Scalable High-Order Gaussian Process Regression

Shandian Zhe, Wei Xing, Robert M. Kirby
School of Computing, University of Utah
zhe@cs.utah.edu, wxing@sci.utah.edu, kirby@cs.utah.edu

Abstract

While most Gaussian processes (GP) work focus on learning single-output functions, many applications, such as physical simulations and gene expressions prediction, require estimations of functions with many outputs. The number of outputs can be much larger than or comparable to the size of training samples. Existing multi-output GP models either are limited to low-dimensional outputs and restricted kernel choices, or assume oversimplified low-rank structures within the outputs. To address these issues, we propose HOGPR, a High-Order Gaussian Process Regression model, which can flexibly capture complex correlations among the outputs and scale up to a large number of outputs. Specifically, we tensorize the high-dimensional outputs, introducing latent coordinate features to index each tensor element (*i.e.,* output) and to capture their correlations. We then generalize a multilinear model to a hybrid of a GP and latent GP model. The model is endowed with a Kronecker product structure over the inputs and the latent features. Using the Kronecker product properties and tensor algebra, we are able to perform exact inference over millions of outputs. We show the advantage of the proposed model on several real-world applications.

1 Introduction

Gaussian processes (GPs) are important function learning models. Due to the nonparametric Bayesian nature, GPs can automatically capture the complexity of the functions underlying the data, and quantify the uncertainty. In the past few decades, numerous GP models/inference algorithms have been proposed for various tasks, such as face detection (Lu and Tang, 2015), collaborative filtering (Lawrence and Urtasun, 2009; Kim et al., 2016) and optimization (Mockus, 2012; Snoek et al., 2012). Most of the work essentially aim to learn a single-output function from data.

Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

However, many applications require us to learn a function with many outputs, from a relatively small number of training samples. For instance, in computational physics, given a set of physical conditions (*i.e.,* input), we are interested to obtain the corresponding field (*e.g.,* pressures or velocities) represented by a high-dimensional vector. The computation of the field, typically through the finite element/difference method (Zienkiewicz et al., 1977; Mitchell and Griffiths, 1980), is very expensive. To minimize the cost, it is urgent to learn a proxy of this low-to-high mapping, based upon a limited number of simulation examples.

One can simply learn for each output an independent GP model. However, among these outputs are often strong and complex correlations. Ignoring the valuable correlations is likely to result in inferior performance, especially on small training samples. To address this problem, several multi-output GP models were proposed to capture the output correlations. For example, the convolved GPs (Higdon, 2002; Boyle and Frean, 2005) model the covariance between outputs through convolution operations. A class of other methods, such as PCA-GP (Higdon et al., 2008) and IsoMap-GP (Xing et al., 2015) assume a low-rank structure within the outputs, and model the outputs as the linear combination of a group of (fixed) bases; typically, the coefficients are estimated by separate GP regression models.

Despite their success, the aforementioned work either are limited to low-dimensional outputs and restricted kernels, or use oversimplified correlation structures. The time complexity of the convolved GPs is $\mathcal{O}\big((Nd)^3\big)$ where N and d are the numbers of training samples and outputs. Hence, they are infeasible even for a tiny number of outputs, *e.g.,* 100 outputs/100 samples, and have to seek for sparse approximations (Alvarez and Lawrence, 2009). In addition, they have to use easy smoothing/base kernels (*e.g.,* Gaussians and delta) to obtain analytical convolutions. PCA-GP (Higdon et al., 2008) and the congenic methods, while being scalable to a large-number of outputs, impose linear correlation structures within the outputs, which might be too restricted, and inadequate to capture more complex correlations.

To address these issues, we propose HOGPR, a high-order Gaussian process regression model, which is not only flexible enough to capture complex output correlations, but

also scalable to very high-dimensional outputs (without any sparse approximations). Specifically, we first organize the outputs into a tensor, and for each coordinate in tensor modes, we introduce a latent feature vector; in this way, we can index each tensor element, *i.e.,* output, by a set of coordinate feature vectors. Next, we use a multilinear model to combine the input and latent coordinate features to generate the tensorized outputs. We further generalize the multilinear model via feature mapping and kernel tricks into a hybrid of a GP and latent GP model. The correlations between different outputs are naturally captured by the covariance of their latent coordinate features. The model is endowed with a Kronecker product structure over the inputs and the latent features in each mode. Using Kronecker product properties and tensor algebra, we develop an exact inference algorithm that can linearly scale to the number of outputs.

For evaluation, we first examined HOGPR on three real-world small datasets with hundreds of samples and hundreds/thousands of outputs for each sample. HOGPR outperforms sparse convolved GP and low-rank multi-output GP regression methods in most cases. We then applied HOGPR in topology structure design — HOGPR predicted much better structures than the competing methods; on various training sizes, HOGPR often improved the overall prediction accuracy upon the competing methods by a large margin. Finally, we applied HOGPR in physical simulations. We used HOGPR to predict a pressure field with one million outputs but only hundreds of training samples. HOGPR often outperforms the competing low-rank methods.

2 Background

Single-output Gaussian processes. We first review GP regression to learn a single-output function. Given an observed dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ where each $\mathbf{x}_i$ is a p dimensional input vector, and y_i the observed scalar output, we aim to learn a function $f : \mathbb{R}^p \to \mathbb{R}$. GP regression assumes the finite set of function values on $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^\top$, namely $\mathbf{f} = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)]^\top$, follow a multivariate Gaussian distribution, $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{m}, \mathbf{K})$ where $\mathbf{m} = [m(\mathbf{x}_1), \dots, m(\mathbf{x}_n)]^\top$ and $\mathbf{K}$ is the covariance matrix. Here, $m(\cdot)$ is the mean function, usually set to 0. Each element of $\mathbf{K}$ is defined as a kernel function of the corresponding inputs: $[\mathbf{K}]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$.The observed outputs $\mathbf{y} = [y_1, \dots, y_N]^\top$ are assumed to be corrupted, *e.g.,* by some Gaussian random noises, $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \tau^{-1}\mathbf{I})$ where τ is the inverse noise variance. We can marginalize out $\mathbf{f}$ to obtain the marginal likelihood of $\mathbf{y}$, $p(\mathbf{y}|\mathbf{X}) = \mathcal{N}\big(\mathbf{y}|\mathbf{0}, \mathbf{K} + \tau^{-1}\mathbf{I}\big)$.

The inference of the GP regression is to estimate the covariance (or kernel) parameters and the inverse noise variance τ. We can optimize these parameters by maximizing the marginal likelihood. To this end, we need to compute the $N \times N$ covariance matrix, taking $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$ time and space complexity, respectively. To predict the output for a test input $\mathbf{x}^*$, we use the conditional Gaussian distribution, $p\big(f(\mathbf{x}*)|\mathbf{x}^*, \mathbf{X}, \mathbf{y}\big) = \mathcal{N}\big(f^*|\mu(\mathbf{x}^*), v(\mathbf{x}^*)\big)$, where $\mu(\mathbf{x}^*) = \mathbf{k}_*^\top(\mathbf{K}+\tau^{-1}\mathbf{I})^{-1}\mathbf{y}$, $v(\mathbf{x}^*) = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}_*^\top(\mathbf{K}+\tau^{-1}\mathbf{I})^{-1}\mathbf{k}_*$ and $\mathbf{k}_* = [k(\mathbf{x}^*, \mathbf{x}_1), \dots, k(\mathbf{x}^*, \mathbf{x}_N)]^\top$.

Multi-output Gaussian processes. Many applications require one to learn a function with multiple outputs. In this case, one input $\mathbf{x}$ will correspond to d outputs $[f_1(\mathbf{x}), \dots, f_d(\mathbf{x})]^\top (d > 1)$. Several seminal works have been proposed for multi-output regression. The convolved GPs (Higdon, 2002; Boyle and Frean, 2005) assume a set of latent functions, and each output $f_q(\mathbf{x})$ is generated from a convolution between a smoothing kernel and the latent functions. Using easy smoothing kernels (typically Gaussians) and covariances for the latent functions, we can derive a closed-form covariance function between arbitrarily two outputs for two inputs $\mathbf{x}_i, \mathbf{x}_j$, *i.e.,* $\mathrm{cov}\big(f_q(\mathbf{x}_i), f_r(\mathbf{x}_j)\big)(1 \le q, r \le d)$. Then we can combine all the input and (scalar) output instances $\{(\mathbf{x}_i, y_{iq})\}_{1\le i\le N, 1\le q\le d}$ to learn a single-output GP regression model. Because the number of training instances becomes Nd, the time and space complexity are $\mathcal{O}((Nd)^3)$ and $\mathcal{O}((Nd)^2)$, respectively.

Another line of research model the output correlations by imposing a low-rank structure (Higdon et al., 2008; Xing et al., 2016, 2015). Typically, they introduce a set of fixed bases, $\{\mathbf{b}_1, \dots, \mathbf{b}_K\} (K \ll d)$, and assume that $\mathbf{y} = \sum_{k=1}^K \alpha_k(\mathbf{x})\mathbf{b}_k$. Each coefficient function $\alpha_k(\mathbf{x})$ is learned via a separate GP regression model. For example, PCA-GP (Higdon et al., 2008) obtains the bases and training coefficients from Principal Component Analysis (PCA) on the training output matrix.

3 Model

In many applications, we need to learn a function with many outputs yet from a relatively small set of training samples. The reason is that when the output dimension grows higher, collecting training instances often is more expensive. For example, in physical simulations, a set of boundary conditions usually correspond to a vector field. The simulations with higher fidelities (resulting in higher dimensional fields) are much more costly due to the explosion of computation overheads. Thus, it is often the case that we are left with a few hundred samples to learn a function with thousands or even millions of outputs. To enhance the function learning, it is critical to exploit the (possibly) strong and complex correlations among the outputs.

Despite the elegance and success of the existing multi-output GP models, they might be restricted by the limited scalability to the output dimension and kernel choices, or over-simplified correlation structures, especially in the "small N, large d" scenario. The time complexity of the convolved GPs is $\mathcal{O}\big((Nd)^3\big)$. Hence, even at a tiny scale, *e.g.,* 100 outputs/100 samples, the convolved GPs will be infeasible, and have to use inexact, sparse approximations (Alvarez and Lawrence, 2009). Moreover, to ensure an analytical covariance function, the convolved GPs have to choose easy but

perhaps less expressive smoothing/base kernels for tractable convolution operations. PCA-GP and other methods, although are very efficient to handle large output dimensions, they impose a linear correlation structure within the outputs. This might be too restrictive and inflexible to capture more complex correlations among the outputs in practice.

To address these issues, we propose HOGPR, a high-order GP regression model that is both more flexible to capture complex output correlations and scalable to a large number of outputs. We present HOGPR as follows.

3.1 A Motivated Example to Model Output Correlations

To introduce our model, we start with a motivated example, that is, predicting the pixel values of an image, given the input $\mathbf{x}$. Obviously, each pixel (*i.e.,* output) is indexed by a pair of coordinates (c_1, c_2) (in a 2-dimensional space). Let us denote each output by $y_{c_1 c_2}$. Then how to model the correlations among these outputs? We can look into their coordinates — intuitively, the closer the two pixels, the more correlated they are. Hence, we can model the output correlations via a kernel (similarity) function of their coordinates, *e.g.,* $h([c_1, c_2], [c_1', c_2']) = \exp(-\frac{1}{\sigma^2}\|(c_1, c_2) - (c_1', c_2')\|^2)$.

3.2 High-Order Gaussian Process Regression

In general, however, the function outputs do not represent image pixels; we may not have explicit coordinates that index the outputs and reflect their correlations. To address this issue, we rearrange all the outputs into a multidimensional space (*i.e.,* tensor space), and introduce latent coordinate features to index the outputs and to capture their correlations. Then we combine the input and latent features to build a GP regression model for the rearranged outputs, *i.e.,* HOGPR.

Specifically, we organize the outputs as a Q-mode tensor, $\mathcal{Y} \in \mathbb{R}^{d_1 \times \ldots \times d_Q}$. Suppose there are d outputs for each input, then $d = \prod_{k=1}^{Q} d_k$. Each output in this tensor space can be located by a tuple of Q coordinates $\mathbf{c} = (c_1, \ldots, c_Q)(1 \leq c_k \leq d_k, 1 \leq k \leq Q)$. Let us denote it by $y_{\mathbf{c}}$. Note that we cannot assume outputs at nearby coordinates have strong correlations, because these (discrete) coordinates can be assigned arbitrarily. To capture the output correlations, we introduce a latent feature vector $\mathbf{v}_{c_k}^k$ for each coordinate c_k (in mode k). Then we can index each output $y_{\mathbf{c}}$ by Q coordinate feature vectors $\{\mathbf{v}_{c_1}^1, \ldots, \mathbf{v}_{c_Q}^Q\}$. We will use these coordinate features to model the output correlations.

Next, we combine the original inputs and the latent coordinate features to build our HOGPR model. We will first introduce a Bayesian multilinear regression model and then generalize it to HOGPR via feature mapping and kernel tricks. Specifically, given the training dataset $\mathcal{D}$, we can use a $Q+1$-mode tensor $\mathcal{Z} \in \mathbb{R}^{d_1 \times \cdots \times d_Q \times N}$ to represent all the observed outputs for the N training inputs. Each slice $\mathcal{Z}(:, \ldots, :, i)$ is the output tensor $\mathcal{Y}_i$ for input i. Hence the training data $\mathcal{D}$ can be represented by $(\mathbf{X}, \mathcal{Z})$, where $\mathbf{X}$ is the $N \times p$ input matrix and $\mathcal{Z}$ output tensor. We can stack the latent feature vectors into Q matrices, $\mathcal{V} = \{\mathbf{V}^1, \ldots, \mathbf{V}^Q\}$ where the rows of each $\mathbf{V}^k (1 \leq k \leq Q)$ are the feature vectors for the coordinates in mode k. Hence each $\mathbf{V}^k$ is a $d_k \times r_k$ matrix, where r_k is the dimension of the feature vectors in mode k. In our multilinear regression model, we first sample a weight tensor $\mathcal{W} \in \mathbb{R}^{r_1 \times \ldots \times r_Q \times p}$ from a standard Gaussian prior, $p(\mathcal{W}) = \mathcal{N}\big(\mathrm{vec}(\mathcal{W})|\mathbf{0}, \mathbf{I}\big)$. Given $\mathcal{W}$, we sample the output tensor $\mathcal{Z}$ from $p(\mathcal{Z}|\mathbf{X}, \mathcal{V}) = \mathcal{N}\big(\mathrm{vec}(\mathcal{Z})|\mathrm{vec}(\mathcal{F}), \tau^{-1}\mathbf{I}\big)$, where τ is the inverse variance, and

$$\mathcal{F} = \mathbf{W} \times_1 \mathbf{V}^1 \ldots \times_Q \mathbf{V}^Q \times_{Q+1} \mathbf{X}. \tag{1}$$

Here $\times_k$ is the tensor-matrix product at mode k (Kolda, 2006). Given a tensor $\mathcal{M} \in \mathbb{R}^{r_1 \times \ldots \times r_Q}$ and a matrix $\mathbf{U} \in \mathbb{R}^{s \times t}$, we can multiply $\mathcal{M}$ by $\mathbf{U}$ at mode k when $r_k = t$. The result is an $r_1 \times \ldots \times r_{k-1} \times s \times r_{k+1} \times \ldots \times r_Q$ tensor. Each entry is computed by $(\mathcal{M} \times_k \mathbf{U})_{i_1 \ldots i_{k-1} j i_{k+1} \ldots i_Q} = \sum_{i_k=1}^{r_k} m_{i_1 \ldots i_Q} u_{j i_k}$. The multilinear form in (1) is referred to as the Tucker operator (Kolda, 2006). An important property of the Tucker operator is

$$\mathrm{vec}(\mathcal{F}) = (\mathbf{V}^1 \otimes \ldots \otimes \mathbf{V}^Q \otimes \mathbf{X})\mathrm{vec}(\mathcal{W}). \tag{2}$$

The joint probability of our multilinear regression model is

$$p(\mathcal{Z}, \mathcal{W}|\mathbf{X}, \mathcal{V}, \tau) = \mathcal{N}\big(\mathrm{vec}(\mathcal{W})|\mathbf{0}, \mathbf{I}\big) \cdot \mathcal{N}\big(\mathrm{vec}(\mathcal{Z})|\mathrm{vec}(\mathcal{F}), \tau^{-1}\mathbf{I}\big). \tag{3}$$

This model combines the original inputs $\mathbf{X}$ and the latent coordinate features $\mathcal{V}$ to generate all the outputs, through the multilinear Tucker operator in (1).

To generalize (3) to a GP model so as to enable more powerful regression, say, nonlinear regression, we first marginalize out the weight tensor $\mathcal{W}$, to obtain the marginal distribution of the output tensor $\mathcal{Z}$. Obviously, it is still a Gaussian distribution, with the mean $\mathbf{0}$ and the covariance $\mathbb{E}(\mathrm{vec}(\mathcal{F})\mathrm{vec}(\mathcal{F})^\top) + \tau^{-1}\mathbf{I}$. According to (2), we have $\mathbb{E}\big(\mathrm{vec}(\mathcal{F})\mathrm{vec}(\mathcal{F})^\top\big) = (\mathbf{V}^1 \otimes \ldots \otimes \mathbf{V}^Q \otimes \mathbf{X})\mathbb{E}\big(\mathrm{vec}(\mathcal{W})\mathrm{vec}(\mathcal{W})^\top\big)(\mathbf{V}^1 \otimes \ldots \otimes \mathbf{V}^Q \otimes \mathbf{X})^\top = \mathbf{V}^1(\mathbf{V}^1)^\top \otimes \ldots \otimes \mathbf{V}^Q(\mathbf{V}^Q)^\top \otimes \mathbf{X}\mathbf{X}^\top$. The marginal probability of $\mathcal{Z}$ is then given by

$$p(\mathcal{Z}|\mathbf{X}, \mathcal{V}, \tau) = \mathcal{N}\big(\mathrm{vec}(\mathcal{Z})|\mathbf{0}, \mathbf{\Sigma} + \tau^{-1}\mathbf{I}\big) \tag{4}$$

where $\mathbf{\Sigma} = \mathbf{V}^1(\mathbf{V}^1)^\top \otimes \ldots \otimes \mathbf{V}^Q(\mathbf{V}^Q)^\top \otimes \mathbf{X}\mathbf{X}^\top$.

We then perform a (nonlinear) feature mapping of the inputs $\mathbf{X}$ and the latent coordinate features $\mathcal{V}$ in (4). Take $\mathbf{V}^1$ as an example. We replace each row $\mathbf{v}_j^1$ in $\mathbf{V}^1$ by the mapped feature vector $\phi(\mathbf{v}_j^1)$. Denote by $\phi(\mathbf{V}^1)$ the mapped feature matrix. Then $\mathbf{V}^1(\mathbf{V}^1)^\top$ in $\mathbf{\Sigma}$ becomes $\phi(\mathbf{V}^1)\phi(\mathbf{V}^1)^\top$ — the elements are the inner products between the mapped feature vectors. We now apply the kernel trick — we replace $\phi(\mathbf{V}^1)\phi(\mathbf{V}^1)^\top$ by a kernel matrix $\mathbf{K}^1$, where each element

$[\mathbf{K}^1]_{i,j} = k_1(\mathbf{v}_i^1, \mathbf{v}_j^1)$ is a kernel function that performs implicit feature mapping and inner product. Now we have

$$p(\mathcal{Z}|\mathbf{X}, \mathcal{V}, \tau) = \mathcal{N}\big(\mathrm{vec}(\mathcal{Z})|\mathbf{0}, \mathbf{K}^1 \otimes \ldots \otimes \mathbf{K}^Q \otimes \mathbf{K} + \tau^{-1}\mathbf{I}\big) \quad (5)$$

where $\{\mathbf{K}^1, \ldots, \mathbf{K}^Q\}$ are the kernel matrices for the latent coordinate features $\{\mathbf{V}^1, \ldots, \mathbf{V}^Q\}$, and $\mathbf{K}$ for the input $\mathbf{X}$.

The probability in (5) defines our HOGPR model — a GP regression model for the tensorized outputs $\mathcal{Z}$. If we set $Q = 2$, we return to the image case. However, we introduce latent coordinate features rather than use the coordinates themselves to capture the output correlations, because in general these coordinates can be freely assigned and cannot reflect the correlation strengths between the outputs. In our model, the covariance between any two outputs $y_{\mathbf{c}}$ and $y_{\mathbf{c}'}$ for samples i and j is given by $\mathrm{cov}\big(\mathrm{y}_{\mathbf{c}}(\mathbf{x}_\mathrm{i}), \mathrm{y}_{\mathbf{c}'}(\mathbf{x}_\mathrm{j})\big) = k(\mathbf{x}_i, \mathbf{x}_j) \prod_{j=1}^{Q} k_j(\mathbf{v}_{c_j}^j, \mathbf{v}_{c'_j}^j)$.

To capture various complex correlations among the outputs, we can choose arbitrary kernel functions for the coordinate features $\{\mathbf{V}^1, \ldots, \mathbf{V}^Q\}$ and the input $\mathbf{X}$. Compared with PCA-GP and other low-rank approaches, we do not impose any simplified, linear structures. Compared with the convolved GPs, we do not need to restrict the kernel forms to ensure the tractability of the convolution operations. Furthermore, the covariance of HOGPR (5) is endowed with a Kronecker product structure over the kernel matrices. This enables exact and scalable inference to a large number of outputs, as we will present in the following section.

4 Algorithm

We now present the model inference algorithm. Note that HOGPR is a hybrid of a GP and latent GP model — each output $y_{\mathbf{c}}$ is associated with an observed input vector $\mathbf{x}$, and Q latent coordinate feature vectors $\{\mathbf{v}_{c_1}^1, \ldots, \mathbf{v}_{c_K}^K\}$. Hence, given an observed dataset $\mathcal{D} = (\mathbf{X}, \mathcal{Z})$, the inference needs to estimate all the latent features in the Q-mode tensor space, $\mathcal{V} = \{\mathbf{V}^1, \ldots, \mathbf{V}^Q\}$, as well as the kernel parameters for the latent features and the original inputs, and the inverse variance τ. We estimate these parameters by maximizing the log likelihood of the model,

$$\mathcal{L} = \log\big(p(\mathcal{Z}|\mathbf{X}, \mathcal{V}, \tau)\big) = -\frac{1}{2}\log|\mathbf{S}| - \frac{1}{2}\mathrm{vec}(\mathcal{Z})^\top \mathbf{S}^{-1}\mathrm{vec}(\mathcal{Z}) + \mathrm{const}, \quad (6)$$

where $\mathbf{S} = \mathbf{K}^1 \otimes \ldots \otimes \mathbf{K}^Q \otimes \mathbf{K} + \tau^{-1}\mathbf{I}$. The major challenge is the covariance matrix $\mathbf{S}$, of size $Nd \times Nd$. When the number of outputs is large, say, one million, the covariance matrix $\mathbf{S}$ will be infeasible to compute and so the inverse and log determinant. However, by exploiting the Kronecker product in $\mathbf{S}$, we will be able to efficiently compute the log likelihood and its gradient for model estimation.

4.1 Efficient Likelihood and Gradient Computation

We first represent each kernel matrix in $\mathbf{S}$ (see (6)) with its eigendecomposition, $\mathbf{K} = \mathbf{U}^\top \mathrm{diag}(\boldsymbol{\lambda})\mathbf{U}^\top$ and $\mathbf{K}^j = \mathbf{U}_j \mathrm{diag}(\boldsymbol{\lambda}_j)\mathbf{U}_j^\top$. Then we have $\mathbf{S} = \mathbf{U}_1 \mathrm{diag}(\boldsymbol{\lambda}_1)\mathbf{U}_1^\top \otimes \ldots \otimes \mathbf{U}_Q \mathrm{diag}(\boldsymbol{\lambda}_Q)\mathbf{U}_Q^\top \otimes \mathbf{U}^\top \mathrm{diag}(\boldsymbol{\lambda})\mathbf{U}^\top + \tau^{-1}\mathbf{I}$. With the Kronecker product property, we can derive that

$$\mathbf{S} = \mathbf{P}\boldsymbol{\Lambda}\mathbf{P}^\top + \tau^{-1}\mathbf{I} \quad (7)$$

where $P = \mathbf{U}_1 \otimes \ldots \otimes \mathbf{U}_Q \otimes \mathbf{U}$ and $\boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{\lambda}_1 \otimes \ldots \otimes \boldsymbol{\lambda}_Q \otimes \boldsymbol{\lambda})$. Note that since $\mathbf{U}$ and each $\mathbf{U}_k (1 \le k \le Q)$ are eigenvectors and orthogonal, their Kronecker product is orthogonal as well, *i.e.,* $\mathbf{P}^\top\mathbf{P} = \mathbf{P}\mathbf{P}^\top = \mathbf{I}$. Hence, we can further obtain

$$\mathbf{S} = \mathbf{P}(\boldsymbol{\Lambda} + \tau^{-1}\mathbf{I})\mathbf{P}^\top, \;\; \mathbf{S}^{-1} = \mathbf{P}(\boldsymbol{\Lambda} + \tau^{-1}\mathbf{I})^{-1}\mathbf{P}^\top. \quad (8)$$

Using (8), we can compute the likelihood (6) very efficiently. First, we find that $\log|\mathbf{S}| = \log|\mathbf{P}(\boldsymbol{\Lambda} + \tau^{-1}\mathbf{I})\mathbf{P}^\top| = \log|\mathbf{P}^\top\mathbf{P}(\boldsymbol{\Lambda} + \tau^{-1}\mathbf{I})| = \log|\boldsymbol{\Lambda} + \tau^{-1}|$. Note that $\boldsymbol{\Lambda} + \tau^{-1}\mathbf{I}$ is a diagonal matrix and we only need to know the Nd diagonal elements. We can compute a corresponding $d_1 \times \ldots \times d_Q \times N$ tensor,

$$\mathcal{A} = \boldsymbol{\lambda}_1 \circ \ldots \circ \boldsymbol{\lambda}_Q \circ \boldsymbol{\lambda} + \tau^{-1}\mathbb{1}, \quad (9)$$

where $\mathbb{1}$ is a tensor of full ones, and $\boldsymbol{\lambda}_1 \circ \ldots \circ \boldsymbol{\lambda}_Q \circ \boldsymbol{\lambda}$ is the Kruskal operator (Kolda, 2006) that generates a tensor where the element at $\mathbf{c} = (c_1, \ldots, c_{Q+1})$ is $\lambda_{c_{Q+1}} \prod_{k=1}^{Q} \lambda_{k_{c_k}}$. To compute $\log|\boldsymbol{\Lambda} + \tau^{-1}|$, we can sum over the logarithm of all the elements in $\mathcal{A}$. In this way, computing $\log|\mathbf{S}|$ only takes $\mathcal{O}(Nd)$ time complexity.

Next, we observe that $\mathrm{vec}(\mathcal{Z})^\top \mathbf{S}^{-1}\mathrm{vec}(\mathcal{Z}) = \mathbf{b}^\top\mathbf{b}$ where $\mathbf{b} = \mathbf{S}^{-\frac{1}{2}}\mathrm{vec}(\mathcal{Z}) = \mathbf{P}(\boldsymbol{\Lambda} + \tau^{-1})^{-\frac{1}{2}}\mathbf{P}^\top \cdot \mathrm{vec}(\mathcal{Z})$. Since $\mathbf{P}$ is a Kronecker product, we can recursively apply the property of the Tucker operator (see (2)) to compute $\mathbf{b}$:

$$\begin{aligned} \mathcal{T}_1 &= \mathcal{Z} \times_1 \mathbf{U}_1^\top \ldots \times_Q \mathbf{U}_Q^\top \times_{Q+1} \mathbf{U}^\top, \\ \mathcal{T}_2 &= \mathcal{T}_1 \odot \mathcal{A}^{\cdot -\frac{1}{2}}, \\ \mathcal{T}_3 &= \mathcal{T}_2 \times_1 \mathbf{U}_1 \ldots \times_Q \mathbf{U}_Q \times_{Q+1} \mathbf{U}, \\ \mathbf{b} &= \mathrm{vec}(\mathcal{T}_3), \end{aligned} \quad (10)$$

where $\odot$ is the element-wise product, and $(\cdot)^{\cdot -\frac{1}{2}}$ is to take the power of $-\frac{1}{2}$ element-wisely. Hence, $\mathcal{A}^{\cdot -\frac{1}{2}}$ correspond to the diagonal of $(\boldsymbol{\Lambda} + \tau^{-1}\mathbf{I})^{-\frac{1}{2}}$. The time complexity of (10) is $\mathcal{O}\big(Nd(\sum_{k=1}^{Q} d_k + N)\big)$.

Now, we consider the gradient calculation. Since all the parameters — $\mathcal{V}, \tau$ and the kernel parameters — are only included in the covariance $\mathbf{S}$, we can derive the derivative for $\mathbf{S}$ first and then apply the chain rule to calculate the gradient w.r.t to different parameters. It is easy to obtain

$$\nabla\mathcal{L} = -\frac{1}{2}\mathrm{tr}(\mathbf{S}^{-1}\nabla\mathbf{S}) + \frac{1}{2}\mathbf{g}^\top\nabla\mathbf{S}\mathbf{g} \quad (11)$$

where $\mathbf{g} = \mathbf{S}^{-1}\mathrm{vec}(\mathcal{Z})$.

We first consider the derivative for the inverse variance τ. Since $\nabla\mathbf{S} = -\tau^{-2}\mathbf{I}\nabla\tau$, we can obtain

$$\frac{\nabla\mathcal{L}}{\nabla\tau} = \frac{1}{2}\tau^{-2}\big(\mathrm{tr}(\mathbf{S}^{-1}) - \mathbf{g}^\top\mathbf{g}\big). \quad (12)$$

According to (8), $\mathrm{tr}(\mathbf{S}^{-1}) = \mathrm{tr}\big((\mathbf{\Lambda} + \tau^{-1}\mathbf{I})^{-1}\big)$. Hence, to compute this trace term, we can simply sum over all the elements in the corresponding tensor $\mathcal{A}^{\cdot -1}$. Furthermore, the computation of $\mathbf{g}$ is very similar to that of $\mathbf{b}$ (see (10)), and has the same complexity.

We then consider the derivative for all the other parameters. They are included in the kernel matrices. We first derive the derivative w.r.t each kernel matrix, and apply the chain rule to compute the derivatives w.r.t the latent coordinate features $\mathcal{V}$ and kernel parameters. Take $\mathbf{K}^1$ as an example. We observe that $\nabla\mathbf{S} = \nabla\mathbf{K}^1 \otimes \mathbf{K}^2 \otimes \ldots \otimes \mathbf{K}^Q \otimes \mathbf{K} = \mathbf{U}_1(\mathbf{U}_1^\top \nabla\mathbf{K}^1\mathbf{U}_1)\mathbf{U}_1^\top \otimes \mathbf{U}_2\mathrm{diag}(\boldsymbol{\lambda}_2)\mathbf{U}_2^\top \otimes \ldots \otimes \mathbf{U}_Q\mathrm{diag}(\boldsymbol{\lambda}_Q)\mathbf{U}_Q^\top \otimes \mathbf{U}\mathrm{diag}(\boldsymbol{\lambda})\mathbf{U}^\top = \mathbf{P}\big(\mathbf{U}_1^\top\nabla\mathbf{K}^1\mathbf{U}_1 \otimes \mathrm{diag}(\boldsymbol{\lambda}_2) \otimes \ldots \otimes \mathrm{diag}(\boldsymbol{\lambda}_Q) \otimes \mathrm{diag}(\boldsymbol{\lambda})\big)\mathbf{P}^\top$. Combining with (8), we can derive that $\mathrm{tr}(\mathbf{S}^{-1}\nabla\mathbf{S}) = \mathrm{tr}\big((\mathbf{\Lambda} + \tau^{-1}\mathbf{I})^{-1}(\mathbf{U}_1^\top\nabla\mathbf{K}^1\mathbf{U}_1 \otimes \mathrm{diag}(\boldsymbol{\lambda}_2) \otimes \ldots \otimes \mathrm{diag}(\boldsymbol{\lambda}_Q) \otimes \mathrm{diag}(\boldsymbol{\lambda}))\big) = \big(\mathrm{diag}^\top(\mathbf{U}_1^\top\nabla\mathbf{K}^1\mathbf{U}_1) \otimes \boldsymbol{\lambda}_2^\top \otimes \ldots \otimes \boldsymbol{\lambda}_Q^\top \otimes \boldsymbol{\lambda}^\top\big)\mathrm{vec}(\mathcal{A}^{\cdot -1})$. Using the property of the Tucker operator again, we have

$$\mathrm{tr}(\mathbf{S}^{-1}\nabla\mathbf{S}) = \mathrm{vec}\big(\mathcal{A}^{\cdot -1} \times_1 \mathrm{diag}^\top(\mathbf{U}_1^\top\nabla\mathbf{K}^1\mathbf{U}_1) \times_2 \boldsymbol{\lambda}_2^\top \ldots \times_Q \boldsymbol{\lambda}_Q^\top \times_{Q+1} \boldsymbol{\lambda}^\top\big) = \mathrm{tr}(\mathbf{U}_1\mathrm{diag}(\mathbf{b}_1)\mathbf{U}_1^\top\nabla\mathbf{K}^1), \quad (13)$$

where $\mathbf{b}_1 = \mathcal{A}^{\cdot -1} \times_2 \boldsymbol{\lambda}_2^\top \ldots \times_Q \boldsymbol{\lambda}_Q^\top \times_{Q+1} \boldsymbol{\lambda}^\top$. Note that after this multiplication, $\mathbf{b}_1$ becomes a d_1 dimensional vector. The cost of (13) (including applying the chain-rule to calculate the derivatives of kernel parameters and/or latent coordinate features) is dominated by the calculation of $\mathbf{b}_1$, and only takes $\mathcal{O}(Nd)$ time complexity.

We then look into $\mathbf{g}^\top\nabla\mathbf{S}\mathbf{g}$ in (11). Combing (8), Tucker and Kruskal operators, we follow the similar idea as above to obtain that $\mathbf{g}^\top\nabla\mathbf{S}\mathbf{g} = \mathrm{vec}(\mathcal{G})^\top\mathrm{vec}(\mathcal{H} \times_1 \mathbf{U}_1^\top\nabla\mathbf{K}^1\mathbf{U}_1)$ where $\mathcal{G} = \mathcal{A}^{\cdot -1} \odot (\mathcal{Z} \times_1 \mathbf{U}_1^\top \ldots \times_Q \mathbf{U}_Q^\top \times_{Q+1} \mathbf{U}^\top)$, and $\mathcal{H} = \mathcal{G} \times_2 \mathrm{diag}(\boldsymbol{\lambda}_2) \ldots \times_Q \mathrm{diag}(\boldsymbol{\lambda}_Q) \times_{Q+1} \mathrm{diag}(\boldsymbol{\lambda})$. We further unfold $\mathcal{G}$ and $\mathcal{H}$ at mode 1 to obtain two $d_1 \times (N\prod_{k=2}^{Q+1} d_k)$ matrices, $\mathbf{G}$ and $\mathbf{H}$. Then we have

$$\mathbf{g}^\top\nabla\mathbf{S}\mathbf{g} = \mathrm{tr}(\mathbf{U}_1\mathbf{G}\mathbf{H}^\top\mathbf{U}_1^\top\nabla\mathbf{K}^1). \quad (14)$$

The cost of (14) is obviously dominated by the computation of $\mathbf{G}$, $\mathbf{H}$ and their product, and the time complexity is $\mathcal{O}\big(Nd(\sum_{k=1}^Q d_k + N)\big)$. We use the same procedure to derive the derivatives for all the other kernel matrices, apply the chain rule to calculate the derivatives of the kernel parameters and latent coordinate features, and finally add them together to obtain the gradient of the model likelihood.

We can use any gradient-based optimization algorithm to maximize the model likelihood for inference. We used L-BFGS throughout our experiments.

4.2 Prediction

Given a new input $\mathbf{x}^*$, the predictive distribution of the d outputs, tensorized as $\mathcal{Y}^*$, is computed through a conditional Gaussian distribution,

$$p(\mathcal{Y}^*|\mathbf{x}^*, \mathcal{D}) = \mathcal{N}\big(\mathrm{vec}(\mathcal{Y}^*)|\boldsymbol{\mu}^*, \mathbf{\Sigma}^*\big)$$

where $\boldsymbol{\mu}^* = (\mathbf{K}^1 \otimes \ldots \otimes \mathbf{K}^Q \otimes \mathbf{k}_*)\mathbf{g}$, $\mathbf{\Sigma}^* = (\mathbf{K}^1 \otimes \ldots \otimes \mathbf{K}^Q)k(\mathbf{x}^*, \mathbf{x}^*) - (\mathbf{K}^1 \otimes \ldots \otimes \mathbf{K}^Q \otimes \mathbf{k}_*)\mathbf{S}^{-1}(\mathbf{K}^1 \otimes \ldots \otimes \mathbf{K}^Q \otimes \mathbf{k}_*^\top)$, and $\mathbf{k}_* = [k(\mathbf{x}^*, \mathbf{x}_1), \ldots, k(\mathbf{x}^*, \mathbf{x}_N)]$.

We can use the Tucker operator to efficiently compute the predictive mean $\boldsymbol{\mu}^*$. However, when d is large, it is not practical to compute the $d \times d$ predictive covariance matrix $\mathbf{\Sigma}^*$. Nonetheless, we can compute the variance of each individual output efficiently. Specifically, using (8) and the eigendecomposition of each kernel matrix, we can derive that $\mathbf{\Sigma}^* = (\mathbf{K}^1 \otimes \ldots \otimes \mathbf{K}^Q)k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{L}\mathbf{L}^\top$ where $\mathbf{L} = \big(\mathbf{U}_1 \otimes \ldots \otimes \mathbf{U}_Q \otimes \mathbf{k}_*\mathbf{K}^{-1}\mathbf{U}\big)\big(\mathbf{\Lambda}(\mathbf{\Lambda} + \tau^{-1}\mathbf{I})^{-\frac{1}{2}}\big)$. Therefore, the variances of all the outputs are $\mathrm{diag}(\mathbf{\Sigma}^*) = k(\mathbf{x}^*, \mathbf{x}^*)\mathrm{diag}(\mathbf{K}^1 \otimes \ldots \otimes \mathbf{K}^Q) - \mathrm{diag}(\mathbf{L}\mathbf{L}^\top) = k(\mathbf{x}^*, \mathbf{x}^*)\mathrm{diag}(\mathbf{K}^1 \otimes \ldots \otimes \mathbf{K}^Q) - \mathbf{L}^{\cdot 2} \cdot \mathbf{1}$. We then calculate these variances through tensor algebra, $\mathrm{diag}(\mathbf{\Sigma}^*) = \mathrm{vec}(\mathcal{M})$, where $\mathcal{M} = k(\mathbf{x}^*, \mathbf{x}^*) \cdot \big(\mathrm{diag}(\mathbf{K}^1) \circ \ldots \circ \mathrm{diag}(\mathbf{K}^Q)\big) + \mathcal{S}^{\cdot 2} \times_1 \mathbf{U}_1^{\cdot 2} \ldots \times_Q \mathbf{U}_Q^{\cdot 2} \times_{Q+1} (\mathbf{k}_*\mathbf{K}^{-1}\mathbf{U})^{\cdot 2}$ and $\mathcal{S} = (\boldsymbol{\lambda}_1 \circ \ldots \circ \boldsymbol{\lambda}_Q \circ \boldsymbol{\lambda}) \odot \mathcal{A}^{\cdot -\frac{1}{2}}$. Note that $\mathcal{S}$ corresponds to the diagonal matrix $\mathbf{\Lambda}(\mathbf{\Lambda} + \tau^{-1}\mathbf{I})^{-\frac{1}{2}}$ in $\mathbf{L}$. Since the computation for both $\boldsymbol{\mu}^*$ and $\mathrm{diag}(\mathbf{\Sigma}^*)$ involves the Tucker operator, the time complexity is $\mathcal{O}\big(Nd(\sum_{k=1}^Q d_k + N)\big)$.

4.3 Algorithm Complexity

The overall time complexity for inference and prediction is $\mathcal{O}\big(Nd(\sum_{k=1}^Q d_k + N)\big)$ which is dominated by the Tucker operator. The space complexity is $\mathcal{O}\big(Nd + N^2 + \sum_{k=1}^Q (d_k^2 + d_k r_k)\big)$, including the storage of the training data, the latent coordinate features in each tensor mode, and the kernel matrices of the latent features and the inputs. Obviously, the way to tensorize the outputs can significantly influence the computational complexity.

Lemma 4.1. *If we can arrange d outputs into a Q-mode tensor with an equal dimension in each mode, and $Qd^{\frac{1}{Q}} \leq N$, the time complexity of HOGPR is $\mathcal{O}(N^2 d)$ — linear to the number of outputs. Such Q always exists when $d \leq e^{\frac{N}{e}}$.*

The proof is given in the supplementary material. To identify an appropriate mode number Q, we can start with $\log_N^d$, and gradually increase it until the condition $Qd^{\frac{1}{Q}} \leq N$ is satisfied. The restriction for the number of outputs d is quite mild — when we have 100 samples, the linear scalability maintains until d grows to 9.5×10^{15} (9,500 trillion).

5 Related Works

Seminal works were proposed for multi-output GP regression. One type of methods (Higdon, 2002; Boyle and Frean, 2005; Teh et al., 2005; Bonilla et al., 2008) can be considered as convolved GPs that convolve smoothing kernels with a set of latent functions to produce the outputs. Despite the elegance, they have to use friendly (but perhaps less expressive) kernels, *e.g.,* Gaussian and delta, to ensure analytical convolutions. This might restrict their flexibility to exploit more expressive kernels. Moreover, their time complexity ($\mathcal{O}\big((Nd)^3\big)$) is prohibitively expensive and not scalable to high-dimensional outputs. To alleviate this issue,

several sparse approximations were proposed (Alvarez and Lawrence, 2009; Álvarez et al., 2010). Another type of methods (Higdon et al., 2008; Xing et al., 2015, 2016), use a linear combination of fixed bases to model the outputs, and hence can efficiently handle a large number of outputs. However, the linear structure might be oversimplified and inflexible to capture more complex output correlations. To address these issues, we propose HOGPR that does not impose any linear structure, and can exploit any kernel functions to model complex correlations. The Kronecker product in HOGPR further allows us to develop an exact inference algorithm scalable to very high-dimensional outputs. Recently, the Kronecker product has also been used for efficient approximate inference for single-output GPs (Wilson and Nickisch, 2015; Izmailov et al., 2018), specifically with factorized kernels (*e.g.,* RBF). These excellent works place inducing points and/or their variational posterior into grids of input dimensions to generate Kronecker products, and combine with interpolations to speed up computation. Although for a completely different problem, our model formulation resembles the infinite Tucker (InfTucker) decomposition (Xu et al., 2012) in multi-aspect data analysis. The key difference is that our model is a hybrid of GP and latent GP model while InfTucker is purely a latent GP without any observed inputs; our model predicts the test outputs as an empty full tensor, while InfTucker the missing entry values in the training tensor. Although both methods use intense tensor algebras for efficient inference, our method directly computes the marginal likelihood and gradient for optimization, while InfTucker uses alternative EM steps which seems unnecessary and could converge more slowly.

The idea of tensorization has been applied in learning compact neural networks (Novikov et al., 2015; Yang et al., 2017; Ye et al., 2018). In these works, the network weights are organized as tensors, and then replaced in training by a low-rank approximation, *e.g.,* the tensor-train (Oseledets, 2011) decomposition, to reduce the number of parameters. HOGPR, however, tensorizes the high-dimensional (observed) outputs rather than the model parameters.

6 Experiments

6.1 Small-Scale Multi-Output Regression

We first examined HOGPR on small datasets from three real-world applications: *Cantilever*, *PM2.5*[1] and *GeneExp*[2]. *Cantilever* are material structures that have the maximum stiffness when bearing forces from the right side. In each instance, the input defines a force, and the outputs are a 100 dimensional vector that describes the structure in the 10×10 squared domain. *PM2.5* measures the particulate matter pollution (PM) in Salt Lake City during July 4-7, 2018, including 100 spatial measurements (*i.e.,* 100 outputs). *GeneExp* comprise the expressions of $1,643$ genes (*i.e.,* outputs) measured by different microarrays. Each microarray is described by a 10 dimensional vector.

Competing Methods. We compared HOGPR with the following multi-output GP methods. (1) SCGPR — sparse convolved GP regression (Alvarez and Lawrence, 2009). The standard convolved GPs are infeasible for our datasets (even the smallest one). Hence, we used sparse approximations. (2) PCA-GPR (Higdon et al., 2008) — multi-output GP regression methods using a linear combination of fixed bases to produce the outputs. Here the bases are obtained from Principal Component Analysis (PCA) on the training outputs. We also compared with two cogenetic methods, (3) IsoMap-GPR (Xing et al., 2015) and (4) KPCA-GPR (Xing et al., 2016) that obtain the bases by IsoMap (Balasubramanian and Schwartz, 2002) and Kernel PCA (Schölkopf et al., 1998), respectively.

Parameter Settings. For HOGPR, we organized the outputs for *Cantilever*, *PM2.5* and *GeneExp*, into 2-mode tensors of sizes 10×10, 10×10 and 31×53, respectively. We employed the same number of latent features in each mode, *i.e.,* $r_1 = \ldots = r_Q = r$, and varied r from $\{1, 2, 5, 10\}$. The initialization of the latent features were element-wisely drawn from the uniform distribution in $[0, 1]$. For SCGPR, we set the number of inducing points to one-fourth of the training sample size, and varied the number of latent functions from $\{1, 2, 5, 10\}$. We used Gaussian smoothing kernels. For PCA-GPR, IsoMAP-GPR and KPCA-GPR, we varied the number of bases from the same range as well. All the methods were implemented with MATLAB. For SCGPR, we used the implementation by the authors and their group — the MULTIGP[3] software package. From each dataset, we randomly selected $\{128, 256\}$ samples for training, and from the remaining data 100 samples for test. All the datasets were normalized. We repeated this procedure for 5 times, and report the average Root-Mean-Squared-Error (RMSE). We used the ARD squared exponential kernel for all the methods, and the same initialization for the kernel parameters and the inverse variance.

The results are reported in Fig. 1. As we can see, HOGPR outperforms the competing methods in most of the cases, and often improves upon them by a large margin. It is worth noting that SCGPR is not very stable when the number of latent functions is set to 1 (see Fig. 1c,e,f). For example, in Fig. 1e, the average RMSE of SCGPR (*i.e.,* 8.4) turned out to be far larger than all the other methods and hence was not shown. In contrast, our method performed well even when only taking one latent feature (in each mode).

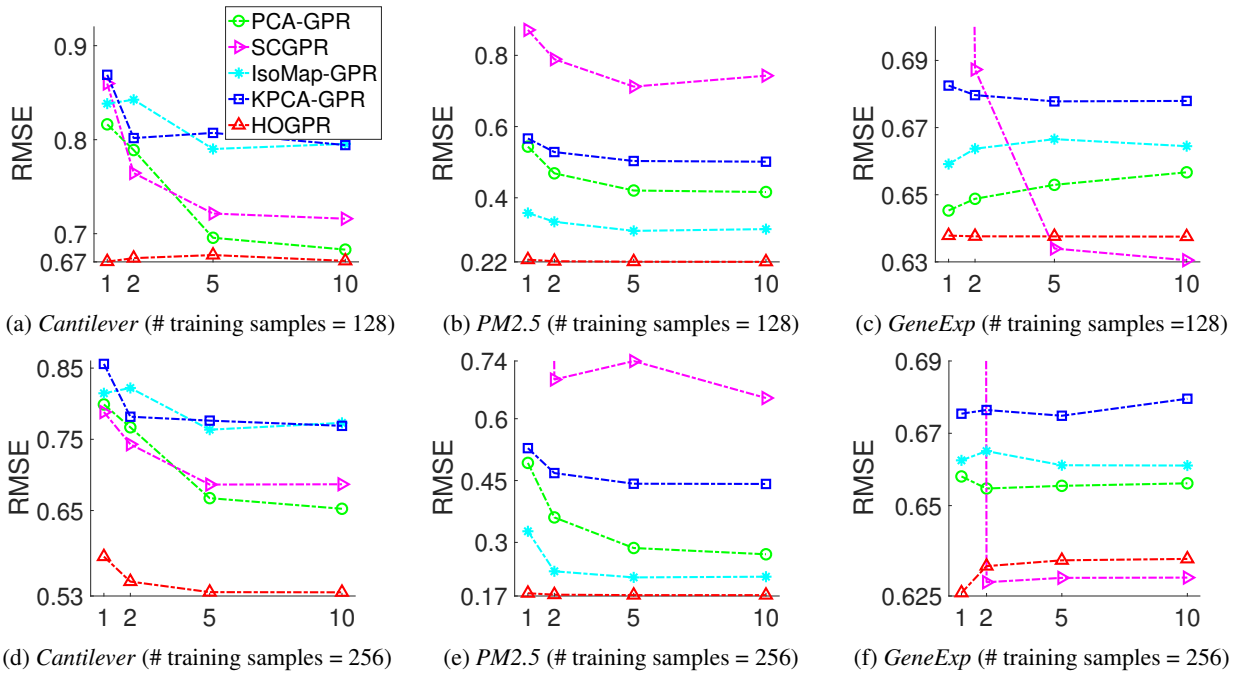
6.2 Topology Structure Design

We next applied HOGPR to predict optimal topology structures to withstand specific external forces. A topology structure is a layout of given materials (*e.g.,* concretes and alloys) in the designated spatial domain; the optimal structure is

[1] http://www.aqandu.org/

[2] https://www.synapse.org/#!Synapse:syn2787209/wiki/70349

[3] https://github.com/SheffieldML/multigp



(a) *Cantilever* (# training samples = 128) (b) *PM2.5* (# training samples = 128) (c) *GeneExp* (# training samples =128)

(d) *Cantilever* (# training samples = 256) (e) *PM2.5* (# training samples = 256) (f) *GeneExp* (# training samples = 256)

Figure 1: The root-mean-squared-error (RMSE) of all the multi-output regression methods on three small datasets. The results were averaged from 5 runs. The x-axis represents the number of the latent features for HOGPR, latent functions for SCGPR and bases for PCA-GPR, IsoMap-GPR and KPCA-GPR. Note that some results of SCGPR are not shown because they are much larger than all the other methods.

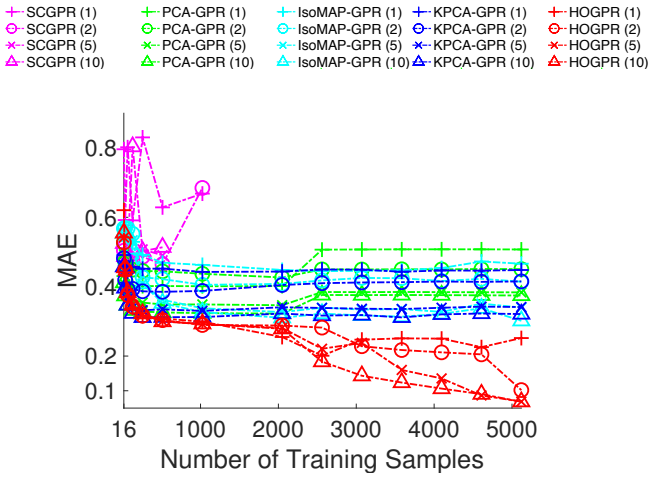
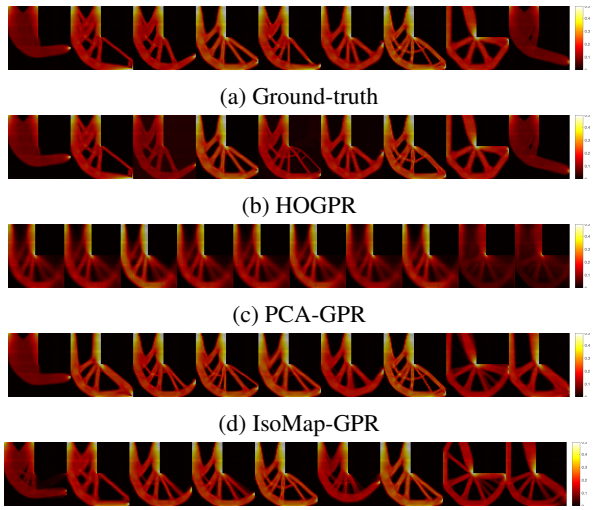


Figure 2: The mean-absolute-error (MAE) on the topology design datasets ($3,600$ outputs) w.r.t various numbers of training samples. For each particular training set size, the results were averaged over 5 runs. Numbers in the legend indicate the number of latent features, latent functions and bases for HOGPR, SCGPR, and {PCA-GPR, IsoMap-GPR, KPCA-GPR}, respectively. SCGPR failed and has no results when the number of training samples is larger than 1024, 512 or 256 and the latent functions $\{1, 2\}$, 5 or 10, respectively.



(a) Ground-truth

(b) HOGPR

(c) PCA-GPR

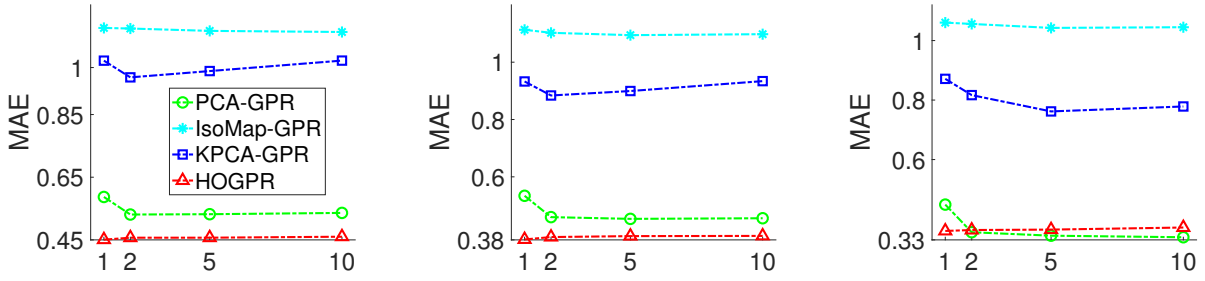
(d) IsoMap-GPR

(e) KPCA-GPR

Figure 3: The predicted topology structures for 9 different external forces. The number of the latent features for HOGPR, and the bases for PCA-GPR, IsoMAP-GPR and KPCA-GPR were set to 10. All the models were trained on $5,120$ samples.

required to achieve the maximum load or stiffness. Finding optimal topology structures is critical to many part design and manufacturing problems, such as 3D printing, airfoil and slab bridge design. Traditionally, it is modeled as a constrained optimization problem that minimizes a compliance subject to a total volume constraint (Sigmund, 1997). However, the numerical computation is usually exceedingly expensive. Here, we aim to learn a multi-output regression model to directly predict the optimal structures (given the force settings) and to prevent the costly optimization.

The dataset was generated by following the stress experiment in (Keshavarzzadeh et al., 2018), within an L-shape domain. Each structure is represented by a $3,600$ dimensional vector, and the force by a 3 dimensional vector. We collected $6,000$ structures. For HOGPR, we organized the outputs (*i.e.,* each structure) into a 2-mode 60×60 tensor.



(a) 64 training samples (b) 128 training samples (c) 256 training samples

Figure 4: The mean-absolute-error (MAE) in predicting pressure fields with *one million* outputs. The results were averaged from 5 runs.

We used 128 structures for test, and ran all the methods on training sets of a wide range of sizes, from 16 to $5,120$. For each size, we performed 5 runs on the randomly selected the training/test structures, and calculated the average Mean Absolute Error (MAE).

The predictive performance of all the methods is shown in Fig. 2. We found that although using sparse approximations, SCGPR failed to run due to numerical errors/excessive memory consumptions when the training set size grows to a certain degree (*e.g.,* $1,024$ with one latent function). Similar to the results in Section 6.1, the performance of SCGPR is not stable, especially when the number of latent functions is small (1 or 2). In general, the prediction accuracies of all the methods are close at the beginning, *i.e.,* the training size is small. When the training size grows, say, over $1K$, HOGPR exhibits evident improvement upon all the competing methods and the improvement grows as well. This might because, at the beginning, the training set is too small to provide effective information such that all the methods end up with poor predictive performance. When training samples are sufficient, HOGPR is able to capture the complex output correlations (better and better) and exhibits superior prediction accuracy.

To enable a fine-grained comparison, we visualize 9 structures predicted by all the methods except SCGPR (when the number of latent features/bases is set to 10, and training samples $5,120$) and the ground-truth. As shown in Fig. 3, the structures predicted by HOGPR are closest to the optimal structures (*i.e.,* the ground-truth). PCA-GPR often predicted inaccurate material intensity (see the first and last structure in Fig. 3c) and blurred local structures. While IsoMap-GPR and KPCA-GPR yielded clearer local structures, their local details often deviate significantly from the ground-truth (see the second to sixth and ninth structures in Fig. 3d, third to seventh structures in Fig. 3e).

6.3 Large Physical Simulations Based on the Incompressible Navier-Stokes Equations

Finally, we examined HOGPR in a large-scale problem to predict one million outputs for each input. Specifically, the tasks is to predict the pressure field of the lid-driven cavity flow (Bozeman and Dalton, 1973). The liquid inside a cavity is driven by the walls, causing different pressures locally and eventually leading to turbulent flow inside the cavity. The simulation of the field involves solving the incompressible Navier-Stokes equations (Chorin, 1968) which are well known to be challenging to solve due to their complicated behaviors under large Reynolds numbers. Hence, in most cases, fine-grained meshes are required to ensure convergence of the numerical solver. This implies that the output field is very high dimensional. For each boundary condition (represented by a 5 dimensional input vector), we simulated a field of one million dimensions. Due to the computational cost, we only collected 400 instances. Therefore, this is a typical "large d, small N" problem. We organized each output field into a $100 \times 100 \times 100$ tensor. We randomly selected 128 fields for test, and $\{64, 128, 256\}$ instances for training. We only compared with PCA-GPR, IsoMAP-GPR and KPCA-GPR, because SCGPR is infeasible. We randomly selected the training and test fields, computed the test MAE for each method, and repeated 5 times to report the average MAE. The results are shown in Fig. 4. As we can see, in most of the cases, HOGPR outperforms the competing methods by a large margin, demonstrating superior predictive performance. The running time of HOGPR on 256 samples are $\{8.9, 9.2, 9.3, 10.0\}$ minutes per-iteration for $r = 1, 2, 5, 10$, while PCA-GPR, IsoMAP-GRP and KPCA-GPR are much faster, taking less than one minute. This is as expected, because the competing methods use a linear combination of fixed bases to predict the outputs, and only need to estimate the coefficient of the bases. This will drastically reduce the computation overhead. However, HOGPR possessed more flexibility to capture complex correlations among outputs (via all kinds of kernels) so as to improve the predictive power. In the mean time, HOGPR can achieve a linear scalability to large output dimensions.

7 Conclusion

We have proposed HOGPR, a GP model for high-dimensional output regression. HOGPR not only is flexible to incorporate arbitrary kernels to model/capture complex output correlations, but also enables exact, scalable inference for a large number of outputs.

Acknowledgement

This work was supported by the DARPA TRADES Award HR0011-17-2-0016.

References

Alvarez, M. and Lawrence, N. D. (2009). Sparse convolved gaussian processes for multi-output regression. In Advances in neural information processing systems, pages 57–64.

Álvarez, M., Luengo, D., Titsias, M., and Lawrence, N. (2010). Efficient multioutput gaussian processes through variational inducing kernels. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pages 25–32.

Balasubramanian, M. and Schwartz, E. L. (2002). The isomap algorithm and topological stability. Science, 295(5552):7–7.

Bonilla, E. V., Chai, K. M., and Williams, C. (2008). Multi-task gaussian process prediction. In Advances in neural information processing systems, pages 153–160.

Boyle, P. and Frean, M. (2005). Dependent gaussian processes. In Advances in neural information processing systems, pages 217–224.

Bozeman, J. D. and Dalton, C. (1973). Numerical study of viscous flow in a cavity. Journal of Computational Physics, 12(3):348–363.

Chorin, A. J. (1968). Numerical solution of the navier-stokes equations. Mathematics of computation, 22(104):745–762.

Higdon, D. (2002). Space and space-time modeling using process convolutions. In Quantitative methods for current environmental issues, pages 37–56. Springer.

Higdon, D., Gattiker, J., Williams, B., and Rightley, M. (2008). Computer model calibration using high-dimensional output. Journal of the American Statistical Association, 103(482):570–583.

Izmailov, P., Novikov, A., and Kropotov, D. (2018). Scalable gaussian processes with billions of inducing inputs via tensor train decomposition. In International Conference on Artificial Intelligence and Statistics, pages 726–735.

Keshavarzzadeh, V., Kirby, R. M., and Narayan, A. (2018). Parametric topology optimization with multi-resolution finite element models. arXiv preprint arXiv:1808.10367.

Kim, H., Lu, X., Flaxman, S., and Teh, Y. W. (2016). Collaborative filtering with side information: a gaussian process perspective. arXiv preprint arXiv:1605.07025.

Kolda, T. G. (2006). Multilinear operators for higher-order decompositions, volume 2. United States. Department of Energy.

Lawrence, N. D. and Urtasun, R. (2009). Non-linear matrix factorization with gaussian processes. In Proceedings of the 26th Annual International Conference on Machine Learning, pages 601–608. ACM.

Lu, C. and Tang, X. (2015). Surpassing human-level face verification performance on lfw with gaussianface. In AAAI, pages 3811–3819.

Mitchell, A. R. and Griffiths, D. F. (1980). The finite difference method in partial differential equations. Number BOOK. John Wiley.

Mockus, J. (2012). Bayesian approach to global optimization: theory and applications, volume 37. Springer Science & Business Media.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. (2015). Tensorizing neural networks. In Advances in Neural Information Processing Systems, pages 442–450.

Oseledets, I. V. (2011). Tensor-train decomposition. SIAM Journal on Scientific Computing, 33(5):2295–2317.

Schölkopf, B., Smola, A., and Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. Neural computation, 10(5):1299–1319.

Sigmund, O. (1997). On the design of compliant mechanisms using topology optimization. Journal of Structural Mechanics, 25(4):493–524.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Advances in neural information processing systems, pages 2951–2959.

Teh, Y.-W., Seeger, M., and Jordan, M. (2005). Semiparametric latent factor models. In Artificial Intelligence and Statistics 10, number EPFL-CONF-161317.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured gaussian processes (kiss-gp). In International Conference on Machine Learning, pages 1775–1784.

Xing, W., Shah, A. A., and Nair, P. B. (2015). Reduced dimensional gaussian process emulators of parametrized partial differential equations based on isomap. In Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, volume 471, page 20140697. The Royal Society.

Xing, W., Triantafyllidis, V., Shah, A., Nair, P., and Zabaras, N. (2016). Manifold learning for the emulation of spatial fields from computational models. Journal of Computational Physics, 326:666–690.

Xu, Z., Yan, F., and Qi, Y. (2012). Infinite Tucker decomposition: Nonparametric Bayesian models for multiway data analysis. In Proceedings of the 29th International Conference on Machine Learning (ICML).

Yang, Y., Krompass, D., and Tresp, V. (2017). Tensor-train recurrent neural networks for video classification. In International Conference on Machine Learning, pages 3891–3900.

Ye, J., Wang, L., Li, G., Chen, D., Zhe, S., Chu, X., and Xu, Z. (2018). Learning compact recurrent neural networks

with block-term tensor decomposition. In Proceedings of 2018 IEEE international conference on computer vision and pattern recognition. Google Scholar.

Zienkiewicz, O. C., Taylor, R. L., Zienkiewicz, O. C., and Taylor, R. L. (1977). The finite element method, volume 36. McGraw-hill London.