Multi-output Gaussian Process Regression via Multi-task Neural Network

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

A neural network can be viewed as a finite feature map, from which a reduced-rank Gaussian process model can be built. On the other hand, multiple correlated outputs can be represented by a neural network with shared hidden layers. In this paper, we propose a simple multi-output Gaussian process regression model based on the aforementioned two ideas. The kernel functions of multiple outputs are constructed from a multi-task neural network with shared hidden layers and task-specific layers. The correlations of the outputs are thus captured by the shared hidden layers. We compare our multi-task neural network enhanced Gaussian process (MTNN-GP) model with several existing multi-output Gaussian process models using two public datasets and one example of real-world analog integrated circuits. The results show that our model is competitive compared with these models.

1 Introduction

The Gaussian process (GP) regression is popular as the model provides both predictions and well-calibrated uncertainties. The uncertainty estimation makes the model more robust to unseen events as the model *knows what it knows*. The conventional GP models are usually designed to learn a scalar-valued function. However, in some scenarios, we need to model a vector-valued function, and the multiple outputs are possibly correlated. Instead of treating the vector-valued function as multiple separate scalar-valued functions, the multi-output learning [1] tries to build a unified model and simultaneously learn all the outputs. The overall performance could be enhanced by exploiting the correlations of the tasks.

Multi-task Gaussian process [2] tries to combine multi-task learning and Gaussian process. The linear model of coregionalization (LMC) methods [3] assume the Q outputs $f_i(\boldsymbol{x}), i \in \{1 \dots Q\}$ are linear combinations of several latent functions as $f_i(\boldsymbol{x}) = \sum_{j=1}^{U} a_{ij} u_j(\boldsymbol{x})$. In [4], the covariance matrix is expressed as the Kronecker product of the task covariance and the task-irrelevant input covariance matrix. In [5], the output function $f_i(\boldsymbol{x})$ is expressed as the linear combinations of U latent functions $u_j(\boldsymbol{x}), j \in \{1 \dots U\}$ plus a task-specific function $h_i(\boldsymbol{x})$. Efficient training and inference methods are also developed. In the Gaussian process regression network (GPRN) model [6], the i-th output function $f_i(\boldsymbol{x})$ is also expressed as weighted combinations of U latent functions, however, the weights are also nonlinear functions characterized by GP so that $f_i(\boldsymbol{x}) = \sum_{j=1}^{U} w_{ij}(\boldsymbol{x})(u_j(\boldsymbol{x}) + \epsilon_u) + \epsilon_f$ where ϵ_u and ϵ_f are the noise terms. Another methodology of building multi-task kernels is *process convolution* [7, 8, 9], where the output function $f_i(\boldsymbol{x})$ is assumed to be the convolution of output-dependent smoothing kernel $g_i(\boldsymbol{x})$ and a latent function $u(\boldsymbol{x})$.

In this paper, we propose a multi-output Gaussian process model based on the multi-task neural network. As a degenerate GP can be built from a finite feature map and neural networks can generate representative features, a degenerate GP can be derived from a neural network with finite hidden units

[10, 11]. In our proposed model, the multi-output GP is built from a neural network with shared layers and task-specific layers. The input data is first transformed into *shared features* through the shared layers and further mapped to *task-specific features* by each task's task-specific layers. GP kernels for the outputs are then built from the inner products of the task-specific features. The weights of the multi-task neural network is obtained by maximizing the likelihood with gradient back-propagation.

We compared our model with independent GP model and several multi-output GP models. Three datasets were used, including two public datasets and one dataset from the simulation results of a real-world analog integrated circuit. It is demonstrated that our model can provide better predictions and uncertainty estimations than the compared models.

The rest of the paper is organized as follows. In Section 2, we present the background of the Gaussian process and GP model built from neural networks with finite hidden units. In Section 3, we present our proposed multi-output GP model via multi-task neural network. The experimental results are given in Section 4. We conclude the paper in Section 5.

2 Background

2.1 Gaussian Process Regression

Given a training set $D_T = \{X, \boldsymbol{y}\}$ where $X = \{\boldsymbol{x}_1, \ \dots \ \boldsymbol{x}_N\}$, and $\boldsymbol{y} = \{y_1, \ \dots \ y_N\}$, we assume the target value y is generated by a latent function $f(\boldsymbol{x})$ with additive noise $\epsilon \sim N(0, \sigma_n^2)$ such that

$$y_i \sim N(f(\boldsymbol{x}_i), \sigma_n^2) \tag{1}$$

Here $N(\cdot, \cdot)$ denotes a Gaussian distribution. We use Gaussian process (GP) [12] to learn the latent function $f(\boldsymbol{x})$. A GP model defines a prior over $f(\boldsymbol{x})$. GP is fully characterized by a mean function $m(\boldsymbol{x})$ and a covariance function $k(\boldsymbol{x}, \boldsymbol{y})$. For the training set D_T, the latent function values $\boldsymbol{f} = (f(\boldsymbol{x}_1), \ \dots \ f(\boldsymbol{x}_N))^T$ follow a joint Gaussian distribution $\boldsymbol{f} \sim N(\boldsymbol{m}, K)$, where $\boldsymbol{m} = (m(\boldsymbol{x}_1), \ \dots \ , m(\boldsymbol{x}_N))^T$ is the mean vector, and $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the covariance matrix. The mean function $m(\boldsymbol{x})$ can be any function, while the kernel function $k(\boldsymbol{x}, \boldsymbol{y})$ has to make sure that the covariance matrix is a symmetric positive definite (SPD) matrix. In this paper, we fix $m(\boldsymbol{x}) = 0$.

Given a new input $\boldsymbol{x}_*$, GP model predicts the distribution of the output, i.e., $y \sim N(\mu(\boldsymbol{x}_*), \sigma^2(\boldsymbol{x}_*))$, where $\mu(\boldsymbol{x}_*)$ and $\sigma^2(\boldsymbol{x}_*)$ can be expressed as

$$\left\{ \begin{array}{lcl} \mu(\boldsymbol{x}_*) & = & k(\boldsymbol{x}_*, X)(K + \sigma_n^2 I)^{-1}\boldsymbol{y} \\ \sigma^2(\boldsymbol{x}_*) & = & \sigma_n^2 + k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, X)(K + \sigma_n^2 I)^{-1} k(X, \boldsymbol{x}_*), \end{array} \right. \tag{2}$$

where $k(\boldsymbol{x}_*, X) = (k(\boldsymbol{x}_*, \boldsymbol{x}_1), \ \dots \ , k(\boldsymbol{x}_*, \boldsymbol{x}_N))$ and $k(X, \boldsymbol{x}_*) = k(\boldsymbol{x}_*, X)^T$. In (2), $\mu(\boldsymbol{x}_*)$ and $\sigma^2(\boldsymbol{x}_*)$ can be viewed as the prediction and the uncertainty measure.

There are usually some hyperparameters for a GP model, including the noise level σ_n and the hyperparameters for the kernel functions. For example, the squared exponential kernel is a commonly used kernel in GP regression. The kernel function is defined as

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_f^2 \exp\Big(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \Lambda^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)\Big), \tag{3}$$

where $\Lambda = \mathrm{diag}(l_1, \dots, l_d)$ is a diagonal matrix and l_i denotes the length scale of the i-th dimension. σ_f and Λ are the hyperparameters for the kernel. Denote $\boldsymbol{\theta}$ as the vector of hyperparameters, the hyperparameters can be learned via maximum likelihood estimation (MLE) by maximizing the following likelihood function

$$\log p(\boldsymbol{y}|X, \boldsymbol{\theta}) = -\frac{1}{2}(\boldsymbol{y}^T K_{\boldsymbol{\theta}}^{-1} \boldsymbol{y} + \log|K_\theta| + N \log(2\pi)), \tag{4}$$

where $K_{\boldsymbol{\theta}}$ is the covariance matrix of the training input calculated by the kernel function.

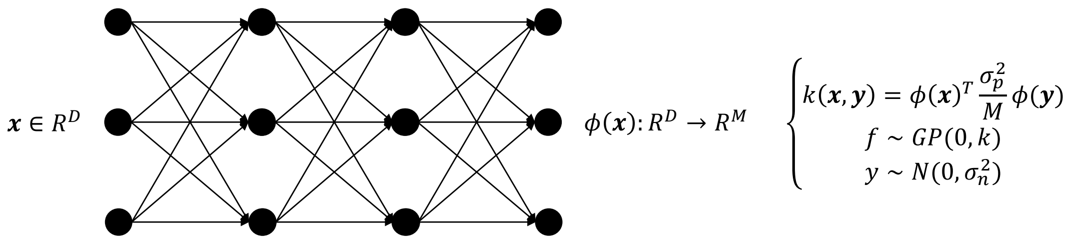

$\boldsymbol{x} \in R^D$
$\phi(\boldsymbol{x}): R^D \to R^M$
$k(\boldsymbol{x},\boldsymbol{y}) = \phi(\boldsymbol{x})^T \frac{\sigma_p^2}{M} \phi(\boldsymbol{y})$
$f \sim GP(0,k)$
$y \sim N(0,\sigma_n^2)$

Figure 1: Architecture of the Gaussian process model with kernel characterized by neural network.

2.2 Reduced-Rank Gaussian Process Model from Finite-Dimensional Feature Map

A Gaussian process model can also be derived from a weight space view. Let $\phi(\boldsymbol{x}) : R^D \to R^M$ be a feature map from D-dimensional input space to the M-dimensional feature space. We assume that the latent function $f(\boldsymbol{x})$ is a linear combination of the nonlinear features, and the observed target values are generated from $f(\boldsymbol{x})$ with additive noise

$$\left\{ \begin{array}{lcl} f(\boldsymbol{x}) & = & \boldsymbol{w}^T\phi(\boldsymbol{x}) \\ y_i & \sim & N(f(\boldsymbol{x}_i), \sigma_n^2) \end{array} \right. . \tag{5}$$

If a zeros mean Gaussian prior with covariance matrix Σ_p is imposed on the weights $\boldsymbol{w}$, i.e., $\boldsymbol{w} \sim N(0, \Sigma_p)$, it can be proved that f follows a Gaussian process $f \sim \mathcal{GP}(0, k)$ [12], with the kernel function k defined as

$$k(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^T \Sigma_p \phi(\boldsymbol{y}). \tag{6}$$

The GP model defined from finite feature map is called *degenerate* Gaussian process, as the covariance matrix calculated from (6) would have a lower rank M than N.

If we set Σ_p to a diagonal matrix $\Sigma_p = \frac{\sigma_p^2}{M} I$, the predictive distribution of (2) can be reformulated as [12, 10]

$$\left\{ \begin{array}{lcl} \mu(\boldsymbol{x}) & = & \phi(x)^T A^{-1} \Phi \boldsymbol{y} \\ \sigma^2(\boldsymbol{x}) & = & \sigma_n^2 + \sigma_n^2 \phi(\boldsymbol{x})^T A^{-1} \phi(\boldsymbol{x}) \\ \Phi & = & (\phi(\boldsymbol{x}_1), \dots, \phi(\boldsymbol{x}_N)) \\ A & = & \Phi\Phi^T + \frac{M\sigma_n^2}{\sigma_p^2} I \end{array} \right. . \tag{7}$$

Note that the when calculating $\mu(\boldsymbol{x})$ and $\sigma^2(\boldsymbol{x})$ directly using (2), the time complexity are $O(N)$ and $O(N^2)$, respectively. However, if (7) is used, the time complexity for calculating $\mu(\boldsymbol{x})$ and $\sigma^2(\boldsymbol{x})$ become $O(M)$ and $O(M^2)$ as long as the inverse of A is pre-calculated.

The log likelihood of the training data defined in (4) can also be reformulated as [10]

$$\log p(\boldsymbol{y}|X, \boldsymbol{\theta}) = -\frac{1}{2\sigma_n^2}(\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T \Phi^T A^{-1} \Phi \boldsymbol{y}) - \frac{1}{2}\log|A| + \frac{M}{2}\log\frac{M\sigma_n^2}{\sigma_p^2} - \frac{N}{2}\log(2\pi\sigma_n^2), \tag{8}$$

where $\boldsymbol{\theta}$ is the vector containing σ_p, σ_n and the parameters of ϕ. In (4), the covariance matrix K should be inverted, which would take $O(N^3)$ operations. For (8), the matrix A is of the size $M \times M$, so the time complexity for calculating (8) is only $O(NM^2 + M^3)$.

It can be seen that GP model can be constructed from a finite feature map. Neural network (NN) can provide effective feature representations, it is thus natural to use a neural network as the feature map ϕ. In [10], a neural network with one hidden layer is proposed as the feature map $\phi(\boldsymbol{x})$. The weights of the neural network are obtained by maximizing the likelihood function in (8) with gradient back-propagation. In [11], the single hidden layer is extended to multiple layers. The Gaussian process with kernel characterized by neural network is illustrated in Figure 1. A similar work is [13], where a neural network is firstly pre-trained, and Bayesian linear regression is then performed to the last layer. The finally trained model is used for Bayesian optimization.

2.3 Model Averaging to Improve the Quality of Uncertainty Prediction

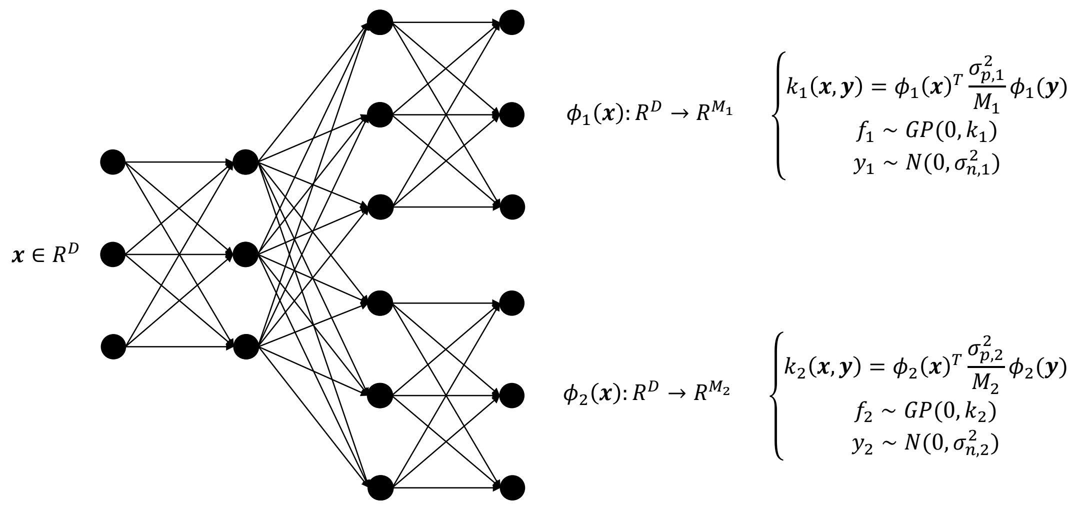
When probabilistic models are built from neural networks, a simple model averaging technique [10, 11, 14] can significantly improve the quality of the estimated uncertainty.

Firstly, K independent probabilistic neural network models are trained with random initializations. Each model would give predictive distribution $p(y|\boldsymbol{x}, \boldsymbol{\theta}_k) = N(\mu_k(\boldsymbol{x}), \sigma_k^2(\boldsymbol{x}))$ where $\boldsymbol{\theta}_k$ is the neural network parameters for the k-th model, $\mu_k(\boldsymbol{x})$ and $\sigma_k^2(\boldsymbol{x})$ are the mean and variance of the corresponding predictive Gaussian distribution. The final predictive distribution can be expressed as $p(y|\boldsymbol{x}) = N(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x}))$, where

$$\left\{\begin{array}{lll} \mu(\boldsymbol{x}) & = & \frac{1}{K}\sum_k \mu_k(\boldsymbol{x}) \\ \sigma^2(\boldsymbol{x}) & = & \frac{1}{K}\sum_k (\mu_k^2(\boldsymbol{x}) + \sigma_k^2(\boldsymbol{x})) - \mu^2(\boldsymbol{x}) \end{array}\right. . \tag{9}$$

In [10, 11], the uncertainty is obtained according to (7), while in [14], the uncertainty is generated by adversarial training. It is shown that the ensemble technique defined in (9) can greatly improve the quality of the uncertainty measure. In [14], with $K = 5$, the model-averaging significantly improves the quality of the uncertainty estimation and outperforms Bayesian-based models like probabilistic backpropagation [15] and MC-dropout [16].

3 Multi-output Gaussian Process Regression via Neural network



$\boldsymbol{x} \in R^D$
$\phi_1(\boldsymbol{x}): R^D \rightarrow R^{M_1}$
$k_1(\boldsymbol{x},\boldsymbol{y}) = \phi_1(\boldsymbol{x})^T \frac{\sigma_{p,1}^2}{M_1} \phi_1(\boldsymbol{y})$
$f_1 \sim GP(0, k_1)$
$y_1 \sim N(0, \sigma_{n,1}^2)$
$\phi_2(\boldsymbol{x}): R^D \rightarrow R^{M_2}$
$k_2(\boldsymbol{x},\boldsymbol{y}) = \phi_2(\boldsymbol{x})^T \frac{\sigma_{p,2}^2}{M_2} \phi_2(\boldsymbol{y})$
$f_2 \sim GP(0, k_2)$
$y_2 \sim N(0, \sigma_{n,2}^2)$

Figure 2: Architecture of the multi-output Gaussian process model.

We have shown that a GP can be constructed from a neural network with finite hidden units in the last layer. Now, we will show how to model the correlations between tasks and how to construct a multi-output Gaussian process model based on neural network.

Suppose we have N observations $D_Q = \{X, Y | X \in R^{D\times N}, Y \in R^{N\times Q}\}$. We assume they are generated by Q latent functions and polluted by noises with different noise levels. Instead of building Q independent models, we build a multi-output model that makes use of the correlations between the Q tasks. Firstly, a neural network is used to define a *shared feature map* $\phi_s : R^D \rightarrow R^{M_s}$. Then, for each task $i \in \{1, \ \dots \ , Q\}$, the shared feature $\phi_s(\boldsymbol{x})$ is followed by a *task-specific* neural network that defines a feature map $\phi_i : R^{M_s} \rightarrow R^{M_i}$. We assume the i-th latent function $f_i(\boldsymbol{x})$ follows a GP distribution $f_i \sim \mathcal{GP}(0, k_i)$ defined as follows:

$$\left\{\begin{array}{lll} k_i(\boldsymbol{x},\boldsymbol{y}) & = & \phi_i(\phi_s(\boldsymbol{x}))^T \ \frac{\sigma_{p,i}^2}{M_i} \ \phi_i(\phi_s(\boldsymbol{y})) \\ f_i & \sim & \mathcal{GP}(0, k_i) \\ y_i & \sim & N(f_i(\boldsymbol{x}), \sigma_{n,i}^2) \end{array}\right. . \tag{10}$$

As illustrated in Figure 2, by defining GP with kernel function of (10), we actually defined a neural network architecture with shared layers and task-specific layers, which is a common architecture used in multi-task deep learning [17]. The correlations between tasks are naturally encoded by the shared layers, while the task-specific features are further learnt from the shared features.

With the model define in (10), different tasks are *conditionally independent* given the shared features. Each specific task still sees a neural network with the same architecture plotted in Figure 1. The inferences of $\mu(\boldsymbol{x})$ and $\sigma^2(\boldsymbol{x})$ are exactly the same as (7), and no additional overhead will be introduced. As different tasks are conditionally independent, the log likelihood of the training data can be expressed as the sum of the log likelihood of each specific task

$$\log p(Y|X, \boldsymbol{\Theta}) = \sum_{i=1}^{Q} \log p(\boldsymbol{y}_i|X, \boldsymbol{\theta}_i, \boldsymbol{\theta}_s), \tag{11}$$

where $\log p(\boldsymbol{y}_i|X, \boldsymbol{\theta}_i, \boldsymbol{\theta}_s)$ is the log likelihood of the i-th task as defined in (8), $\boldsymbol{y}_i$ is the i-the column of Y, $\boldsymbol{\Theta}$ is the vector of all the parameters, including the shared parameters and the task-specific parameters. $\boldsymbol{\theta_i}$ is the vector of parameters for specific task i, including the weights of the i-th task-specific neural network, the weight prior factor $\sigma_{p,i}$ and the noise level $\sigma_{n,i}$ for the i-th task. $\boldsymbol{\theta}_s$ is a vector of the weights of the shared layers. The parameters $\boldsymbol{\Theta}$ are obtained by maximizing the likelihood function in (11) with gradient back-propagation. The model averaging technique as described in subsection 2.3 is also employed in our model to improve the quality of uncertainty estimation. K independent neural network models are trained with random initializations in our model. According to (8) and (11), the time complexity of training for our model is $O(KN\sum_i^Q M_i^2)$.

4 Experimental Results

We implemented the multi-output Gaussian process model using Python 3.5.2. The gradient of the loss function in (11) is calculated by `autograd` package [18]. We compared our model with independent GP and several state-of-the-art multi-output GP models on three datasets. The datasets include two publicly available datasets and one dataset sampled from a real-world analog integrated circuit, as summarized in Table 1. All the datasets and the test code are provided in the supplementary materials and will be made public upon publication.

Table 1: Summary of the used datasets

Dataset	# inputs	# outputs	# training	# testing
ENB	8	2	700	68
SARCOS	21	2	44484	4449
OpAmp	10	15	2000	8000

We use standardized mean squared error (SMSE) and negative log likelihood (NLL) as the evaluation criteria. For all the test cases and the compared models, 10 independent runs were performed to average the random fluctuations. We report both the means and standard deviations of the SMSE and NLL.

4.1 The Energy Building (ENB) Dataset

The ENB dataset used is a small dataset with 768 samples, each sample has 8 inputs and 2 outputs. We use 700 samples as the training data, and the remaining 68 samples are used for testing. The dataset comes from simulations to 768 buildings [19, 20]. The 8 inputs are the building parameters like surface area and orientation, while the 2 outputs are the heating load and the cooling load[1].

For our MTNN-GP model, we use a neural network with 2 shared layers, and 1 task-specific layer per output, K is set to 5. Each layer has 100 hidden units with the tanh activation function. For this architecture, we need to learn more than 30 thousands of parameters using only 700 samples.

The MTNN-GP model is compared with independent GP modeling (IGP) using the `GPML` package [21] and 4 multi-output Gaussian process models, including the collaborative multi-output Gaussian

[1]The dataset is available at http://mulan.sourceforge.net/datasets-mtr.html

processes (COGP)[2] proposed in [5], the sparse convolved Gaussian processes (SCGP)[3] method proposed in [8], and the Gaussian process regression network with nonparametric variational inference and mean-field inference methods[4](GPRN-NPV and GPRN-MF) [22]. For the independent GP (IGP), the ARD squared-exponential kernel function is used. For other methods, we use the default configurations of the corresponding open source packages, except that we used 200 inducing points for the COGP model instead of using the default 500 inducing points.

The SMSE and NLL statistics are given in Table 2. The GPRN-NPV and GPRN-MF models give no predictive variances, so only the SMSE statistics are reported. We can see that although more than 30 thousands of parameters are learnt from only 700 samples, the learnt model gives very good prediction to the test set. Our MTNN-GP is better than the independent GP models. As for other multi-output GP models, they all gave results worse than the IGP models.

Table 2: The SMSE and NLL statistics of the ENB dataset

Algo	Output1(SMSE)	Output2(SMSE)	Output1(NLL)	Output2(NLL)
MTNN-GP	**0.00155 $\pm$ 0.000159**	**0.00753 $\pm$ 0.00135**	**0.332 $\pm$ 0.0634**	**0.972 $\pm$ 0.107**
IGP	0.00188 $\pm$ 0	0.00911 $\pm$ 0	0.538 $\pm$ 0	1.01 $\pm$ 0
GOGP	0.00597 $\pm$ 0.00088	0.0144 $\pm$ 0.000831	1.34 $\pm$ 0.159	2.08 $\pm$ 0.212
SCGP	0.708 $\pm$ 3.78e-05	1.21 $\pm$ 4.1e-05	1.56 $\pm$ 0.00363	1.66 $\pm$ 0.00063
GPRN-NPV	6.87 $\pm$ 9.36e-16	8.63 $\pm$ 1.87e-15	NA	NA
GPRN-MF	0.359 $\pm$ 0.225	0.614 $\pm$ 0.339	NA	NA

4.2 The SARCOS dataset

Table 3: The SMSE and NLL statistics of the SARCOS dataset

Algo	Output1(SMSE)	Output2(SMSE)	Output1(NLL)	Output2(NLL)
MTNN-GP	**0.00156 $\pm$ 3.46e-05**	**0.00307 $\pm$ 5.64e-05**	**0.804 $\pm$ 0.0111**	**-0.509 $\pm$ 0.00813**
IGP	0.0045 $\pm$ 0.000153	0.00787 $\pm$ 0.000257	1.06 $\pm$ 0.00941	-0.236 $\pm$ 0.0124
GOGP	0.00852 $\pm$ 0.000241	0.0149 $\pm$ 0.000433	2.48 $\pm$ 0.0577	2.4 $\pm$ 0.097
SCGP	4.7 $\pm$ 0.0245	3.39 $\pm$ 0.0192	4.63 $\pm$ 0.0128	2.87 $\pm$ 0.011
GPRN-NPV	4.99 $\pm$ 0	3.58 $\pm$ 0	NA	NA
GPRN-MF	2.21 $\pm$ 2.06	1.65 $\pm$ 1.51	NA	NA
COGP [5]	0.2631	0.0127	3.6	0.8302
GPRN-AVI [23]	$\approx$ 0.009	> 0.009	NA	NA

We use the SARCOS dataset[5] to test the scalability of our model for large and high dimensional dataset. The dataset comes from a robot inverse dynamic model. The whole dataset contains 48933 samples. Each sample has 21 inputs (the joint positions, joint velocities, and joint accelerations) and 7 targets (7 joint torques). 44484 samples are used as the training set, and the remaining 4449 samples are used as the testing data. We select the 4-th and 7-th torques as the two outputs, which is the same setting as [5] and [23]. We also compared our experimental results with the results reported by [5, 23].

For our MTNN-GP model, we use the same setting as we used for the ENB dataset. For the independent GP model, we used the FITC approximation with 200 inducing points to speedup the training. As the SCGP, GPRN-NPV and GPRN-MF cannot scale to such large dataset, in each run, we randomly select 1000 points from the training set to train the two GPRN models and 2000 points to train the SCGP model.

The SMSE and NLL results are listed in Table 3. It can be clearly seen that our MTNN-GP gave better predictions than the independent GP models, and the IGP model outperformed all other multi-output Gaussian process models.

[2]downloaded from https://github.com/trungngv/cogp

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

[4]downloaded from https://github.com/trungngv/gprn

[5]available at http://www.gaussianprocess.org/gpml/data/

Note that the data of the last two rows in Table 3 are not from experiments we performed, but from the public reported results of two multi-output Gaussian process models [5, 23]. We show that simply using independent GP model with FITC approximation can generate considerably better performances than the reported results.

The reported results of the COGP model are different with our experimental results due to the different settings. In [5], only 2000 training points were used for the first output, while we used the whole 44484 training data for both outputs. Also, we used 200 inducing points (same as the independent GP), while in [5], 500 inducing points are used. The COGP results from our experiments gave better predictions for the first output but worse predictions for the second output.

4.3 Behaviour Modeling of Operational Amplifier

The last dataset we used is the simulation data of a real-world analog integrated circuit. The circuit is an operational amplifier (OpAmp) with 10 design variables. We randomly sampled the 10 design variables and used a commercial circuit simulator HSPICE to get the performances of the circuit. We considered the gain, unit gain frequency (UGF) and the phase margin (PM) of the OpAmp over 5 design corners, so there are up to 15 performances considered. We gathered 10000 samples and used 2000 points as the training set, the rest 8000 data points are used for testing. The dataset can be seen in the supplementary materials.

The IGP, COGP, SCGP, GPRN-NPV and GPRN-MF models are compared. The algorithm settings for our MTNN-GP and the compared models are same with the settings described in Section 4.2.

The SMSE statistics are listed in Table 4 and the NLL statistics are given in Table 5. Like the previous two datasets, the MTNN-GP model gives far better predictions than the IGP and other multi-output GP models. The SMSE of the COGP model is slightly better than the IGP model, but the NLL results are worse. The SCGP and the GPRN-NPV completely underfitted the dataset, they predict everything to zero. Although the SMSE values of COGP is better than IGP, SCGP give constant prediction for all inputs. We found in Table 5 that the NLL of COGP is much worse than SCGP.

Table 4: The SMSE statistics of the OpAmp dataset

Output	MTNN-GP	IGP	COGP	SCGP	GPRN-NPV	GPRN-MF
O1	**6.5e-4 $\pm$ 3.5e-5**	0.20 $\pm$ 0.0019	0.19 $\pm$ 0.0044	1 $\pm$ 0	1 $\pm$ 0	0.79 $\pm$ 0.039
O2	**6.2e-4 $\pm$ 3.9e-5**	0.18 $\pm$ 0.0088	0.15 $\pm$ 0.0033	1 $\pm$ 0	1 $\pm$ 0	0.73 $\pm$ 0.049
O3	**5.4e-4 $\pm$ 5.4e-5**	0.08 $\pm$ 0.003	0.08 $\pm$ 0.002	1 $\pm$ 0	1 $\pm$ 0	0.70 $\pm$ 0.045
O4	**2.4e-4 $\pm$ 2.0e-5**	0.22 $\pm$ 0.0015	0.05 $\pm$ 0.00099	1 $\pm$ 0	1 $\pm$ 0	0.59 $\pm$ 0.065
O5	**2.9e-4 $\pm$ 2.2e-5**	0.40 $\pm$ 0.0074	0.09 $\pm$ 0.00037	1 $\pm$ 0	1 $\pm$ 0	0.64 $\pm$ 0.068
O6	**2.2e-4 $\pm$ 1.8e-5**	0.23 $\pm$ 0.0025	0.05 $\pm$ 0.00073	1 $\pm$ 0	1 $\pm$ 0	0.61 $\pm$ 0.066
O7	**1.3e-3 $\pm$ 1.0e-4**	0.62 $\pm$ 0.019	0.22 $\pm$ 0.0033	1 $\pm$ 0	1 $\pm$ 0	0.75 $\pm$ 0.043
O8	**1.1e-3 $\pm$ 1.7e-4**	0.44 $\pm$ 0.0059	0.15 $\pm$ 0.0021	1 $\pm$ 0	1 $\pm$ 0	0.71 $\pm$ 0.042
O9	**1.1e-3 $\pm$ 9.6e-5**	0.12 $\pm$ 0.0021	0.07 $\pm$ 0.0019	1 $\pm$ 0	1 $\pm$ 0	0.68 $\pm$ 0.037
O10	**1.2e-3 $\pm$ 1.1e-4**	0.40 $\pm$ 0.11	0.25 $\pm$ 0.0045	1 $\pm$ 0	1 $\pm$ 0	0.82 $\pm$ 0.044
O11	**1.1e-3 $\pm$ 9.5e-5**	0.49 $\pm$ 0.01	0.18 $\pm$ 0.0027	1 $\pm$ 0	1 $\pm$ 0	0.75 $\pm$ 0.046
O12	**1.0e-3 $\pm$ 6.0e-5**	0.40 $\pm$ 0.071	0.10 $\pm$ 0.0015	1 $\pm$ 0	1 $\pm$ 0	0.70 $\pm$ 0.043
O13	**3.3e-3 $\pm$ 2.7e-5**	0.09 $\pm$ 0.0029	0.09 $\pm$ 0.00093	1 $\pm$ 0	1 $\pm$ 0	0.71 $\pm$ 0.061
O14	**3.7e-3 $\pm$ 3.1e-5**	0.13 $\pm$ 0.0022	0.11 $\pm$ 0.0024	1 $\pm$ 0	1 $\pm$ 0	0.74 $\pm$ 0.053
O15	**2.8e-3 $\pm$ 1.4e-5**	0.06 $\pm$ 0.00051	0.06 $\pm$ 0.00099	1 $\pm$ 0	1 $\pm$ 0	0.68 $\pm$ 0.059

5 Conclusion

In this paper, a multi-output Gaussian process regression model is proposed. The reduced-rank Gaussian processes are constructed from neural network feature maps. The correlations between tasks are represented by the shared layers of the neural network. The experimental results show that our proposed model outperforms independent GP model and other state-of-the-art multi-output GP models.

Table 5: The NLL statistics of the OpAmp dataset

Output	MTNN-GP	IGP	COGP	SCGP	GPRN-NPV	GPRN-MF
O1	**-2.3** ± **0.036**	0.43 ± 0.0058	4.7 ± 0.22	1.8 ± 0.17	NA	NA
O2	**-2.5** ± **0.024**	0.18 ± 0.011	3.2 ± 0.13	1.9 ± 0.15	NA	NA
O3	**-2.4** ± **0.032**	-0.1 ± 0.0035	3.2 ± 0.14	1.6 ± 0.11	NA	NA
O4	**-3** ± **0.04**	-1.1 ± 0.0084	2 ± 0.068	1.5 ± 0.16	NA	NA
O5	**-3** ± **0.043**	-1.1 ± 0.029	2.1 ± 0.17	1.5 ± 0.11	NA	NA
O6	**-3** ± **0.035**	-0.95 ± 0.014	1.8 ± 0.11	1.7 ± 0.17	NA	NA
O7	**-2.1** ± **0.051**	-0.73 ± 0.01	5.7 ± 0.35	1.5 ± 0.049	NA	NA
O8	**-2.3** ± **0.057**	-1 ± 0.056	4.9 ± 0.27	1.5 ± 0.038	NA	NA
O9	**-2.2** ± **0.031**	-1 ± 0.016	2.5 ± 0.19	1.6 ± 0.1	NA	NA
O10	**-2** ± **0.059**	0.33 ± 0.062	5.1 ± 0.53	1.8 ± 0.11	NA	NA
O11	**-2.2** ± **0.064**	-0.046 ± 0.015	4 ± 0.095	1.8 ± 0.099	NA	NA
O12	**-2.1** ± **0.032**	-0.27 ± 0.016	3.6 ± 0.28	1.7 ± 0.069	NA	NA
O13	**-2.7** ± **0.032**	0.051 ± 0.0075	2.7 ± 0.14	1.6 ± 0.17	NA	NA
O14	**-2.7** ± **0.034**	-0.046 ± 0.0025	2.5 ± 0.25	1.7 ± 0.22	NA	NA
O15	**-2.7** ± **0.03**	-0.2 ± 0.0036	2.5 ± 0.2	1.5 ± 0.06	NA	NA

References

[1] Yu Zhang and Qiang Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017.

[2] Mauricio A Alvarez, Lorenzo Rosasco, Neil D Lawrence, et al. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266, 2012.

[3] Andre G Journel and Ch J Huijbregts. *Mining geostatistics*. Academic press, 1978.

[4] Edwin V Bonilla, Kian M Chai, and Christopher Williams. Multi-task gaussian process prediction. In *Advances in neural information processing systems*, pages 153–160, 2008.

[5] Trung V Nguyen, Edwin V Bonilla, et al. Collaborative multi-output gaussian processes. In *UAI*, pages 643–652, 2014.

[6] Andrew Gordon Wilson, David A Knowles, and Zoubin Ghahramani. Gaussian process regression networks. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1139–1146. Omnipress, 2012.

[7] Phillip Boyle and Marcus Frean. Dependent gaussian processes. In *Advances in neural information processing systems*, pages 217–224, 2005.

[8] Mauricio Alvarez and Neil D Lawrence. Sparse convolved gaussian processes for multi-output regression. In *Advances in neural information processing systems*, pages 57–64, 2009.

[9] Mauricio A Álvarez and Neil D Lawrence. Computationally efficient convolved multiple output gaussian processes. *Journal of Machine Learning Research*, 12(May):1459–1500, 2011.

[10] Miguel Lázaro-Gredilla and Aníbal R Figueiras-Vidal. Marginalized neural network mixtures for large-scale regression. *IEEE transactions on neural networks*, 21(8):1345–1351, 2010.

[11] Wen-bing Huang, Deli Zhao, Fuchun Sun, Huaping Liu, and Edward Y Chang. Scalable gaussian process regression using deep neural networks. In *IJCAI*, pages 3576–3582, 2015.

[12] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[13] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180, 2015.

[14] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6405–6416, 2017.

[15] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[16] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[17] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[18] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, 2015.

[19] Eleftherios Spyromitros-Xioufis, Grigorios Tsoumakas, William Groves, and Ioannis Vlahavas. Multi-target regression via input space expansion: treating targets as inputs. *Machine Learning*, 104(1):55–98, 2016.

[20] Athanasios Tsanas and Angeliki Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567, 2012.

[21] Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (GPML) toolbox. *Journal of Machine Learning Research*, 11(Nov):3011–3015, 2010.

[22] Trung Nguyen and Edwin Bonilla. Efficient variational inference for gaussian process regression networks. In *Artificial Intelligence and Statistics*, pages 472–480, 2013.

[23] Amir Dezfouli and Edwin V Bonilla. Scalable inference for gaussian process models with black-box likelihoods. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1414–1422. Curran Associates, Inc., 2015.