Sequential Randomized Matrix Factorization for Gaussian Processes

Shaunak D. Bopardikar George S. Eskander Ekladious

***Abstract*— The Gaussian process framework models a function as a stochastic process such that the training data results into a finite number of jointly Gaussian random variables, whose properties can then be used to infer the statistics (the mean and variance) of the function at test values for the input. The computation can be implemented in a *batch* setting, i.e., one-shot over the entire training data, or in a *sequential* setting where the data is processed incrementally. In either setting, the scalability of the computation grows with the number of points in the (training) data. This paper addresses the scalability aspect of Gaussian processes in sequential settings using recent advances in randomized matrix computations.**

***Index Terms*— Gaussian Processes Regression, Low-rank Matrix Factorization, Incremental computation, Scalability**

I. Introduction

The framework of Gaussian processes and their application to regression, classification, etc. is well established. The reader is refered to [1] for a detailed review of the methodology. The computational complexity of Gaussian processes is centered around the need to invert the *kernel matrix* whose size is governed by the number of training data points. Therefore, the worst-case computational complexity of the method is $O(n^3)$, where n is the number of training points. Over the past decade, several approaches have been proposed to improve the scalability using sparse approximations [2], [3]. Representing the Gaussian processes using a set of $k \ll n$ basis vectors, the computational complexity can be lowered down to $O(k^2 n)$ but in a batch setting. [4] applied randomized matrix factorization techniques to compute low rank approximations of the kernel matrix. Recently, [5] showed that for some classes of kernel functions, the covariance matrix can be hierarchically factored into a product of block low-rank updates of the identity matrix, thereby leading to an $O(n \log^2(n))$ algorithm for inversion.

Recently, [6] developed recursive approaches for streaming Gaussian processes based on updating the basis vectors set efficiently at every time step followed by a step involving learning/updating the hyperparameters (variables that define the kernel matrix). The complexity of their basic approach scales as $O(bk^2)$, where b is the size of the new samples at every time step.

In this paper, a sequential randomized low-rank matrix factorization approach is proposed for incrementally predicting values of an unknown function at test points using the Gaussian Processes framework. We propose an approach to compute the inverse using randomized matrix factorization algorithms in a streaming scenario, i.e., data is generated incrementally over time. Accuracy of the proposed method is compared with a state-of-the-art streaming Gaussian Process method from [6] on two publicly available real datasets.

The authors are with United Technologies Research Center. E-mail addresses: bopardsd@utrc.utc.com, ekladigs@utrc.utc.com. This work was supported by the Office of Naval Research under the contract N00014-16-C-2009.

II. Problem Formulation and Background

For a training set $\mathcal{S} := (\mathbf{X}, \mathbf{y}) = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_t, y_t)$, the goal is to estimate the output of the underlying unknown function f at a new point $\mathbf{x}_{t+1}$. A Gaussian process (GP) is a stochastic process with the property that any finite number of random variables taken from a realization of the GP, follows a joint Gaussian distribution. A GP is completely defined by the mean function and the covariance function. For a finite set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$, if $f(\mathbf{X}) = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_t)]^T$, then

$$f(\mathbf{X}) \sim \mathcal{N}(m(\mathbf{X}), k(\mathbf{X}, \mathbf{X})),$$

where $m(\mathbf{X}) = [m(\mathbf{x}_1), ..., m(\mathbf{x}_t)]^T$ and $k(\mathbf{X}, \mathbf{X})$ is the kernel matrix which is symmetric positive semidefinite with $k(\mathbf{x}_i, \mathbf{x}_j) = h(\mathbf{x}_i, \mathbf{x}_j, \phi)$ where ϕ is a set of parameters defining the kernel. Without any loss of generality, we assume that the mean vector is zero. For a test input vector $\mathbf{x}_{t+1}$, the posterior distribution is given by

$$p(f(\mathbf{x}_{t+1})|\mathcal{S}, \mathbf{x}_{t+1}, \phi) = \mathcal{N}(f_*(\mathbf{x}_{t+1}), k_*(\mathbf{x}_{t+1}, \mathbf{x}_{t+1})),$$

where

$$\begin{aligned} f_* &= \mathbf{k}_{\mathbf{x}_{t+1}}^T \mathcal{I}^{-1} \mathbf{y}, \\ k_* &= k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}_{\mathbf{x}_{t+1}}^T \mathcal{I}^{-1} \mathbf{k}_{\mathbf{x}_{t+1}}, \end{aligned} \tag{1}$$

where $\mathcal{I} := k(\mathbf{X}, \mathbf{X}) + \sigma^2 I$, is the regularized covariance matrix, and $\mathbf{k}_{\mathbf{x}_{t+1}}^T := \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_{t+1}) & \ldots & k(\mathbf{x}_t, \mathbf{x}_{t+1}) \end{bmatrix}^T$, is the cross-covariance of $\mathbf{x}_{t+1}$ with the training set.

The GP computations become infeasible for large training sets since the computational complexity of computing $\mathcal{I}^{-1}$ is $O(n^3)$, where n is the number of training points. Instead of inverting the exact regularized covariance matrix, we compute an approximate eigen decomposition of the matrix so that the matrix inversion is more scalable.

To this end, we employ the Fixed Rank Randomized Range Finder Algorithm 1 (Algo. 4.1 from [7]). Given a positive semidefinite matrix $K \in \mathbb{R}^{n \times n}$ and a positive integer p, steps 2-4 yield a matrix $Q \in \mathbb{R}^{n \times k}$ $(k < n)$ such that with probability at least $1 - 3p^{-p}$,

$$\|K - QQ^T K\| \leq \left(1 + 9\sqrt{k+p}\sqrt{n}\right)\sigma_{k+1}(K), \tag{2}$$

where σ_{k+1} is the $k+1$-th singular value of K.

Then, steps 5-7 yield an approximate eigenvalue decomposition of $K \approx UDU^T$. Using the factored form of the regularized covariance matrix, we can use the Woodbury identity [4] to compute $\mathcal{I}^{-1}$, where

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

Algorithm 1 Randomized Eigen-decomposition [7]

1: **Input:** $K \in \mathbb{R}^{n\times n}$, $k < n$, positive integer p.
2: Draw a random $n \times (k + p)$ test matrix Ω.
3: Form the matrix product $Y = K\Omega$.
4: Orthonormalize columns of Y and set them equal to Q such that $K \approx QQ^TK$ with high probability.
5: Form the small matrix $Z = Q^TKQ$.
6: Compute the eigenvalue decomposition $Z = VDV^T$.
7: Set $U = QV$.
8: **Output:** U, D such that $K \approx UDU^T$.

$$\mathcal{I}^{-1} = \sigma^{-2}I - \sigma^{-4}U(D^{-1} + \sigma^{-2}I)^{-1}U^T. \qquad (3)$$

The computational complexity of computing $\mathcal{I}^{-1}$ using Algorithm 1 is $O(kn^2 + k^2n) < O(n^3)$ for $k < n$, where the dominant cost is that of computation of Q.

III. Application to Streaming Gaussian Processes

Algorithm 1 is suitable for batch mode Gaussian processes, where all training data are available before prediction. For the streaming scenario, however, the computations are not efficient since the factorization process runs from scratch once a new batch of data arrives. Here, we propose a more efficient algorithm for sequential Gaussian process predictions. The central idea behind improving the computational complexity is to use the decomposition $K_t \approx U_tD_tU_t^T$ obtained at iteration t to compute the corresponding decomposition at iteration $t + 1$. Let the kernel matrix at iteration $t + 1$ be

$$K_{t+1} := \begin{bmatrix} K_t & \mathbf{k}_{\mathbf{x}_{t+1}} \\ \mathbf{k}_{\mathbf{x}_{t+1}}^T & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix},$$

where $\mathbf{k}_{\mathbf{x}_{t+1}}, k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1})$ are the covariances of $\mathbf{x}_{t+1}$ with itself and with the training set, respectively. Now, to compute the factorization $K_{t+1} \approx \bar{U}_{t+1}\bar{D}_{t+1}\bar{U}_{t+1}^T$, we will use the matrix

$$\bar{K}_{t+1} := \begin{bmatrix} U_tD_tU_t^T & \mathbf{k}_{\mathbf{x}_{t+1}} \\ \mathbf{k}_{\mathbf{x}_{t+1}}^T & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix},$$

in place of K_{t+1}. The action of matrix K_{t+1} on a random test vector Ω, which is computationally the most expensive step, is now substituted by

$$Y = \begin{bmatrix} U_tD_tU_t^T & \mathbf{k}_{\mathbf{x}_{t+1}} \\ \mathbf{k}_{\mathbf{x}_{t+1}}^T & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}, \qquad (4)$$

where the random vector $\Omega = [\omega_1; \omega_2]$, has been appropriately partitioned.

The main advantage of using the proposed approach over batch one is that the computational bottleneck of matrix-matrix multiplication (step 3 in Algorithm 1) is reduced from $O(n^2k)$ down to $O(nk \log k)$ per iteration. The overall technique is summarized in Algorithm 2.

Two datasets are used for the comparative study. The first dataset is the *Sarcos Robot arm* dataset which is publicly available at [8]. The data relates to an inverse dynamics problem for a seven degrees-of-freedom SARCOS anthropomorphic robot arm. The task is to map from a 21-dimensional input space (7 joint positions, 7 joint velocities, 7 joint accelerations) to the corresponding 7 joint torques. In our simulations, we predict the first joint's torque (the 22^{nd} column) using the full input space. The first 5000 samples of the training dataset are used for simulations.

Algorithm 2 Sequential Randomized GP Regression

1: **Initialization:** At $t = 1$, compute K_1 using a Kernel function and the initial input x_1.
2: Compute U_1, D_1 such that $K_1 = U_1D_1U_1^T$, using Algorithm 1.
3: **for** $t \in \{2, 3, \dots\}$ **do**
4: Compute cross covariances $\mathbf{k}_{\mathbf{x}_t}, k(\mathbf{x}_t, \mathbf{x}_t)$ using $\mathbf{x}_t$.
5: Form

$$\bar{K}_t := \begin{bmatrix} U_{t-1}D_{t-1}U_{t-1}^T & \mathbf{k}_{\mathbf{x}_t} \\ \mathbf{k}_{\mathbf{x}_t}^T & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix},$$

6: Using of Algorithm 1, compute U_t, D_t such that $\bar{K}_t = U_tD_tU_t^T$, where (4) is applied in step 3.
7: Compute $\mathcal{I}^{-1} = \sigma^{-2}I - \sigma^{-4}U_t(D_t^{-1} + \sigma^{-2}I)^{-1}U_t^T$
8: Predict outputs for the new input $\mathbf{x}_t$ using (1).
9: **end for**

The second is the *Abalone* dataset, from the UCI machine learning database, where the age of abalone is predicted given other eight attributes [9]. For implementing the sequential learning scenario, only the first 4000 samples of the training set are used. For both datasets, the batch size is set to 100 samples, where the actual values (torques/ages) of first batch are presented to initiate the GP. The GP is updated during subsequent iterations using the predicted values.

For all simulations, a squared exponential kernel is employed for the covariance computations. The hyperparameters are empirically estimated and same values are used for all the algorithms included in this comparative study.

Here we compare the proposed method with the state-of-the-art deterministic streaming GP technique [6]. In this technique, no matrix factorization step is applied, however, the GP computational cost is significantly reduced through fixing the number k of basis vectors. Whenever new batch arrives, the basis vectors are updated recursively to capture the newly introduced data. The computational complexity of this approach is $O(bk^2)$, where b is the batch size. This computational cost is fixed since k is fixed over all iterations. For short, we will refer to this method by Deterministic Recursive Gaussian Process (DRGP).

In our proposed method, however, the size of the training data is being augmented over time and the sequential random matrix factorization approach is employed to reduce the computation cost. We refer to our method by Sequential Random Matrix Factorization (SRMF).

Figures 1 and 2 show the computational time taken by the DRGP and SRMF algorithms for the Sarcos and Abalone datasets, respectively. While the DRGB has no significant computational variation, the proposed SRMF algorithm shows increasing computational time when a new batch updates the GP. During initial iterations, the proposed method is fast since the data size is small. The factorization approach

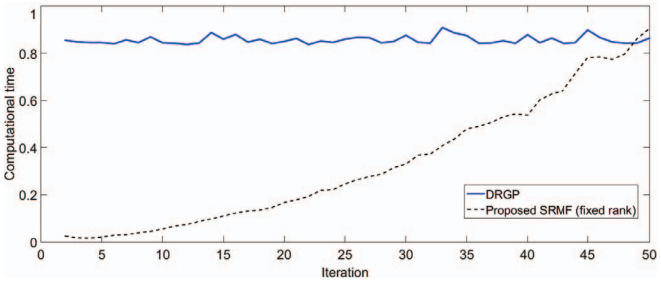

Fig. 1. Comparison of the computational time using the proposed approach and the recursive GP approach for the Sarcos Robotics dataset.

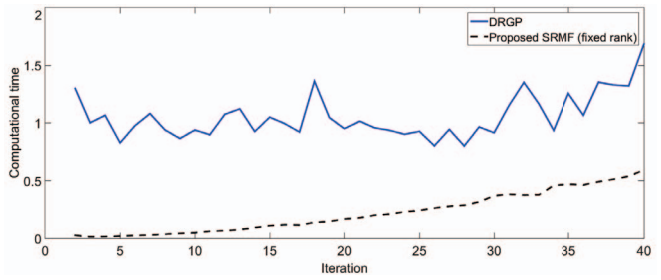

Fig. 2. Comparison of the computational time using the proposed approach and the recursive GP approach for the Abalone dataset.

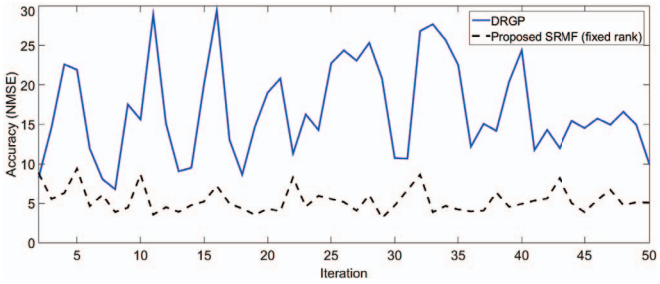

Fig. 3. Comparison of the prediction accuracy of the proposed approach and the recursive GP approach for the Sarcos Robotics dataset.

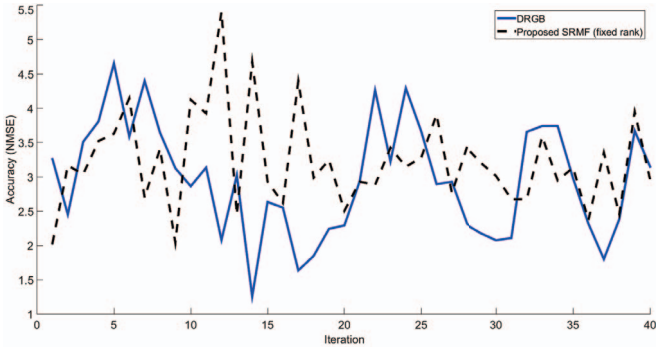

Fig. 4. Comparison of the prediction accuracy of the proposed approach and the recursive GP approach for the Abalone dataset.

results in faster computations than the deterministic approach employed in the DRGP algorithm. Although the deterministic approach keeps small fixed data size, it does not employ any step to factor the matrix for reduced sizes. Over increasing number of iterations, the proposed approach becomes slower since even the factored matrices become larger when data is accumulated. At some point our algorithm will get slower than the DRGP algorithm. For the Sarcos simulations, the two algorithms have similar speed by the final iterations (Figure 1). On the other hand, for the Abalone simulations, our algorithm was faster for all iterations (Figure 2).

Figures 3 and 4 show the accuracy of the DRGP and the proposed SRMF algorithms (measured by normalized mean square error (NMSE)) for the Sarcos and Abalone datasets, respectively. It is clear from the plots that the error is not increasing over the iterations even though more noise is added to the data over the iterations since the predictions (not the actual values) update the GP model at each iteration. For the Sarcos dataset, the proposed approach has a higher accuracy than the deterministic approach. For the Abalone dataset, accuracy of the proposed approach is comparable to that of the deterministic approach.

IV. Conclusion and Future Directions

This paper presented a sequential technique based on randomized low-rank matrix factorization approach for incrementally predicting values of an unknown function at test points using the Gaussian Processes framework. The use of the factored form of the kernel matrix serves as an efficient way to store increasing amounts of data points, as well as provides an advantage in computing the predicted values. The proposed method yields comparable accuracy compared to a state of the art recursive technique, and is computationally more efficient up to a certain data size. An immediate future direction is to relax the assumption that the kernel hyperparameters are fixed, and therefore perform a detailed comparison with the advanced technique from [6].

References

[1] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
[2] J. Quinonero-Candela and C. E. Rasmussen, "A Unified View of Sparse Approximate Gaussian Process Regression," *Journal of Machine Learning*, vol. 6, pp. 1939–1959, 2005.
[3] J. Hensman, N. Fusi, and N. D. Lawrence, " Gaussian Processes for Big Data," *Uncertainty in Artificial Intelligence*, 2013.
[4] A. Banerjee, D. B. Dunson, and S. K. Tokdar, "Efficient Gaussian Process Regression for Large Datasets," *Biometrika*, vol. 100, pp. 75–89, 2013.
[5] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. O'Neill, "Fast Direct Methods for Gaussian Processes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 252–265, 2016.
[6] M. Huber, "Recursive Gaussian Process: On-line Regression and Learning," *Pattern Recognition Letters*, vol. 45, pp. 85–91, August 2014.
[7] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions," *SIAM Review*, vol. 53 (2), 2011.
[8] C. E. Rasmussen, "http://www.gaussianprocess.org/gpml/data/.,"
[9] A. FRANK and A. ASUNCION, "Uci machine learning repository.,"