = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^{n \times D}$ and target values $y = (y_1, y_2, \dots, y_n)^T \in \{1, 2, \dots, C\}^n$, where C is the number of classes.

Consider C Gaussian processes taking place in $\mathbb{R}^D$. Each process corresponds to it's own class. We will place $m = m_0^D$ inducing points Z on a grid in the feature space, and they will be shared between all processes. Each process has it's own set of latent variables representing the values of the process at data points $f^c \in \mathbb{R}^n$, and inducing inputs $u^c \in \mathbb{R}^m$. We will use the following model

$$p(y, f, u) = \prod_{i=1}^{n} p(y_i | f_i^{1,2,\dots,C}) \prod_{c=1}^{C} p(f^c | u^c) p(u^c),$$

where $p(f^c|u^c)$ and $p(u^c)$ are defined as in (1). For $p(y_i|f_i^{1,2,\dots,C})$ we will use discrete distribution with probabilities

$$p(y_i = c | f_i^{1,2,\dots,C}) = \frac{\exp(f_i^c)}{\sum_{j=1}^{C} \exp(f_i^j)}.$$

We will use variational distributions of form

$$q(f^1, f^2, \dots, f^C, u^1, u^2, \dots, u^C) = q(f^1, u^1) \cdot q(f^2, u^2) \cdot \ldots \cdot q(f^C, u^C),$$

where

$$q(f^c, u^c) = p(f^c|u^c) \mathcal{N}(u^c | \mu^c, \Sigma^c) \quad c = 1, 2, \dots, C,$$

where all μ^c are represented in TT-format with TT-ranks not greater than r, and Σ^c are represented as Kronecker products over dimensions. Similarly to (4), we obtain

$$\log p(y) \geq \sum_{i=1}^{n} \mathbb{E}_{q(f_i^{1,2,\dots,C})} \log p(y_i | f_i^{1,2,\dots,C}) - \sum_{c=1}^{C} \mathrm{KL}(q(u_c) || p(u_c)) \tag{10}$$

The second term in (10) can be computed analytically as a sum of KL-divergences between normal distributions. The first term is intractable. In order to approximate the first term we will use a lower bound. Let y_i belong to class c. Then, we can rewrite

$$\mathbb{E}_{q(f_i^{1,2,\dots,C})} \log p(y_i | f_i^{1,2,\dots,C}) = \mathbb{E}_{q(f_i^c)} f_i^c - \mathbb{E}_{q(f_i^{1,2,\dots,C})} \log \left(\sum_{j=1}^{C} \exp(f_i^j) \right), \tag{11}$$

where

$$q(f_i^{1,2\dots,C}) = \mathcal{N}(f^1 | m_i^1, s_i^1) \cdot \mathcal{N}(f^2 | m_2, s_i^2) \cdot \ldots \cdot \mathcal{N}(f^C | m_i^C, s_i^C).$$

The first term in (11) is obviously tractable, while the second term has to be approximated. Bouchard [1] discusses several lower bounds for expectations of this type. Below we derive one of these bounds, which we use in TT-GP.

Concavity of logarithm implies

$$\log(\sum_{j=1}^{C} \exp(f_i^j)) \leq \log \frac{1}{\varphi} + \varphi \left(\sum_{j=1}^{C} \exp(f_i^j) - \frac{1}{\varphi} \right) = \varphi \sum_{j=1}^{C} \exp(f_i^j) - \log \varphi - 1,$$

where the right hand side is the linearization of the logarithm at $\frac{1}{\varphi}$. Taking expectation of both sides of the inequality and minimizing with respect to φ, we obtain

$$\mathbb{E}_{q(f_i^{1,2,\ldots,C})} \log \left(\sum_{j=1}^{C} \exp(f_i^j) \right) \leq \log \left(\sum_{j=1}^{C} \exp \left(m_i^j + \frac{1}{2} s_i^j \right) \right). \tag{12}$$

Substituting (12) back into (10) we obtain a tractable lower bound for multiclass classification task, that can be maximized with respect to kernel hyper-parameters θ^c, TT-cores of μ^c and Kronecker factors of Σ^c. The complexity of the method is C times higher, than in regression case.

3.3 Deep kernels

Wilson et al. [22] and Wilson et al. [22] showed the efficiency of using expressive kernel functions based on deep neural networks with Gaussian processes on a variety of tasks. The proposed TT-GP method is naturally compatible with this idea.

Consider a covariance function k (satisfying (6)) and a neural network (or in fact any parametric transform) net. We can define a new kernel as follows

$$k_{net}(x, x') = k(net(x), net(x')).$$

We can train the neural network weights through maximization of GP marginal likelihood, the same way, as we normally train kernel hyper-parameters θ. This way, the network learns a multidimensional embedding for the data, and the GP is making the prediction working with this embedding. Wilson et al. [21] trained one-dimensional GPs on different outputs of the network. TT-GP allows us to train Gaussian processes on all network outputs, and train the whole model end-to-end without pretraining.

4 Experiments

In this section we evaluate the proposed TT-GP method in different settings and compare it with existing approaches. We first compare our method with SVI-GP (Hensman et al. [3]) on regression tasks and KLSP-GP (Hensman et al. [4]) on binary classification tasks using standard RBF kernel functions. Then, we test the ability of our method to learn expressive deep kernel functions and compare it with SV-DKL (Wilson et al. [21]). For TT-GP we use our implementation availible at `https://anonimized-link`.

4.1 Standard Kernels

For testing our method with standard covariance functions we used a range of classification and regression tasks from UCI and LIBSVM archives and the Airline dataset, that is popular for testing scalable GP models (Hensman et al. [3], Hensman et al. [4], Wilson et al. [21], Cutajar et al. [2]) .

To obtain the Airline dataset we used the scripts[1] provided with Wilson et al. [21]. In UCI and LIBSVM datasets we randomly split the data with 80:20 proportions for train and test samples, unless test data is availible explicitly. For all datasets we first normalize the features X. For regression we also normalize the target variables y.

For SVI-GP and KLSP-GP we used the implementations provided in GPfLow (Matthews et al. [9]). For Airline dataset we provide results reported in the original paper (Hensman et al. [4]).

For our experiments we use a cluster of Intel Xeon E5-2698B v3 CPUs having 16 cores and 230 GB of RAM.

For YearPred, EEG and covtype datasets we used a d-dimensional linear embedding inside the RBF kernel for TT-GP, as the number D of features makes it impractical to set inducing inputs on a grid in

[1]`https://people.orie.cornell.edu/andrew/code/#SVDKL`

Table 1: Results of experiments with UCI, LIBSVM and Airline datasets. In the table acc. stands for r^2 for regression and accuracy for classification tasks. Here and below n is the size of the training set, and D is the dimensionality of the feature space; m is the number of inducing inputs used by the methods, r is TT-ranks of μ for TT-GP; t is the time per one pass over the data (epoch) in seconds; where provided, d is the dimensionality of linear embedding.
$*$ for KLSP-GP on Airline we provide results from the original article where the accuracy is given as a plot, and detailed information about experiment setup is not availible.

Dataset			SVI-GP / KLSP-GP			TT-GP				
Name	n	D	acc.	m	t (s)	acc.	m	r	d	t (s)
Powerplant	7654	4	0.94	200	10	0.95	35^4	30	-	5
Protein	36584	9	0.50	200	45	0.56	30^9	25	-	40
YearPred	$463K$	90	0.30	1000	597	0.32	10^6	10	6	105
Airline	$6M$	8	0.665*	-	-	0.694	20^8	15	-	5200
svmguide1	3089	4	0.967	200	4	0.969	20^4	15	-	1
EEG	11984	14	0.915	1000	18	0.908	12^{10}	15	10	10
covtype bin	465K	54	0.817	1000	320	0.852	10^6	10	6	172

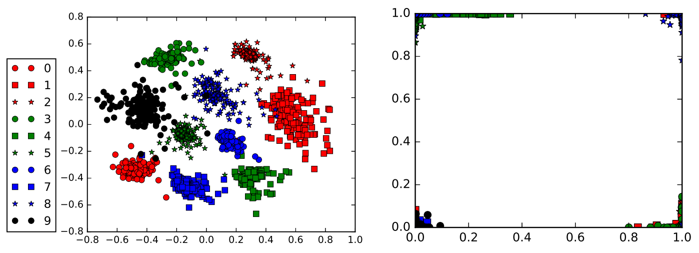
a D-dimensional space in this case. On all other datasets we used the RBF kernel on initial features. For KLSP-GP and SVI-GP we used RBF kernel in all cases.

Table 1 shows the results on different regression and classification tasks. We can see, that TT-GP is able to achieve better predictive quality on all datasets except EEG. We also note that the method is able to achieve good predictive performance with linear embedding, which makes it practical for a wide range of datasets.

4.2 Deep Kernels

4.2.1 Representation learning

We first explore the representation our model learns for data on the small Digits[2] dataset containing $n = 1797$ 8×8 images of handwritten digits. We used a TT-GP with a kernel based on a small fully-connected neural network with two hidden layers with 50 neurons each and $d = 2$ neurons in the output layer to obtain a 2-dimensional embedding. We trained the model to classify the digits to 10 classes corresponding to different digits. Fig. 1a shows the learned embedding. We also trained the same network standalone, adding another layer with 10 outputs and softmax activations. The embedding for this network is shown in fig. 1b.



(a) DNN with TT-GP (b) Plain DNN

Figure 1: Learned representation for Digits dataset

We can see that the stand-alone DNN with linear classifiers is unable to learn a good 2-dimensional embedding. On the other hand, using a flexible GP classifier our model learns to group objects of the same class into compact regions.

[2]`http://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html`

4.2.2 Classification tasks

To test our model with deep kernels we used Airline, CIFAR10 (Krizhevsky [7]) and MNIST (LeCun et al. [8]) datasets.

We will use the following notation to describe DNN architecture. fc(h) means a fully-connected layer with h neurons; conv($h\times w$, f) means a convolutional layer with f $h\times w$ filters; maxpool($h\times w$) means max-pooling with $h\times w$ kernel; ReLU means rectified linear unit activation function; BN means batch normalization (Ioffe and Szegedy [5]).

For the Airline dataset we used a DNN with 5 fully-connected layers with architecture fc(1000)-ReLU-fc(1000)-ReLU-fc(500)-ReLU-fc(50)-ReLU-fc(2). The same architecture was used in Wilson et al. [21] on this task.

On MNIST we used a convolutional DNN with the following architecture: conv(5×5, 32)-ReLU-maxpool(2×2)-conv(5×5, 64)-ReLU-maxpool(2×2)-fc(1024)-ReLU-fc(4).

On CIFAR10 we used an 9-layer convolutional neural network with the following architecture: conv(3×3, 128)-BN-ReLU-conv(3×3, 128)-BN-ReLU-maxpool(3×3)-conv(3×3, 256)-BN-ReLU-conv(3×3, 256)-BN-ReLU-maxpool(3×3)-conv(3×3, 256)-BN-ReLU-conv(3×3, 256)-BN-ReLU-maxpool(3×3)-fc(1536)-BN-ReLU-fc(512)-BN-ReLU-fc(9). We also use standard data augmentation techniques on this dataset with random cropping of 24×24 parts of the image, horizontal flipping, randomly adjusting brightness and contrast.

In all experiments we also add a BN without trainable mean and variance after the DNN output layer to project the outputs into the region where inducing inputs are placed. We use $m_0 = 10$ inducing inputs per dimension and set TT-ranks of μ to $r = 10$ for all three datasets.

For experiments with convolutional neural networks (on MNIST and CIFAR10), we used Nvidia Tesla K80 GPUs to train the model.

Table 2 shows the results of the experiments for our TT-GP with DNN kernel and SV-DKL. Note, that the comparison is not absolutely fair on CIFAR10 and MNIST datasets, as we didn't use the same exact architecture and preprocessing as Wilson et al. [21]. On Airline we used the same exact architecture and preprocessing as SV-DKL, and TT-GP achieves a higher accuracy on this dataset.

We also provide results of stand-alone DNNs for comparison. We used the same networks that were used in TT-GP kernels with the last linear layers replaced by layers with C outputs and softmax activations. Overall, we can see, that our model is able to achieve good predictive performance, improving the results of standalone DNN on Airline and MNIST.

Table 2: Results of experimetns with deep kernels. Here acc. is classification accuracy; C is the number of classes; d is the dimensionality of embedding learned by the model; t is the time per one pass over data (epoch) in seconds.

Dataset				SV-DKL	DNN		TT-GP		
Name	n	D	C	acc.	acc.	t (s)	acc.	d	t (s)
Airline	$6M$	8	2	0.781	0.780	1055	0.784	2	1375
CIFAR10	$50K$	32×32×3	10	0.770	0.915	166	0.909	9	220
MNIST	$60K$	28×28	10	0.992	0.993	23	0.994	10	64

We train all the models from random initialization without pretraining. We also tried using pretrained DNNs as initialization for our TT-GP model's kernel, which sometimes leads to faster convergence, but does not improve the final accuracy.

5 Discussion

We proposed TT-GP method for scalable inference in Gaussian process models for regression and classification. The proposed method is capable of using billions of inducing inputs, which is impossible for existing method. This allows us to improve the performance over state-of-the-art methods both with standard and deep kernels on several important benchmark datasets. Further, we believe, that our model provides a more natural and straightforward way of learning deep kernel functions, than the existing approaches.

Our preliminary experiments showed, that TT-GP is inferior in terms of uncertainty quatification compared to existing methods. We suspect that the reason for this is that we use covariance matrices Σ represented as Kronecker products over dimensions, which is rather restricting. We hope to alleviate this limitation by using Tensor Train format for Σ, and using approximations to compute it's inverse and determinant.

As a promising direction for future work we also consider training TT-GP with deep kernels incrementally, using the variational approximation of posterior distribution as a prior for new data. We also find it interesting to try using the low-dimensional (compared to embeddings learned by standard DNNs) embeddings learned by our model for transfer learning. Finally, we want to explore the performance of our model in different practical applications including hyper-parameter optimization.

References

[1] Guillaume Bouchard. Efficient bounds for the softmax function and applications to approximate inference in hybrid models. In *NIPS 2007 workshop for approximate Bayesian inference in continuous/hybrid systems*, 2007.

[2] Kurt Cutajar, Edwin V Bonilla, Pietro Michiardi, and Maurizio Filippone. Practical learning of deep gaussian processes via random fourier features. *arXiv preprint arXiv:1610.04386*, 2016.

[3] James Hensman, Nicolò Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 282–290. AUAI Press, 2013.

[4] James Hensman, Alexander G de G Matthews, and Zoubin Ghahramani. Scalable variational gaussian process classification. In *AISTATS*, 2015.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456, 2015.

[6] Robert Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981.

[7] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *arXiv preprint 1610.08733*, October 2016.

[10] Alexander Novikov, Anton Rodomanov, Anton Osokin, and Dmitry Vetrov. Putting mrfs on a tensor train. In *International Conference on Machine Learning*, pages 811–819, 2014.

[11] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

[12] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5): 2295–2317, 2011.

[13] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[14] Carl E. Rasmussen and Chris K. Williams. Gaussian processes for machine learning, 2006.

[15] Yunus Saatçi. *Scalable inference for structured Gaussian process models*. PhD thesis, University of Cambridge, 2012.

[16] Bernhard W Silverman. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–52, 1985.

[17] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257, 2006.

[18] Michalis K Titsias. Variational learning of inducing variables in sparse gaussian processes. In *AISTATS*, volume 5, pages 567–574, 2009.

[19] Christopher KI Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, pages 661–667. MIT press, 2000.

[20] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pages 1775–1784, 2015.

[21] Andrew G Wilson, Zhiting Hu, Ruslan R Salakhutdinov, and Eric P Xing. Stochastic variational deep kernel learning. In *Advances in Neural Information Processing Systems*, pages 2586–2594, 2016.

[22] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 370–378, 2016.