SCALABLE GAUSSIAN PROCESS USING INEXACT ADMM FOR BIG DATA

Yue Xu[†*], *Feng Yin*[*], *Jiawei Zhang*[*], *Wenjun Xu*[†], *Shuguang Cui*[‡*] *and Zhi-Quan Luo*[*]

[†]Key Lab of Universal Wireless Communications, Ministry of Education
Beijing University of Posts and Telecommunications
[*] The Chinese University of Hong Kong, Shenzhen and SRIBD
[‡]Department of Electrical and Computer Engineering, University of California, Davis

ABSTRACT

Gaussian process (GP) for machine learning has been well studied over the past two decades and is now widely used in many sectors. However, the design of low-complexity GP models still remains a challenging research problem. In this paper, we propose a novel scalable GP regression model for processing big datasets, using a large number of parallel computation units. In contrast to the existing methods, we solve the classic maximum likelihood based hyper-parameter optimization problem by a carefully designed distributed alternating direction method of multipliers (ADMM). The proposed method is parallelizable over a large number of computation units. Simulation results confirm the benefits of the proposed scalable GP model over the state-of-the-art distributed methods.

***Index Terms*—** ADMM, big data, Gaussian process, hyper-parameter optimization, scalable model.

1. INTRODUCTION AND RELATED WORKS

Gaussian process (GP) model is a class of important Bayesian non-parametric models for machine learning and tightly related to several other learning models, such as the support vector machine (SVM), single-layer Bayesian neural network, regularized-least-squares, relevance vector machine, and auto-regressive-moving-average (ARMA) scheme [1]. The key of GP models is to first set a Gaussian prior to the underlying function/system $f(\boldsymbol{x})$ and then compute the posterior over the function given the observed data. GP models have been well studied and used in a plethora of applications since its introduction due to its outstanding performance in function approximation with a natural uncertainty bound. GP models are also found to be an important component in Bayesian optimization [2].

However, the major drawback of the GP models for both regression and classification lies in the high computational complexity for finding the optimal set of hyper-parameters, which scales as $\mathcal{O}(n^3)$ with n the number of training samples. Thus, the standard GP is computationally demanding when the dataset is large. Low-complexity GP models can be obtained by exploring 1) low-rank kernel matrix approximation [3]; 2) local structures of the kernel matrix [4]; 3) state-space model reformulation and Kalman filter [5]; 4) Bayesian committee machine (BCM) using a number of distributed computing units [6, 7]; 5) sparse representations by choosing a subset of the data with some information criterion [8]; and 6) variational Bayesian formulation [9].

Among the aforementioned works, the BCM-based models in [6, 7] share some similarity with our work herein in the sense that they all deploy a large number k of parallel computation units. Generally speaking, the BCM based distributed GP models approximate the likelihood function over the full training dataset by a product of multiple local likelihood functions over subsets of the training dataset, which is equivalent to making a block-diagonal matrix assumption on the true covariance matrix. Although the computational complexity scales as $\mathcal{O}(n^3/k^3)$, their approximation accuracy becomes worse when using a large number of computing units, each accessing a small subset, due to the loss of correlation information across different subsets of data.

In this paper, we aim to propose a scalable GP model that is able to handle big data by distributing the computational complexity of the standard GP to a large number of parallel computing units. No approximation or information loss is involved in the proposed framework. The main contributions are summarized as follows. First, we make the original maximum likelihood hyper-parameter estimation distributed by introducing k (equal to the number of computing units) local copies of the global hyper-parameters, optimized by the celebrated alternating direction method of multipliers (ADMM), which is able to break a large optimization problem into smaller pieces that are easier to handle [10]. Compared with the existing distributed GP models [6, 7], our approach does not involve any approximation essentially. Second, in each ADMM iteration, thanks to the introduced local copies of the global hyper-parameter, only a small part of the large covariance matrix inverse over the complete dataset is needed to update the gradient, which helps to avoid multiplications of large matrices. Third, the proposed practical implementation of the scalable GP model adopts the Gauss-Seidel method for numerically solving matrix inverse in a parallel way, which distributes the computation load of the standard $\mathcal{O}(n^3)$ GP method to k computing units evenly, resulting in a reduced complexity of $\mathcal{O}(n^3/k)$. Experimental results show that the proposed scalable GP performs similarly to the standard GP, but with a much reduced complexity. It also outperforms the state-of-the-art distributed GP methods significantly, which makes it a promising GP implementation for big data applications.

The remainder of this paper is organized as follows. Section 2 briefly reviews the standard GP regression model. Section 3 presents the proposed scalable GP model. Section 4 contains the experimental results. Finally, Section 5 concludes this paper.

The work was supported in part by the National Natural Science Foundation of China under Grant 61871057 and 61629101, National Natural Science Foundation of China Key Project under Grant 61731018 and Young Scholar Project under Grant 61701426, by NSF under Grants DMS-1622433, AST-1547436, and ECCS-1659025, by Guangdong Province Grant 2017ZT07X152, by Shenzhen Fundamental Research Fund under Grants No. KQTD2015033114415450, No. ZDSYS201707251409055, and JCYJ20170307155957688. (Feng Yin and Yue Xu are the co-first-authors.)

2. STANDARD GP MODEL

2.1. GP Regression

A GP is a collection of random variables, any finite number of which follow a Gaussian distribution [1]. In the sequel, we focus on real-valued GPs that are completely specified by their mean functions and kernel functions as

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta})), \tag{1}$$

where $m(\boldsymbol{x})$ is the mean function, which is often set to zero in practice, and $k(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta})$ is the kernel function with the hyper-parameters $\boldsymbol{\theta}$ to be optimized. Commonly used kernels include the squared-exponential (SE) kernel, rational quadratic kernel, Matérn kernel, etc. More advanced kernels, e.g., [11, 12, 13] could also be used for specific applications.

We consider the following GP based regression model $y = f(\boldsymbol{x}) + e$, where $y \in \mathbb{R}$ is a continuous-value output, the underlying function $f(\boldsymbol{x})$ is modeled by a zero mean GP for simplicity, and the noise e is assumed to be Gaussian distributed with zero mean and variance σ^2. It is not difficult to estimate the independent noise variance σ^2 jointly with $\boldsymbol{\theta}$; but in this paper we assume it is already estimated, e.g., by the robust smoothing method in [14]. Given a training dataset $\mathcal{D} \triangleq \{\boldsymbol{X}, \boldsymbol{y}\}$, where $\boldsymbol{y} = [y_1, y_2, \cdots, y_n]^T$ is the training output and $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]$ is the training input, the aim of GP regression is to predict the output $\boldsymbol{y}_* = [y_{*,1}, y_{*,2}, \cdots, y_{*,n}]^T$, given the test inputs $\boldsymbol{X}_* = [\boldsymbol{x}_{*,1}, \boldsymbol{x}_{*,2}, \cdots, \boldsymbol{x}_{*,n}]$ with a posterior distribution $p(\boldsymbol{y}_* | \mathcal{D}, \boldsymbol{X}_*; \boldsymbol{\theta})$.

2.2. Standard GP Hyper-parameter Optimization

The predictive performance of GP regression depends on the goodness of the model parameters $\boldsymbol{\theta}$. The dominant formulation to tune the model parameters is via maximizing the marginal likelihood function $p(\boldsymbol{y}; \boldsymbol{\theta})$. For GP regression, the model parameters can be tuned equivalently by minimizing the following negative log-likelihood function [1]:

$$\begin{aligned} \mathcal{P}_0: \quad & \arg\min_{\boldsymbol{\theta}} \; g(\boldsymbol{\theta}) \triangleq \boldsymbol{y}^T \boldsymbol{C}^{-1}(\boldsymbol{\theta}) \boldsymbol{y} + \log |\boldsymbol{C}(\boldsymbol{\theta})|, \\ & \text{s.t.} \quad \boldsymbol{\theta} \in \Theta, \end{aligned} \tag{2}$$

where $|\cdot|$ stands for matrix determinant, $\boldsymbol{C}(\boldsymbol{\theta}) \triangleq \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}; \boldsymbol{\theta}) + \sigma^2 \boldsymbol{I}_n$ is the covariance matrix, and $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}; \boldsymbol{\theta})$ is the kernel matrix of the training inputs $\boldsymbol{X}$.

The gradient descent method is the benchmark hyper-parameter optimization method most widely used in the GP community. In particular, at each iteration, the hyper-parameter is updated as

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^r - \mu \cdot \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta})|_{\boldsymbol{\theta} = \boldsymbol{\theta}^r}, \tag{3}$$

where μ is the step size, and $\boldsymbol{\theta} = [\theta_1, \theta_2, \cdots, \theta_p]^T$ with p the dimension of $\boldsymbol{\theta}$. The partial derivative of the j-th element of $\boldsymbol{\theta}$ in $\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta})$ is computed, according to [1], as

$$\frac{\partial g(\boldsymbol{\theta})}{\partial \theta_j} = \mathrm{Tr}\left(\boldsymbol{C}^{-1}(\boldsymbol{\theta}) \frac{\partial \boldsymbol{C}(\boldsymbol{\theta})}{\partial \theta_j} \right) - \boldsymbol{y}^T \boldsymbol{C}^{-1}(\boldsymbol{\theta}) \frac{\partial \boldsymbol{C}(\boldsymbol{\theta})}{\partial \theta_j} \boldsymbol{C}^{-1}(\boldsymbol{\theta}) \boldsymbol{y}, \tag{4}$$

where $\mathrm{Tr}(\cdot)$ represents the matrix trace. Note that at each iteration, $\boldsymbol{C}^{-1}(\boldsymbol{\theta})$ has to be re-evaluated and the multiplications of large matrices of size $n \times n$ have to be performed several times. The total computational complexity per iteration scales as $\mathcal{O}(n^3)$, when the covariance matrix has no favorable structures, such as Toeplitz or banded.

3. SCALABLE GP MODEL USING INEXACT ADMM

3.1. Scalable GP Hyper-parameter Optimization

Our goal here is to break the original maximum likelihood hyper-parameter estimation problem $\mathcal{P}_0$ into smaller pieces that are easier to handle without making any approximations. In other words, we aim to distribute the high computation load of solving a big optimization problem to a bunch of local computing units that can work in parallel. By introducing a set of k local variables $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k$, denoted as $\{\boldsymbol{\theta}_i\}$ where $i \in \mathcal{K}$, $\mathcal{K} = \{1, 2, \cdots, k\}$, and a global variable $\boldsymbol{z}$, we can write the original problem $\mathcal{P}_0$ equivalently as

$$\begin{aligned} \mathcal{P}_1: \; & \arg\min_{\{\boldsymbol{\theta}_i\}} \; g(\{\boldsymbol{\theta}_i\}), \\ & \text{s.t.} \quad \boldsymbol{\theta}_i - \boldsymbol{z} = \boldsymbol{0}, \quad \boldsymbol{\theta}_i \in \Theta, \quad i \in \mathcal{K}, \end{aligned} \tag{5}$$

where $g(\{\boldsymbol{\theta}_i\}) \triangleq \boldsymbol{y}^T \boldsymbol{C}^{-1}(\{\boldsymbol{\theta}_i\}) \boldsymbol{y} + \log |\boldsymbol{C}(\{\boldsymbol{\theta}_i\})|$. Here, we let the i-th block of $\boldsymbol{C}(\{\boldsymbol{\theta}_i\})$ be determined by $\boldsymbol{\theta}_i$. It is worth noting that the block partition of $\boldsymbol{C}(\{\boldsymbol{\theta}_i\})$ is flexible, which solely depends on how we allocate the variable blocks to the computing units. For example, a covariance matrix $\boldsymbol{C}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4)$ with $n = 4$ data samples can be partitioned evenly into $k = 4$ square blocks as

$$\boldsymbol{C}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4) = \left[\begin{array}{cc|cc} a_{11}^{\boldsymbol{\theta}_1} & a_{12}^{\boldsymbol{\theta}_1} & a_{13}^{\boldsymbol{\theta}_2} & a_{14}^{\boldsymbol{\theta}_2} \\ a_{21}^{\boldsymbol{\theta}_1} & a_{22}^{\boldsymbol{\theta}_1} & a_{23}^{\boldsymbol{\theta}_2} & a_{24}^{\boldsymbol{\theta}_2} \\ \hline a_{31}^{\boldsymbol{\theta}_3} & a_{32}^{\boldsymbol{\theta}_3} & a_{33}^{\boldsymbol{\theta}_4} & a_{34}^{\boldsymbol{\theta}_4} \\ a_{41}^{\boldsymbol{\theta}_3} & a_{42}^{\boldsymbol{\theta}_3} & a_{43}^{\boldsymbol{\theta}_4} & a_{44}^{\boldsymbol{\theta}_4} \end{array} \right], \tag{6}$$

where $a_{ij}^{\boldsymbol{\theta}_k} = k(\boldsymbol{x}_i, \boldsymbol{x}_j; \boldsymbol{\theta}_k) + \sigma^2 \cdot \delta(i = j)$ with $\delta = 1$ if $i = j$, or $\delta = 0$ otherwise, is one single element of the full covariance matrix $C(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4)$ at the i-th row and the j-th column, and is determined by $\boldsymbol{\theta}_k$ associated with the k-th computing unit, In this paper, we assume that $l = \sqrt{k}$ and n/l are integers for simplicity. In this way, the $n \times n$ covariance matrix $\boldsymbol{C}(\{\boldsymbol{\theta}_i\})$ can be partitioned evenly into $l \times l$ square blocks, each of size $n/l \times n/l$.

Note that the global minimum of $\mathcal{P}_1$ is the same as that of $\mathcal{P}_0$ [15]. One of the major benefits of the reformulation from $\mathcal{P}_0$ to $\mathcal{P}_1$ is the flexibility of allowing each local computing unit to focus on its own local variable $\boldsymbol{\theta}_i$ [15]. Specifically, in this paper, we propose to solve $\mathcal{P}_1$ via the ADMM. The augmented Lagrangian function can be written as

$$\mathcal{L}(\{\boldsymbol{\theta}_i\}, \boldsymbol{z}, \boldsymbol{\beta}) \triangleq g(\{\boldsymbol{\theta}_i\}) + \sum_{i=1}^{k} \boldsymbol{\beta}_i^T (\boldsymbol{\theta}_i - \boldsymbol{z}) + \sum_{i=1}^{k} \frac{\rho}{2} \parallel \boldsymbol{\theta}_i - \boldsymbol{z} \parallel_2^2 . \tag{7}$$

For any $i \in \mathcal{K}$, the sequential update of the parameters at the $(r+1)$-th iteration can be written as

$$\boldsymbol{\theta}_i^{r+1} = \arg\min_{\boldsymbol{\theta}_i} g(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r) + \boldsymbol{\beta}_i^{r,T} (\boldsymbol{\theta}_i - \boldsymbol{z}^r) + \frac{\rho}{2} \parallel \boldsymbol{\theta}_i - \boldsymbol{z}^r \parallel_2^2 \tag{8a}$$

$$\boldsymbol{z}^{r+1} = \frac{1}{k} \sum_{i=1}^{k} \left(\boldsymbol{\theta}_i^{r+1} + \frac{1}{\rho} \boldsymbol{\beta}_i^r \right), \tag{8b}$$

$$\boldsymbol{\beta}_i^{r+1} = \boldsymbol{\beta}_i^r + \rho (\boldsymbol{\theta}_i^{r+1} - \boldsymbol{z}^{r+1}), \tag{8c}$$

where $g(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r) \triangleq \boldsymbol{y}^T \boldsymbol{C}^{-1}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r) \boldsymbol{y} + \log |\boldsymbol{C}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)|$ with $\boldsymbol{z}_{-i}^r \triangleq \{\boldsymbol{\theta}_1^r, \cdots, \boldsymbol{\theta}_{i-1}^r, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}^r, \cdots, \boldsymbol{\theta}_k^r\} \setminus \boldsymbol{\theta}_i$ containing all previous estimates of the local variables except for $\boldsymbol{\theta}_i$. Here $\boldsymbol{C}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)$ comprises one block determined by $\boldsymbol{\theta}_i$ and the other blocks determined by $\boldsymbol{z}_{-i}^r$ which is *fixed* at the $(r+1)$-th iteration. Note that the objective function in (8a) is non-convex and minimizing it exactly to get the global minimum in each ADMM iteration may not be easy, especially in the early stage of the algorithm. Therefore, in this paper, we propose to use an inexact step to update the local $\boldsymbol{\theta}_i$ as $\boldsymbol{\theta}_i^{r+1} = \boldsymbol{\theta}_i^r - \mu \cdot \nabla_{\boldsymbol{\theta}_i} \mathcal{L}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r, \boldsymbol{z}^r, \boldsymbol{\beta}^r)|_{\boldsymbol{\theta}_i = \boldsymbol{\theta}_i^r}$, which corresponds to updating it once with gradient descent, followed by the consensus immediately. We can derive $\nabla_{\boldsymbol{\theta}_i} \mathcal{L}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r, \boldsymbol{z}^r, \boldsymbol{\beta}^r)|_{\boldsymbol{\theta}_i = \boldsymbol{\theta}_i^r}$ as in (9b) (next page). For clarity, following the example given in (6), the covariance matrix $\boldsymbol{C}(\boldsymbol{\theta}_1, \boldsymbol{z}_{-1}^r)$ of the first computing unit at the $(r+1)$-th iteration can be written as

$$\boldsymbol{C}(\boldsymbol{\theta}_1, \boldsymbol{z}_{-1}^r) = \left[\begin{array}{cc|cc} a_{11}^{\boldsymbol{\theta}_1} & a_{12}^{\boldsymbol{\theta}_1} & a_{13}^{\boldsymbol{\theta}_2^r} & a_{14}^{\boldsymbol{\theta}_2^r} \\ a_{21}^{\boldsymbol{\theta}_1} & a_{22}^{\boldsymbol{\theta}_1} & a_{23}^{\boldsymbol{\theta}_2^r} & a_{24}^{\boldsymbol{\theta}_2^r} \\ \hline a_{31}^{\boldsymbol{\theta}_3^r} & a_{32}^{\boldsymbol{\theta}_3^r} & a_{33}^{\boldsymbol{\theta}_4^r} & a_{34}^{\boldsymbol{\theta}_4^r} \\ a_{41}^{\boldsymbol{\theta}_3^r} & a_{42}^{\boldsymbol{\theta}_3^r} & a_{43}^{\boldsymbol{\theta}_4^r} & a_{44}^{\boldsymbol{\theta}_4^r} \end{array}\right]. \tag{10}$$

Note that optimizing $\boldsymbol{\theta}_i$ using (9b) requires much less computational cost than optimizing $\boldsymbol{\theta}$ using (4). Specifically, as shown in Fig. 1, with $\boldsymbol{C}(\{\boldsymbol{\theta}_i\})$ evenly partitioned into $l \times l$ square blocks, $\boldsymbol{C}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)$ has only one square block that is in terms of $\boldsymbol{\theta}_i$, e.g., marked as the dark one in Fig. 1(a). Accordingly, the partial derivative $\partial \boldsymbol{C}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)/\partial \theta_{ij}$ has only one non-zero block, e.g., marked as the dark one in Fig. 1(b). Hence, the matrix multiplication involved with $\boldsymbol{C}^{-1}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)$ in (9b) only requires knowing one horizontal slice and one vertical slice of $\boldsymbol{C}^{-1}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)$, e.g., marked as the dark slices in Fig. 1(c). Generally, for the (m,n)-th block of $\boldsymbol{C}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)$, where $m, n \in \{1, 2, \cdots, l\}$, one only needs to the know the n-th horizontal slice and the m-th vertical slice of $\boldsymbol{C}^{-1}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)$. As such, we only need to compute a small fraction of a big covariance matrix inverse over the complete dataset to update the gradient, thus avoiding multiplications of large matrices and reducing the computational complexity for each local computing unit.

Following the example given in (10), for the first computing unit, we denote the four square blocks as $\boldsymbol{C}_{11}(\boldsymbol{\theta}_1)$, $\boldsymbol{C}_{12}(\boldsymbol{\theta}_2^r)$, $\boldsymbol{C}_{21}(\boldsymbol{\theta}_3^r)$ and $\boldsymbol{C}_{22}(\boldsymbol{\theta}_4^r)$, respectively. Then, the covariance matrix can be written as

$$\boldsymbol{C}(\boldsymbol{\theta}_1, \boldsymbol{z}_{-1}^r) = \begin{bmatrix} \boldsymbol{C}_{11}(\boldsymbol{\theta}_1) & \boldsymbol{C}_{12}(\boldsymbol{\theta}_2^r) \\ \boldsymbol{C}_{21}(\boldsymbol{\theta}_3^r) & \boldsymbol{C}_{22}(\boldsymbol{\theta}_4^r) \end{bmatrix}. \tag{11}$$

We denote the inverse of the full covariance matrix as $\boldsymbol{B}$, i.e., $\boldsymbol{C}\boldsymbol{B} = \boldsymbol{I}_{n \times n}$, where $\boldsymbol{I}_{n \times n}$ is the identity matrix of size $n \times n$. Similarly, we partition $\boldsymbol{B}$ into four blocks, i.e., $\boldsymbol{B}_{11}$, $\boldsymbol{B}_{12}$, $\boldsymbol{B}_{21}$ and $\boldsymbol{B}_{22}$, satisfying

$$\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{\frac{n}{2} \times \frac{n}{2}} & \boldsymbol{0}_{\frac{n}{2} \times \frac{n}{2}} \\ \boldsymbol{0}_{\frac{n}{2} \times \frac{n}{2}} & \boldsymbol{I}_{\frac{n}{2} \times \frac{n}{2}} \end{bmatrix}. \tag{12}$$

As discussed above, evaluating (9b) does not require knowing the complete $\boldsymbol{B}$. Specifically, for any $j \in \{1, 2, ..., p\}$ with p the dimension of $\boldsymbol{\theta}$, the partial derivatives of θ_{1j} can be written as

$$\frac{\partial \boldsymbol{C}}{\partial \theta_{1j}} = \begin{bmatrix} \frac{\partial \boldsymbol{C}_{11}(\boldsymbol{\theta}_1)}{\partial \theta_{1j}} & \boldsymbol{0}_{\frac{n}{2} \times \frac{n}{2}} \\ \boldsymbol{0}_{\frac{n}{2} \times \frac{n}{2}} & \boldsymbol{0}_{\frac{n}{2} \times \frac{n}{2}} \end{bmatrix}, \tag{13}$$

where we use $\boldsymbol{C} = \boldsymbol{C}(\boldsymbol{\theta}_1, \boldsymbol{z}_{-1}^r)$ for short. Accordingly, the first computing unit only needs $\{\boldsymbol{B}_{11}, \boldsymbol{B}_{21}, \boldsymbol{B}_{12}\}$ to evaluate $\boldsymbol{C}^{-1}\frac{\partial \boldsymbol{C}}{\partial \theta_{ij}}$ and $\boldsymbol{y}^T \boldsymbol{C}^{-1} \frac{\partial \boldsymbol{C}}{\partial \theta_{ij}} \boldsymbol{C}^{-1} \boldsymbol{y}$ in (9b). Similarly, the other computing units only need $\{\boldsymbol{B}_{11}, \boldsymbol{B}_{21}, \boldsymbol{B}_{22}\}$, $\{\boldsymbol{B}_{11}, \boldsymbol{B}_{12}, \boldsymbol{B}_{22}\}$ and $\{\boldsymbol{B}_{21}, \boldsymbol{B}_{22}, \boldsymbol{B}_{12}\}$ to optimize $\boldsymbol{\theta}_2$, $\boldsymbol{\theta}_3$ and $\boldsymbol{\theta}_4$, respectively. Actually, for $l = 2$, it is easy to derive the closed-form expressions of the block-wise matrix inverses as

$$\boldsymbol{B}_{11} = \left(\boldsymbol{C}_{11} - \boldsymbol{C}_{12}\boldsymbol{C}_{22}^{-1}\boldsymbol{C}_{21}\right)^{-1}, \tag{14a}$$

$$\boldsymbol{B}_{12} = \boldsymbol{B}_{21}^T = -\boldsymbol{C}_{22}^{-1}\boldsymbol{C}_{21}\boldsymbol{B}_{11}, \tag{14b}$$

$$\boldsymbol{B}_{22} = \left(\boldsymbol{C}_{22} - \boldsymbol{C}_{21}\boldsymbol{C}_{11}^{-1}\boldsymbol{C}_{12}\right)^{-1}. \tag{14c}$$

Algorithm 1 Gauss-Seidel Method For Computing A Slice of Complete Matrix Inverse: e.g., $\boldsymbol{B}_1$

```
1: Initialization: $\epsilon$, $t = 0$, $\boldsymbol{B}_1^0$.
2: Iteration:
3: while $\|\boldsymbol{B}_1^{t+1} - \boldsymbol{B}_1^t\|_F > \epsilon$ do
4:     $t = t + 1$.
5:     $\boldsymbol{B}_{11}^{t+1} = \boldsymbol{C}_{11}^{-1}\left(\boldsymbol{I}_{\frac{n}{2} \times \frac{n}{2}} - \sum_{i=2}^{l} \boldsymbol{C}_{1i}\boldsymbol{B}_{1i}^t\right)$. (The first unit)
6:     for $1 < j < l$ do
7:         (The $j$-th unit)
8:         $\boldsymbol{B}_{1j}^{t+1} = -\boldsymbol{C}_{jj}^{-1}\left(\sum_{i=1}^{j-1} \boldsymbol{C}_{ji}\boldsymbol{B}_{1i}^{t+1} + \sum_{i=j+1}^{l} \boldsymbol{C}_{ji}\boldsymbol{B}_{1i}^t\right)$.
9:     end for
10:    $\boldsymbol{B}_{1l}^{t+1} = -\boldsymbol{C}_{ll}^{-1}\left(\sum_{i=1}^{l-1} \boldsymbol{C}_{li}\boldsymbol{B}_{1i}^{t+1}\right)$. (The $l$-th unit)
11: end while
12: Output: $\boldsymbol{B}_{11}^*, \boldsymbol{B}_{12}^*, \cdots, \boldsymbol{B}_{1l}^*$ at convergence.
```

For $l \geq 3$, the closed-form inverse of each block can be derived, albeit in a tedious, recursive manner. Next, we will show a practical solution without deriving the closed-form expressions.

3.2. A Practical Implementation

Using (9b) on next page with closed-form expressions of $\boldsymbol{B}_{mn}$ to update $\boldsymbol{\theta}_i$ is readily tedious for $l \geq 3$ in practice. Therefore, we propose to approximate $\boldsymbol{C}^{-1}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)$ in (9b) with $\boldsymbol{C}^{-1}(\boldsymbol{z}^r)$ and use the Gauss-Seidel iterative method [16] to compute $\boldsymbol{B}_{mn}$. More specifically, we use (9c) instead of (9b) when updating $\boldsymbol{\theta}_i$ and run the Gauss-Seidel method with parallel computing units to get all blocks of $\boldsymbol{C}^{-1}(\boldsymbol{z}^r)$ computed in a similar way. After that, all computing units communicate in a way that each one gets its desired vertical slice and horizontal slice of $\boldsymbol{C}^{-1}(\boldsymbol{z}^r)$ at the end for updating the gradients according to (9c). The detailed procedure for computing the first vertical slice of $\boldsymbol{C}^{-1}(\boldsymbol{z}^r)$ using l computing units is presented in Algorithm 1, i.e., the procedure to compute an approximated solution of $\boldsymbol{B}_1 = \boldsymbol{C}^{-1}(\boldsymbol{z}^r)\left[\boldsymbol{I}_{\frac{n}{2} \times \frac{n}{2}}; \boldsymbol{0}_{\frac{n}{2} \times \frac{n}{2}}\right]$, where $\boldsymbol{B}_1 \triangleq [\boldsymbol{B}_{11}; \boldsymbol{B}_{21}; ...; \boldsymbol{B}_{l1}]$. The other vertical slices of $\boldsymbol{C}^{-1}(\boldsymbol{z}^r)$, i.e., $\boldsymbol{B}_2, \cdots, \boldsymbol{B}_l$, can be computed in the same way.

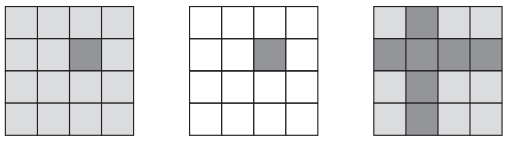
Note that in Algorithm 1: 1) every computing unit uses the full dataset; 2) all principal sub-matrices $\boldsymbol{C}_{jj}^{-1}$, $j = 1, 2, \cdots, l$, are positive definite thus invertible; 3) the computational complexity scales as $\mathcal{O}(n^3/k)$ at each computing unit. Specifically, to compute one slice of $\boldsymbol{C}^{-1}(\boldsymbol{z}^r)$, e.g., $\boldsymbol{B}_1$, with l computing units in Algorithm 1, each computing unit needs to take an inverse of one principal submatrix of size $n/l \times n/l$ and perform $l-1$ multiplications of two matrices both of size $n/l \times n/l$. Hence, when the number of iterations required by the Gauss-Seidel method to achieve a target approximation precision is small and not a function of k, the computational complexity of our scheme scales as $\mathcal{O}(n^3/l^2) = \mathcal{O}(n^3/k)$ per computing unit. Note that the overall complexity still scales as $\mathcal{O}(n^3)$ due to the fact that unlike other distributed GP models to optimize the GP hyper-parameter with approximations [6, 7], we do not make any approximations in $\mathcal{P}_1$. However, better matrix partition strategies or alternatives for the Gauss-Seidel method for computing the matrix inverse can be explored in the future to further reduce the overall computational complexity, subject to an acceptable performance loss.

The complete procedure of our proposed scalable GP hyper-parameter optimization using inexact ADMM is presented in Algorithm 2 for clarity. After the algorithm converges, each local comput-

$$\left[\nabla_{\boldsymbol{\theta}_i}\mathcal{L}(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r,\boldsymbol{z}^r,\boldsymbol{\beta}^r)|_{\boldsymbol{\theta}_i=\boldsymbol{\theta}_i^r}\right]_j = \frac{\partial}{\partial\theta_{ij}}\left(g(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r)+(\boldsymbol{\beta}_i^r)^T(\boldsymbol{\theta}_i-\boldsymbol{z}^r)+\frac{\rho}{2}\parallel \boldsymbol{\theta}_i-\boldsymbol{z}^r\parallel_2^2\right)\Big|_{\boldsymbol{\theta}_i=\boldsymbol{\theta}_i^r} \tag{9a}$$

$$= \mathrm{Tr}\left(\boldsymbol{C}^{-1}(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r)\frac{\partial\boldsymbol{C}(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r)}{\partial\theta_{ij}}\right) - \boldsymbol{y}^T\boldsymbol{C}^{-1}(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r)\frac{\partial\boldsymbol{C}(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r)}{\partial\theta_{ij}}\boldsymbol{C}^{-1}(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r)\boldsymbol{y}+\beta_{ij}^r+\rho(\theta_{ij}-z_j^r)\Big|_{\boldsymbol{\theta}_i=\boldsymbol{\theta}_i^r} \tag{9b}$$

$$\approx \mathrm{Tr}\left(\boldsymbol{C}^{-1}(\boldsymbol{z}^r)\frac{\partial\boldsymbol{C}(\boldsymbol{\theta}_i^r,\boldsymbol{z}_{-i}^r)}{\partial\theta_{ij}}\right) - \boldsymbol{y}^T\boldsymbol{C}^{-1}(\boldsymbol{z}^r)\frac{\partial\boldsymbol{C}(\boldsymbol{\theta}_i^r,\boldsymbol{z}_{-i}^r)}{\partial\theta_{ij}}\boldsymbol{C}^{-1}(\boldsymbol{z}^r)\boldsymbol{y}+\beta_{ij}^r+\rho(\theta_{ij}^r-z_j^r). \tag{9c}$$



(a) $\boldsymbol{C}(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r)$ (b) $\frac{\partial\boldsymbol{C}(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r)}{\partial\theta_{ij}}$ (c) $\boldsymbol{C}^{-1}(\boldsymbol{\theta}_i,\boldsymbol{z}_{-i}^r)$

Fig. 1. When updating its local hyper-parameter, say $\boldsymbol{\theta}_i$, the i-th local computing unit requires only one block of the full covariance matrix, e.g., the dark block in Fig. 1(a); only one block of the partial derivative matrix is non-zero, e.g., the dark block in Fig. 1(b); only one vertical and one horizontal slice of the full matrix inverse is needed, e.g., the two dark slices in Fig. 1(c), for gradient update.

ing unit can immediately compute the posterior mean and variance of a given test point conditioned on the complete data and the global hyper-parameter estimate.

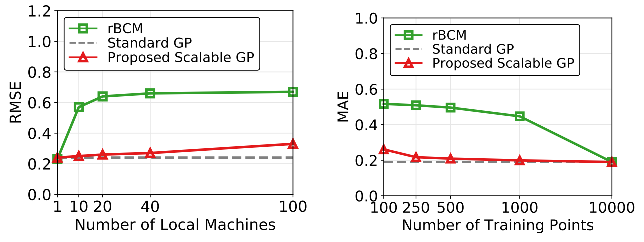
4. SIMULATION RESULTS

In this section, we test our proposed scalable GP regression model with some simulated datasets. Specifically, we use ten sets of randomly generated hyper-parameters and noises to generate ten datasets with the SE kernel. The regression task is defined as follows. For each dataset, we use 10000 data points as the full training set, and predict the next upcoming data point. Moreover, we iteratively update the training set by removing the oldest data point and adding the newest data point dynamically, i.e., advancing the training set forward in time one sample point at a time. In this way, we can repeat the regression tasks 300 times and use the average to measure the algorithm performance. For all the following experiments, we run at a workstation with eight Intel Xeon E3 CPU cores at 3.50 GHz. The local computing units are simulated by activating them with the same computing power within the workstation cyclically.

We choose two baseline models: 1) the standard GP model [1], i.e., optimizing the hyper-parameters as in $\mathcal{P}_0$; 2) the state-of-art BCM based distributed GP model, i.e., the robust BCM (rBCM) model [7]. Specifically, the rBCM approximates the likelihood function over the full training dataset by a product of multiple local likelihood functions over subsets of the training dataset when optimizing the GP hyper-parameter, and combines the local inferences with heuristic weights to generate the final outcome. The root-mean-square-error (RMSE) and the mean-absolute-error (MAE) averaged over multiple test points are used as the performance metrics. The results are as follows. In Fig. 2(a) and Fig. 2(b), we present the RMSE and MAE performances of all comparing schemes, respectively. The scalability of the rBCM based distributed GP model and our proposed scalable GP model are evaluated with 100, 40, 20, 10, 1

Algorithm 2 Scalable GP Hyper-parameter Optimization

```
1:  Initialization: $r = 0$; $\epsilon$; $\boldsymbol{z}^0$; $\boldsymbol{\theta}_i^0$; $\boldsymbol{\beta}_i^0$, $i \in \mathcal{K}$
2:  Iteration:
3:  while $\parallel \boldsymbol{\theta}_i^{r+1} - \boldsymbol{\theta}_i^r \parallel_2 > \epsilon$ do
4:      $r = r + 1$.
5:      Obtain $\boldsymbol{C}^{-1}(\boldsymbol{z}^r)$ according to Algorithm 1.
6:      for $i \in \mathcal{K}$ do
7:          Obtain the desired blocks of $\boldsymbol{C}^{-1}(\boldsymbol{z}^r)$.
8:          for $j \in \{1, 2, ..., p\}$ do
9:              Compute $\partial\boldsymbol{C}(\boldsymbol{\theta}_i, \boldsymbol{z}_{-i}^r)/\partial\theta_{ij}$.
10:             Update $\boldsymbol{\theta}_i$ according to (8a) and implemented inexactly
                via (9c).
11:         end for
12:     end for
13:     Update $\boldsymbol{z}^r$ and $\boldsymbol{\beta}_i^r$ according to (8b) and (8c), respectively.
14: end while
15: Output: $\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, ..., \boldsymbol{\theta}_k^*$ and $\boldsymbol{z}^*$ at convergence.
```



(a) Prediction Accuracy (b) Prediction Accuracy

Fig. 2. The prediction performance

local machines, trained with 100, 250, 500, 1000, 10000 data points, respectively. The result shows that our proposed scalable GP method outperforms the rBCM based distributed GP method considerably. Moreover, the scalable GP method can well-approximate the standard GP model when we increase the number of local computing units, which demonstrates the scalability of the proposed ADMM algorithm.

5. CONCLUSION

In this paper, we proposed a scalable GP method with inexact ADMM to optimize the GP hyper-parameter in a parallel way without making any approximations. Moreover, we proposed a practical implementation by adopting the Gauss-Seidel method to compute the matrix inverse. Experimental results verified that our proposed method could outperform the state-of-art rBCM and well-approximate the standard GP model, making it promising for big data applications.

6. REFERENCES

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

[2] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems (NIPS)*, Lake Tahoe, Nevada, USA, December 2012, pp. 2951–2959.

[3] C. K. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems (NIPS)*, Denver, CO, USA, December 2000, pp. 682–688.

[4] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. O'Neil, "Fast direct methods for Gaussian processes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 252–265, February 2016.

[5] S. Sarkka, A. Solin, and J. Hartikainen, "Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering," *IEEE Signal Process. Mag.*, vol. 30, no. 4, pp. 51–61, July 2013.

[6] V. Tresp, "A Bayesian committee machine," *Neural Computation*, vol. 30, no. 12, pp. 2719–2741, November 2000.

[7] M. P. Deisenroth and J. W. Ng, "Distributed Gaussian processes," in *International Conference on Machine Learning (ICML)*, Lille, France, July 2015, pp. 1481–1490.

[8] J. Quiñonero Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *J. Mach. Learn. Res.*, vol. 6, no. 1, pp. 1939–1959, December 2005.

[9] M. K. Titsias, "Variational learning of inducing variables in sparse Gaussian processes," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, Clearwater Beach, Florida, USA, April 2009, pp. 567–574.

[10] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, January 2011.

[11] Y. Xu, W. Xu, F. Yin, J. Lin, and S. Cui, "High-accuracy wireless traffic prediction: A GP-based machine learning approach," in *IEEE Global Communications Conference (GLOBECOM)*, Singapore, December 2017, pp. 1–6.

[12] F. Yin, X. He, L. Pan, T. Chen, Z.-Q. Luo, and S. Theodoridis, "Sparse structure enabled grid spectral mixture kernel for temporal Gaussian process regression," in *International Conference on Information Fusion (FUSION)*, Cambridge UK, July 2018, pp. 47–54.

[13] T. Chen, "On kernel design for regularized LTI system identification," *Automatica*, vol. 90, no. 1, pp. 109 – 122, April 2018.

[14] D. Garcia, "Robust smoothing of gridded data in one and higher dimensions with missing values," *Computational Statistics and Data Analysis*, vol. 54, no. 4, pp. 1167–1178, September 2010.

[15] M. Hong, Z. Q. Luo, and M. Razaviyayn, "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 337–364, January 2016.

[16] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 2013.