Fast large scale Gaussian process regression using approximate matrix-vector products

Vikas C. Raykar and Ramani Duraiswami
Department of Computer Science and Institute for advanced computer studies
University of Maryland, College Park, College Park, MD, USA

Abstract

Gaussian processes allow the treatment of non-linear non-parametric regression problems in a Bayesian framework. However the computational cost of training such a model with N examples scales as $\mathcal{O}(N^3)$. Iterative methods for the solution of linear systems can bring this cost down to $\mathcal{O}(N^2)$, which is still prohibitive for large data sets. We consider the use of ϵ-exact matrix-vector product algorithms to reduce the computational complexity to $\mathcal{O}(N)$. Using the theory of inexact Krylov subspace methods we show how to choose ϵ to guarantee the convergence of the iterative methods. We test our ideas using the improved fast Gauss transform. We demonstrate the speedup achieved on large data sets. For prediction of the mean the computational complexity is reduced from $\mathcal{O}(N)$ to $\mathcal{O}(1)$. Our experiments indicated that for low dimensional data ($d \leq 8$) the proposed method gives substantial speedups.

1 INTRODUCTION

The Gaussian process (GP) is a popular method for Bayesian non-linear nonparametric regression. Unfortunately its non-parametric nature causes computational problems for large data sets, due to an unfavorable $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ memory scaling for training. While the use of iterative methods, as first suggested by [10], can reduce the cost to $\mathcal{O}(kN^2)$ where k are the number of iterations, this is still too large. An important subfield of work in GP has attempted to bring this scaling down to $\mathcal{O}\left(m^2N\right)$ by making sparse approximations of size m to the full GP where $m \ll N$ [24, 18, 3, 9, 2, 21, 20, 19]. Most of these methods are based on using a representative subset of the training examples of size m. Different schemes specify different strategies to effectively choose the subset. A good review can be found in Ch. 8 of [13] or [12] for a more recent survey. A recent work [19] considers choosing m datapoints not constrained to be a subset of the data. While these methods often work quite well, there is no guarantee on the quality of the GP that results from the sparse approximation.

A GP is completely specified by its mean and covariance functions. Different forms of the covariance function gives us the flexibility to model different kinds of generative processes. One of the most popular covariance function used is the negative squared exponential (Gaussian). In this paper we explore an alternative class of methods that seek to achieve a speed-up for GP regression by computing an ϵ-exact approximation to the matrix-vector product used in the conjugate gradient method. Unlike methods which rely on choosing a subset of the dataset we use all the available points and still achieve $\mathcal{O}(N)$ complexity. There are at least three methods proposed to accelerate the matrix-vector product using approximation ideas: the dual-tree method [4], the fast Gauss transform (FGT) [6], and the improved fast Gauss transform (IFGT) [26], which have their own areas of applicability and performance characteristics. These methods claim to provide the matrix-vector product with a guaranteed accuracy ϵ, and achieve $\mathcal{O}(N \log N)$ or $\mathcal{O}(N)$ performance at fixed ϵ in both time and memory.

Novel contribution: An important question when using these methods is the influence of the approximate matrix-vector product on the convergence of the iterative method. Obviously these methods converge at machine precision. However, the accuracy necessary to guarantee convergence must be studied. Generally previous papers [26, 16] choose ϵ to a convenient small value such as 10^{-3} or 10^{-6} based on the application. We use a more theoretical approach and base our results on the theory of inexact Krylov subspace methods. We show that the *matrix-vector product may be* *performed in an increasingly inexact manner* as the iteration progresses and still allow convergence.

2 GAUSSIAN PROCESS MODEL

While Gaussian processes are covered well elsewhere (e.g. see [13]), both to establish notation and for completeness we provide a brief introduction here.

Model: The simplest most often used model for regression [23] is $y = f(x) + \varepsilon$, where $f(x)$ is a zero-mean Gaussian process with covariance function $K(x, x') : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and ε is independent zero-mean normally distributed noise with variance σ^2, i.e., $\mathcal{N}(0, \sigma^2)$. Therefore the observation process $y(x)$ is a zero-mean Gaussian process with covariance function $K(x, x') + \sigma^2\delta(x, x')$.

Inference: Given training data $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$ the $N \times N$ covariance matrix $\mathbf{K}$ is defined as $[\mathbf{K}]_{ij} = K(x_i, x_j)$. If we define the vector $\mathbf{y} = [y_1, \ldots, y_N]^T$ then $\mathbf{y}$ is a zero-mean multivariate Gaussian with covariance matrix $\mathbf{K} + \sigma^2\mathbf{I}$. Given the training data $\mathcal{D}$ and a new input x_* our task is to compute the posterior $p(f_*|x_*, \mathcal{D})$. Observing that the joint density $p(f_*, \mathbf{y})$ is a multivariate Gaussian, the posterior density $p(f_*|x_*, \mathcal{D})$ can be shown to be [13]

$$\begin{aligned} p(f_*|x_*, \mathcal{D}) \sim \mathcal{N}\big(&\mathbf{k}(x_*)^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}, \\ &K(x_*, x_*) - \mathbf{k}(x_*)^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}(x_*)\big), \end{aligned} \quad (1)$$

where $\mathbf{k}(x_*) = [K(x_*, x_1), \ldots, K(x_*, x_N)]^T$.

If we define $\xi = (\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$, then the mean prediction and the variance associated with it are

$$\mathbb{E}[f_*] = \mathbf{k}(x_*)^T\xi, \quad \text{and} \quad (2)$$

$$\mathbb{V}\text{ar}[f_*] = K(x_*, x_*) - \mathbf{k}(x_*)^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}(x_*). \quad (3)$$

Gaussian covariance: The covariance function has to be chosen to reflect the prior information about the problem. For high-dimensional problems, in the absence of any prior knowledge, the negative squared exponential (Gaussian) is the most widely used covariance function, and is the one that we use in this paper.

$$K(x, x') = \sigma_f^2 \exp\left(-\sum_{k=1}^{d} \frac{(x_k - x_k')^2}{h_k^2}\right). \quad (4)$$

The $d + 2$ parameters ($[h_1, \ldots, h_d, \sigma_f, \sigma]$) are referred to as the hyperparameters.

3 CONJUGATE GRADIENTS

Training: Given the hyperparameters, the *training phase* consists of the evaluation of the vector

$$\xi = (\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \quad (5)$$

which needs the inversion of an $N \times N$ matrix $\mathbf{K} + \sigma^2\mathbf{I}$. Direct computation of the inverse of the symmetric matrix (using Cholesky decomposition) requires $\mathcal{O}(N^3)$ operations and $\mathcal{O}(N^2)$ storage, which is impractical even for problems of moderate size (typically a few thousands).

Conjugate gradient method: For larger systems it is more efficient to solve the system

$$\widetilde{\mathbf{K}}\xi = \mathbf{y} \quad \text{where,} \quad \widetilde{\mathbf{K}} = \mathbf{K} + \sigma^2\mathbf{I} \quad (6)$$

using iterative methods, provided the method converges quickly. Modern iterative Krylov subspace methods show good convergence properties, especially when preconditioned [8]. Since $\widetilde{\mathbf{K}}$ is *symmetric and positive definite* we can use the well known *conjugate-gradient* (CG) method [7] to iteratively solve Eq. (6) A good exposition of this method can be found in Ch. 2 of [8]. The idea of using conjugate gradient for GP was first suggested by [10].

Convergence of CG: The iterative method generates a sequence of approximate solutions ξ_k at each step, which converge to the true solution ξ. One of the sharpest known convergence results for the iterates is given by

$$\frac{\|\xi - \xi_k\|_{\widetilde{\mathbf{K}}}}{\|\xi - \xi_0\|_{\widetilde{\mathbf{K}}}} \leq 2\left[\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right]^{2k}, \quad \|w\|_{\widetilde{\mathbf{K}}} = w^T\widetilde{\mathbf{K}}w \quad (7)$$

where the $\widetilde{\mathbf{K}}$-*norm* of any vector w is defined as above [8]. The constant $\kappa = \lambda_{max}/\lambda_{min}$, the ratio of the largest to the smallest eigenvalues is called the *spectral condition number* of the matrix $\widetilde{\mathbf{K}}$. Since $\kappa \in (1, \infty)$, Equation 7 implies that if κ is close to one, the iterates will converge very quickly.

Given a tolerance $0 < \eta < 1$ a practical CG scheme iterates till it computes a vector ξ_k such that the ratio of the current residual $\|\mathbf{y} - \widetilde{\mathbf{K}}\xi_k\|_2$ to the initial residual is below the tolerance.

$$\frac{\|\mathbf{y} - \widetilde{\mathbf{K}}\xi_k\|_2}{\|\mathbf{y} - \widetilde{\mathbf{K}}\xi_0\|_2} \leq \eta. \quad (8)$$

Most implementations start the iteration at $\xi_0 = \mathbf{0}$, though a better guess can be used if available. The relative residual in the Euclidean norm is related to the relative error in the $\widetilde{\mathbf{K}}$-norm as [8]

$$\frac{\|\mathbf{y} - \widetilde{\mathbf{K}}\xi_k\|_2}{\|\mathbf{y} - \widetilde{\mathbf{K}}\xi_0\|_2} \leq \sqrt{\kappa}\frac{\|\xi - \xi_k\|_{\widetilde{\mathbf{K}}}}{\|\xi - \xi_0\|_{\widetilde{\mathbf{K}}}} \leq 2\sqrt{\kappa}\left[\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right]^{2k}. \quad (9)$$

This implies that for a given η the number of iterations required is

$$k \geq \ln\left[\frac{2\sqrt{\kappa}}{\eta}\right] / 2\ln\left[\frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} - 1}\right]. \quad (10)$$

Sometimes the estimate (10) can be very pessimistic. Even if the condition number is large, the convergence is fast if the eigenvalues are clustered in a few small intervals [8]. In the examples we consider later convergence was achieved relatively quickly. If convergence is slow we must consider preconditioning, which is a topic outside the scope of the present paper.

Computational complexity: The actual implementation of the CG method requires *one* $\mathcal{O}(dN^2)$ *matrix-vector multiplication* and $5N$ flops per iteration. Four vectors of length N are required for storage. The storage is $\mathcal{O}(N)$ since the matrix-vector multiplication can use elements computed on the fly and not storing the entire matrix. Empirically the number of iterations required is generally small compared to N leading to a computational cost of $\mathcal{O}(kdN^2)$. It should be noted that the $\mathcal{O}(N)$ space comes at a time trade-off. If the matrix is cached (*i.e.* $\mathcal{O}(N^2)$ memory) then the computational cost is $\mathcal{O}(dN^2 + kN^2)$.

4 FAST MATRIX-VECTOR PRODUCTS

The quadratic complexity is still too high for large datasets. The core computational step in each CG iteration involves the multiplication of the matrix $\mathbf{K}$ with some vector, say $\mathbf{q}$. The j^{th} element of the matrix-vector product $\mathbf{Kq}$ can be written as $(\mathbf{Kq})_j = \sum_{i=1}^{N} q_i k(x_i, x_j)$.

In general for each *target point* $\{t_j \in \mathbb{R}^d\}_{j=1}^M$ (which in our case are the same as the *source points* x_i) this can be written as

$$G(t_j) = \sum_{i=1}^{N} q_i k(x_i, t_j). \tag{11}$$

The computational complexity to evaluate (11) at M target points is $\mathcal{O}(MN)$. For the Gaussian kernel various approximation algorithms have been proposed to compute the above sum in $\mathcal{O}(M + N)$ time. These algorithms compute the sum to any arbitrary ϵ precision. Broadly there are two kinds of methods–the series based methods and data structure based methods.

Series based methods: These methods are inspired by the fast multipole methods (FMM) which were originally developed for the fast summation of the potential fields generated by a large number of sources, such as those arising in gravitational potential problems [5]. The fast Gauss transform (FGT) is a special case where FMM ideas were used for the Gaussian potential [6]. The improved fast Gauss transform (IFGT) is a similar algorithm based on a single different factorization and data structures. It is suitable for higher dimensional problems and provides comparable performance in lower dimensions [26].

Data structure based methods: Another class of methods proposed are the dual-tree methods [4]. These methods rely on space partitioning trees like kd-trees and ball trees and not on series expansions.

5 THE ACCURACY ϵ, NECESSARY

Obviously the accuracy, ϵ, that minimizes work while achieving the best performance must be chosen. However, determining this quantity in a principled way is often difficult. Most previous methods choose ϵ to a convenient small value such as 10^{-3} or 10^{-6} based on the application and *a posteriori* analysis. Indeed, one can in principle adaptively vary ϵ as the iteration proceeds. We were however able to use some recent results from linear algebra [17] and analyze the effect of the choice of ϵ on the CG method.

Krylov subspace method: The conjugate gradient method is a *Krylov subspace method* adapted for a symmetric positive definite matrix. Krylov subspace methods at the k^{th} iteration compute an approximation to the solution of any linear system $Ax = b$ by minimizing some measure of error over the affine space $x_0 + \mathcal{K}_k$, where x_0 is the initial iterate and the k^{th} Krylov subspace is $\mathcal{K}_k = \text{span}(r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0)$. The residual at the k^{th} iterate is $r_k = b - Ax_k$.

Inexact Krylov subspace method: A general framework for understanding the effect of approximate matrix-vector products on Krylov subspace methods for the solution of symmetric and nonsymmetric linear systems of equations is given in [17]. The paper considers the case where at the k^{th} iteration instead of the exact matrix-vector multiplication Av_k, the product

$$\mathcal{A}v_k = (A + E_k)v_k \tag{12}$$

is computed, where E_k is an error matrix which may change as the iteration proceeds. A nice result in the paper shows how large $\|E_k\|$ can be at each step while still achieving convergence with the desired tolerance. Let $r_k = \|Ax_k - b\|$ be the residual at the end of the k^{th} iteration. Let $\tilde{r}_k$ be the corresponding residual when an approximate matrix-vector product is used. If at every iteration

$$\|E_k\| \leq l_m \frac{1}{\|\tilde{r}_{k-1}\|} \delta, \tag{13}$$

then at the end of k iterations $\|\tilde{r}_k - r_k\| \leq \delta$ [17]. The term l_m in general is unavailable since it depends on knowing the spectrum of the matrix. However our empirical results and also some experiments in [17] suggest that $l_m = 1$ seems to be a reasonable value. This shows that the matrix-vector product may be performed *in an increasingly inexact manner as the iteration progresses and still allow convergence to the solution.*

ϵ-exact approximation: We will use the following notion of ϵ-exact approximation as used previously in [6, 26, 14]. Given any $\epsilon > 0$, $\hat{G}(t_j)$ is an ϵ-exact approximation to $G(t_j)$ if the maximum absolute error relative to the total weight $Q = \sum_{i=1}^{N} |q_i|$ is upper bounded by ϵ, i.e.,

$$\max_{t_j} \left[\frac{|\hat{G}(t_j) - G(t_j)|}{Q} \right] \leq \epsilon. \tag{14}$$

For our problem because of the ϵ-exact approximation criterion (Equation 14) every element in the approximation to the vector $\mathbf{Kq}$ is within $\pm Q\epsilon_k$ of the true value, where $Q = \sum_{i=1}^{N} |q_i|$ and ϵ_k is the error in the matrix vector product at the k^{th} iteration. Hence the error matrix E_k is of the form

$$E_k = \epsilon_k \begin{pmatrix} \pm e_{11} & \dots & \pm e_{1N} \\ \vdots & \ddots & \vdots \\ \pm e_{N1} & \dots & \pm e_{NN} \end{pmatrix}, \tag{15}$$

where $e_{ij} = sign(q_j) \in (+1, -1)$. It should be noted that this matrix is an upper bound rather than the actual error matrix. It can be seen that $\|E_k\| = N\epsilon_k$. Hence Equation 13 suggests the following strategy to choose ϵ_k.

$$\epsilon_k \leq \frac{\delta}{N} \frac{\|\mathbf{y} - \widetilde{\mathbf{K}}\xi_0\|}{\|\tilde{r}_{k-1}\|}. \tag{16}$$

This guarantees that

$$\frac{\|\mathbf{y} - \widetilde{\mathbf{K}}\xi_k\|_2}{\|\mathbf{y} - \widetilde{\mathbf{K}}\xi_0\|_2} \leq \eta + \delta. \tag{17}$$

Figure 1 shows the ϵ_k selected at each iteration for a sample regression problem. As the iteration progresses the ϵ_k required increases.

6 PREDICTION

Once ξ is computed in the training phase, the mean prediction for any new x_* is given by $\mathbb{E}[f_*] = \mathbf{k}(x_*)^T\xi = \sum_{i=1}^{N} \xi_i k(x_i, x_*)$. Predicting at M points is again a matrix vector multiplication operation. Direct computation of $\mathbb{E}[f_*]$ at M test points due to the N training examples is $\mathcal{O}(NM)$. Using the fast matrix-vector product reduces the computational cost to $\mathcal{O}(N + M)$.

The variance for each prediction is given by $\mathbb{V}\text{ar}[f_*] = K(x_*, x_*) - \mathbf{k}(x_*)^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}(x_*)$. First we need to

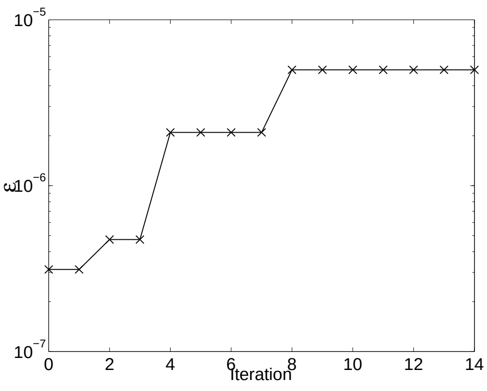

Figure 1: The error for the IFGT ϵ_k selected at each iteration for a sample 1D regression problem. The error tolerance for the CG was set to $\eta = 10^{-3}$ and $\delta = 10^{-3}$.

solve a linear system with $\mathbf{K} + \sigma^2\mathbf{I}$ during the training phase via some suitable decomposition. Once the decomposition is computed for each x the computation of uncertainty is $\mathcal{O}(N^2)$. For M points it is $\mathcal{O}(MN^2)$. Using the conjugate gradient method and the IFGT we can compute $\widetilde{\mathbf{K}}^{-1}\mathbf{k}(x_*)$ in $\mathcal{O}(kN)$ time. For M points we need $\mathcal{O}(kMN)$ time.

Table 1 compares the computational and space complexities for different stages of Gaussian process regression using different methods.

7 EXPERIMENTS

Datasets: We use the following two datasets – ***robotarm*** [1][dataset size $N = 10,000$, dataset dimension $d = 2$] and ***abalone*** [2][$N = 4,177$, $d = 7$]. The datasets are chosen to be representative of low and medium dimensions respectively. These are also known to be highly non-linear regression problems and widely used to benchmark regression algorithms.

Evaluation Procedure: For each dataset 90% of the examples were used for training and the remaining 10% were used for testing. The results are shown for a ten-fold cross validation experiment. The inputs are linearly re-scaled to have zero mean and unit variance on the training set. The outputs are centered to have zero mean on the training set. The mean

[1] A synthetic 2-d nonlinear robot arm mapping problem [11]. The data is generated according to $f(x_1, x_2) = 2.0\cos(x_1) + 1.3\cos(x_1 + x_2)$. The value of x_1 is chosen randomly in $[-1.932, -0.453]$ and x_2 is chosen randomly in $[0.534, 3.142]$ as in [13]. The target values are obtained by adding Gaussian noise of variance 0.1 to $f(x_1, x_2)$.

[2] The task is to predict the age of abalone (number of rings) from physical measurements. Downloaded from `http://www.liacc.up.pt/~ltorgo/Regression/DataSets.html`

Table 1: *The dominant computational and space complexities.* We have N training points and M test points in d dimensions. k is the number of iterations required by the conjugate gradient procedure to converge to a specified tolerance. The memory requirements are for the case where the Gram matrix is constructed on the fly at each iteration. For the fast MVM procedure, the constant $\mathcal{D}$ grows with d depending on the type of fast MVM used.

	Direct Method		*Conjugate gradient*		*Conjugate gradient+Fast MVM*	
	Time	Space	Time	Space	Time	Space
Training	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$	$\mathcal{O}(kdN^2)$	$\mathcal{O}(dN)$	$\mathcal{O}(k\mathcal{D}N)$	$\mathcal{O}(dN+\mathcal{D})$
Prediction	$\mathcal{O}(dMN)$	$\mathcal{O}(dM+dN)$			$\mathcal{O}(\mathcal{D}M+\mathcal{D}N)$	$\mathcal{O}(dM+dN+\mathcal{D})$
Uncertainty	$\mathcal{O}(MN^2)$		$\mathcal{O}(kdMN^2)$	$\mathcal{O}(dM+dN)$	$\mathcal{O}(k\mathcal{D}MN)$	$\mathcal{O}(dM+dN+\mathcal{D})$

squared error (MSE) is defined as the squared error between the mean prediction and the actual value averaged over the test set. Since the MSE is sensitive to the overall scale of the target values we normalize it by the variance of the targets of the test cases to obtain the *standardized mean squared error* (SMSE) [13]. This causes the trivial method of guessing the mean of the training targets to have a SMSE of approximately 1. For all the experiments we used the squared exponential covariance function (Equation 4). The $d+2$ hyperparameters ($[h_1, \ldots, h_d, \sigma_f, \sigma]$) were selected by optimizing the marginal likelihood on the subset using the direct method (automatic relevance determination (ARD) [23]) [3]. For all methods the same hyperparameters were used. For larger subsets where we cannot find the hyperparameters directly we use the one computed from the largest possible subset.

Fast Matrix-Vector multiplication algorithms: In order to compute the matrix-vector product we experimented with two algorithms–(1) the IFGT [4] [26] and (2) the kd-tree based dual-tree algorithm [5] [4]. However it should be noted that the results regarding the choice of ϵ hold for any fast algorithm.

With regard to the bandwidth–the general trend is that IFGT shows better speedups for large bandwidths while the dual-tree algorithm performs better at small bandwidths. In our experiments–for the hyperparameters chosen by the ARD procedure–we found that the IFGT gave better speedups than the dual-tree methods.

The IFGT requires the choice of two parameters (truncation number p and the number of clusters K) such that the actual error will be less than the desired. The IFGT originally proposed [26] did not specify how to choose these parameters and the error bound specified there was also not tight. However, recent work by the developers [14] has given an optimal way to choose the parameters and is part of the code available online. Another issue was that in the IFGT code the bandwidth h_k has to be the same for all dimensions. However this was easily addressed by dividing each co-ordinate with the corresponding bandwidth h_k and then using the IFGT with a bandwidth $h = 1$.

[3]Code downloaded from `http://www.gaussianprocess.org/gpml/code/matlab/doc/`.

[4]The IFGT code was downloaded from `http://www.cs.umd.edu/~vikas/code/IFGT/IFGT_code.htm`

[5]The dual-tree code was downloaded from `http://www.cs.ubc.ca/~awll/nbody_methods.html`

7.1 RESULTS

Figure 2 shows the total training time, the SMSE, and the total prediction time for the two datasets as a function of the number of datapoints. For each fold a subset of the training data of size m was selected at random. The process was repeated 10 times. m was progressively increased to get a learning curve. All the experiments were run on a 1.83 GHz processor with 1GB of RAM. We show the scaling behavior for the following four methods.

1. ***Subset of datapoints*** (SD) This is simply the direct implementation with a subset of the training data. The subset is chosen randomly. The training and prediction time scale as $\mathcal{O}(m^3)$ and $\mathcal{O}(mM)$ respectively. M is the total number of test points.

2. ***Subset of regressors/projected process*** (SR and PP) (See Chapter 8 in [13] for a description of these methods.) The training and prediction time scale as $\mathcal{O}(m^2N)$ and $\mathcal{O}(mM)$ respectively. N is the total number of training points. The SR and PP methods have the same predictive mean. The recent paper [19] also has the same computational complexity [6].

3. ***Proposed method with IFGT*** (CG+IFGT) The training and prediction time scale as $\mathcal{O}(km)$ and $\mathcal{O}(m+M)$ respectively. The tolerance for the

[6]Also it is expected to be much more expensive because of the optimization procedure to find the location of the pseudo-inputs.

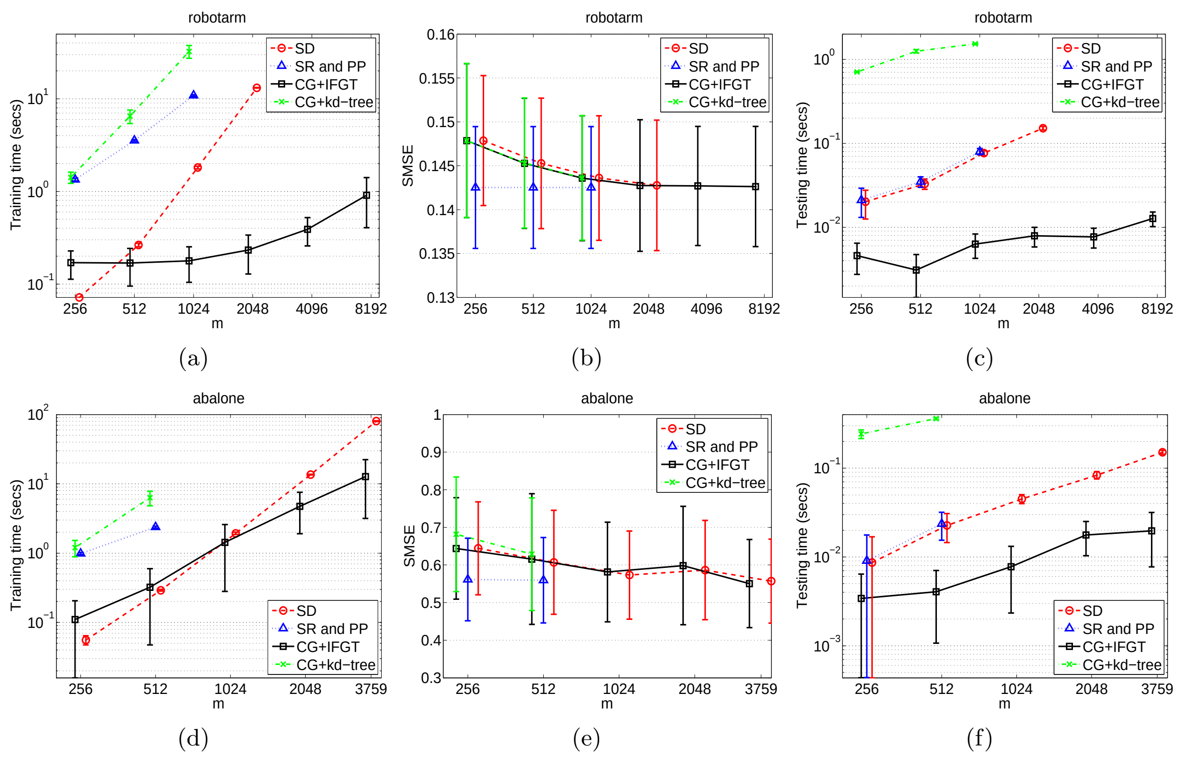

Figure 2: (a) The total training time, (b) the SMSE, and (c) the testing time as a function of m for the robotarm dataset. The errorbars show one standard deviation over a 10-fold experiment. The results are slightly displaced w.r.t. the horizontal axis for clarity. The lower panel show the same for the abalone dataset.

conjugate gradient procedure η was set to 10^{-3} and the δ in Equation 17 for IFGT was set to 10^{-3}. The accuracy for testing using IFGT was set to $\epsilon = 10^{-6}$.

4. ***Proposed method with kd-tree***(CG+kd-tree) Same as above but using kd-tree instead of the IFGT.

The following observations can be made:

- From Figure 2(a) it can be seen that as m increases the training time for the proposed method increases linearly in contrast to the quadratic increase for the SD method. The SR and PP methods have small training times only for small m.
- As m increases the general trend for all methods is that SMSE decreases (see Figure 2(b)).
- It is not surprising that SR and PP show the least SMSE. This is because SR and PP use all the datapoints while retaining m of them as the active set. However the proposed method can still catch up with the SMSE of SR and PP and still have a significantly lower running time.
- Regarding the testing time the proposed method shows significant speedups.
- As the dimension of the problem increases the cutoff point, i.e., N at which the proposed fast method is better than the direct method increases.
- At the hyperparameters chosen, the dual-tree algorithms ended up taking larger time than the direct method probably because of the time taken to build up the kd-trees.

For large dimensional data the fast algorithms like IFGT and dual-tree methods do not scale well. We were unable to get good speedups for high dimensional datasets like SARCOS [7] (a 21 dimensional robot arm dataset) using the IFGT. However either the subset of data or PP/SR methods can be used with a higher dimensional data set such as SARCOS.

8 IMPLICIT SURFACE FITTING

Recently implicit models for surface representation are gaining popularity [1, 15, 22]. An implicit representation describes the surface S as the set of all points where a certain smooth function, $f : \mathbb{R}^d \to \mathbb{R}$ vanishes, *i.e.*, $\mathrm{S} = f^{-1}(0) = \{x \in \mathbb{R}^d \text{ such that } f(x) = 0\}$. Once

[7] `http://www.gaussianprocess.org/gpml/data/`

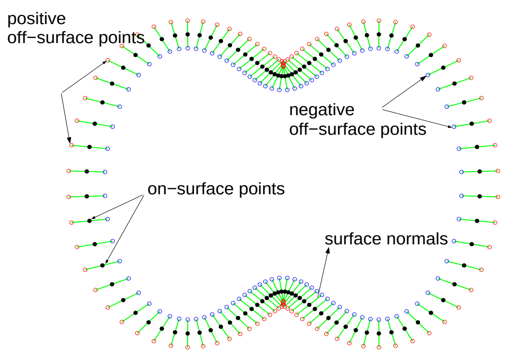

Figure 3: Appending the positive and the negative off-surface points along the surface normals at each point.

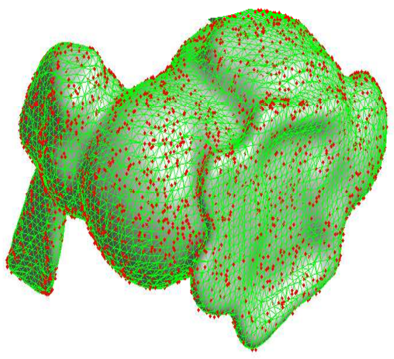

Figure 4: The isosurface extracted using the function learnt by Gaussian Process regression. The point cloud data is also shown. It took 6 minutes to learn this implicit surface form $10,000$ surface points.

we have a representation f it can be evaluated on a grid in $\mathbb{R}^d$ and an explicit representation can be formed as a mesh of polygons for visualization purposes–often referred to as isosurface extraction.

Given a set of N points $\{x_i \in \mathbb{R}^d\}_{i=1}^N$ ($d = 3$ for surface fitting) lying on a smooth manifold S we have to find a function $f : \mathbb{R}^d \to \mathbb{R}$ such that $f(x_i) = 0$, for $i = 1, \ldots, N$, and it smoothly interpolates for any other $x \in \mathbb{R}^d$. In order to avoid the trivial solution $f(x) = 0$ we need to add additional constraints, *i.e.*, points where the function f is not zero. Such additional points are referred to as the *off-surface* points. So the formulation now is to find a function f such that $f(x_i) = 0$, for $i = 1, \ldots, N$ and $f(x_i) = d_i \neq 0$, for $i = N + 1, \ldots$. For generating the off-surface points we use the following scheme [1]. We append each data point x_i with two off-surface points x_i^+ and x_i^-, one on each side of the surface as shown in Figure 3. The off-surface points are generated by projecting along the surface normals at each point. It should be ensured that the surface normals are consistently oriented. We use the signed-distance function for the off-surface points. The value of the function is chosen to be the distance to the closest on-surface point. Points outside the surface are assigned a positive value while those inside the surface are given a negative value. We ensure that the off-surface points do not intersect the underlying surface using the normal length validation scheme described in [1].

We have used Gaussian process regression to learn this function from the point cloud data. One of the major bottlenecks for most implicit surface methods is their prohibitive computational complexity, and this applies to Gaussian process regression as well, the computational complexity of which scales as $\mathcal{O}(N^3)$. Using the propose method we were able to handle large data-sets. Since surface fitting in done in $d = 3$ IFGT gave good speedups. Figure 4 shows the fitted surface for the *bunny* data. It took 6 minutes to fit the model using 10,000 surface points. Note that the actual number of points used for Gaussian process regression is 30,000 due to the off-surface points. We were unable to run the direct method on such large datasets. Unlike regression for surface fitting we would like to use all the available data to get accurate surface reconstruction. The IFGT was also used for isosurface extraction.

9 DISCUSSION/FURTHER ISSUES

We have demonstrated that the approximate fast matrix vector products achieved by ϵ-exact methods such as the improved fast Gauss transform can achieve a fast solution of the Gaussian process regression. The following are the contributions of this paper:

(1) We show that the training time for GP regression is reduced to linear $\mathcal{O}(N)$ by using the conjugate-gradient method coupled with the IFGT. The prediction time per test input is reduced to $\mathcal{O}(1)$.

(2) Using results from the theory of inexact Krylov subspace methods we show that the matrix-vector product may be performed in an increasingly inexact manner as the iteration progresses and still allow convergence to the correct solution.

(3) Our experiments indicated that for low dimensional data ($d \leq 8$) the proposed method gives substantia speedups.

The idea of speeding up matrix-vector multiplication for Gaussian process regression was first explored in [16]–who use *kd*-trees to speed up the matrix-vector multiplication. The main contribution of this paper is a strategy to choose ϵ while using such methods.

While the scope of this paper is to speed up the original GPR it should be noted that methods which use a subset of the data [24, 18, 3, 9, 2, 21, 20] can also be further speeded up using these algorithms. This is because even these methods require matrix-vector products to be taken with a smaller subset of the data.

One drawback of the IFGT is that it is specific to the Gaussian kernel. For other covariance functions, like the Matern class of kernels–fast algorithms can be developed. The results presented in this paper, regarding the choice of ϵ should hold independent of the covariance function used.

It would also be interesting to explore whether the techniques presented here can be used to speedup classification [25] using a Gaussian process model.

References

[1] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *ACM SIGGRAPH 2001*, pages 67–76, Los Angeles, CA, 2001.

[2] L. Csato and M. Opper. Sparse on-line gaussian processes. *Neural Computation*, 14(3):641–668, 2002.

[3] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243264, December 2001.

[4] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *SIAM International conference on Data Mining*, 2003.

[5] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. of Comp. Physics*, 73(2):325–348, 1987.

[6] L. Greengard and J. Strain. The fast Gauss transform. *SIAM Journal of Scientific and Statistical Computing*, 12(1):79–94, 1991.

[7] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[8] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations.* SIAM, 1995.

[9] N. Lawrence, M. Seeger, and R. Herbrich. *Advances in Neural Information Processing Systems 15*, chapter Fast Sparse Gaussian Process methods: The Informative Vector Machine, pages 625–632. MIT Press, 2003.

[10] D. MacKay and M. N. Gibbs. Efficient implementation of Gaussian processes. unpublished, 1997.

[11] D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4:448–472, 1992.

[12] J. Quiñonero Candela and C. E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1935–1959, 2005.

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

[14] V. C. Raykar, C. Yang, R. Duraiswami, and N. Gumerov. Fast computation of sums of Gaussians in high dimensions. Technical Report CS-TR-4767, Department of Computer Science, University of Maryland, CollegePark, 2005.

[15] B. Schölkopf, J. Giesen, and S. Spalinger. Kernel methods for implicit surface modeling. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1193–1200. MIT Press, Cambridge, MA, 2005.

[16] Y. Shen, A. Ng, and M. Seeger. Fast Gaussian process regression using KD-trees. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18.* MIT Press, Cambridge, MA, 2006.

[17] V. Simoncini and D. B. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM J. Sci. Comput.*, 25(2):454–477, 2004.

[18] A. Smola and B. Bartlett. Sparse greedy gaussian process regression. In *Advances in Neural Information Processing Systems*, page 619625. MIT Press, 2001.

[19] E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18.* MIT Press, Cambridge, MA, 2006.

[20] M. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1:211–244, 2001.

[21] V. Tresp. A bayesian committee machine. *Neural Computation*, 12(11):2719–2741, 2000.

[22] C. Walder, O. Chapelle, and B. Schölkopf. Implicit surface modelling as an eigenvalue problem. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 937 – 944, 2005.

[23] C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems*, volume 8, 1996.

[24] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, page 682688. MIT Press, 2001.

[25] C. K. I. Willimas and D. Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.

[26] C. Yang, R. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast Gauss transform. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1561–1568. MIT Press, Cambridge, MA, 2005.