Gaussian Processes and Fast Matrix-Vector Multiplies

Iain Murray MURRAY@CS.TORONTO.EDU
Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3G4 CANADA

Abstract

Gaussian processes (GPs) provide a flexible framework for probabilistic regression. The necessary computations involve standard matrix operations. There have been several attempts to accelerate these operations based on fast kernel matrix-vector multiplications. By focussing on the simplest GP computation, corresponding to test-time predictions in kernel ridge regression, we conclude that simple approximations based on clusterings in a kd-tree can never work well for simple regression problems. Analytical expansions can provide speedups, but current implementations are limited to the squared-exponential kernel and low-dimensional problems. We discuss future directions.

1. Introduction

One attraction of Gaussian process based regression is that inference requires only simple, standard matrix operations. Straightforward implementations scale poorly: $\mathcal{O}(n^2)$ in memory and $\mathcal{O}(n^3)$ in time, which has lead to interest in improving the applicability of Gaussian processes using fast numerical methods. We have had difficulty obtaining worthwhile speedups by using the methods proposed in the literature. This abstract outlines some of the difficulties particular to GPs and considers future directions.

2. Gaussian Process setup

Rasmussen and Williams (2006) provides a full review of Gaussian processes for machine learning. Inferences are based on n training pairs given by inputs $X = \{\mathbf{x}_1, \dots \mathbf{x}_n\}$ and corresponding scalar outputs stored in a vector $\mathbf{y}$ of length n. We assume that each observation y_i corresponds to a noisy observation, $y_i \sim \mathcal{N}(f_i, \sigma_n^2)$, of an underlying function value $f_i = f(\mathbf{x}_i)$. The prior distribution over latent function values at any set of input points X is a multivariate Gaussian distribution: $\mathbf{f} \sim \mathcal{N}(0, K)$. Each element K_{ij} of the $n \times n$ covariance matrix is given by $k(\mathbf{x}_i, \mathbf{x}_j)$, a positive semi-definite kernel function evaluated at a pair of input locations.

The most common, although not always the most appropriate, kernel for D-dimensional vectorial inputs is the squared-exponential or "Gaussian":

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\bigg(-\frac{1}{2} \sum_{d=1}^{D} (x_{d,i} - x_{d,j})^2 / \ell_d^2 \bigg). \quad (1)$$

Here σ_f^2 is the 'signal variance' controlling the overall scale of the function, and the ℓ_d give the characteristic lengthscale of each dimension. The lengthscales are often set equal using a fixed 'bandwidth' $\ell_d = h/\sqrt{2}$.

3. Fast Matrix-Vector Multiplication

The mean predictor for a set of test inputs is $\bar{\mathbf{f}}_* = K_*^\top \boldsymbol{\alpha}$, where K_* is an $n \times n_*$ matrix obtained by evaluating the kernel function between the n training points and the n_* test inputs, and the weights $\boldsymbol{\alpha}$ are the result of solving a linear system at training time. The test-time cost of the matrix-vector multiplication (MVM) for prediction is $\mathcal{O}(n_* n)$. Algorithms providing faster MVM operations can be directly applied to making rapid mean predictions at test time. If this simplest task can be accelerated there is hope for applying fast MVM operations to other GP computations such as finding $\boldsymbol{\alpha}$ (section 4).

3.1. Sparsity of the kernel matrix

Matrix multiplications and other matrix computations can be performed more cheaply when a matrix is *sparse*, i.e. it contains many zero entries. It may also be possible to exploit an *approximately sparse* matrix containing many near-zero entries.

In local kernel regression training points far from a test location can be ignored because the lengthscale

Submitted to the workshop on numerical mathematics at the 26th International Conference on Machine Learning, Montreal, Canada, 2009.

(or bandwidth) of the kernel becomes narrow for large datasets. This has previously been exploited to create a fast regression method (Moore et al., 1997).

In Gaussian process regression, however, the width of the kernel is often comparable to the range of the input points in the training set, as the underlying function is often a simple trend with respect to any single input. To back up this assertion, we trained GPs on 25 standard regression problems collated by Luís Torgo (`http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html`). Using both maximum likelihood and cross validation, the best lengthscales for most datasets is similar to the width of the data set. This means that the covariance matrix is often not sparse, even when using a covariance function with compact support (e.g. Rasmussen and Williams 2006, § 4.2). A common exception is time-series datasets, where often only a short window of time is useful for prediction.

In high-dimensional input spaces it is possible to move by a lengthscale in each of several dimensions, giving very small covariances between some pairs of points. The training set covariance matrix for a real-world moderate-dimensional dataset SARCOS (`http://www.gaussianprocess.org/gpml/data/`) does have many near-zero values in it.

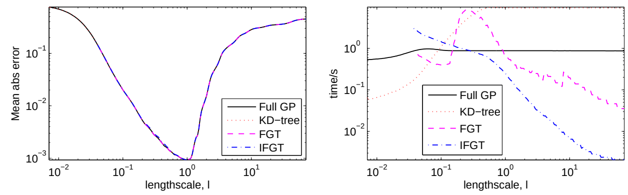
3.2. Multi-resolution data structures

There is a belief that multi-resolution space-partitioning data structures enable fast use of huge datasets in many statistical methods, including Gaussian Process regression (Gray & Moore, 2001). Shen et al. (2006) suggested adapting the fast locally-weighted polynomial regression algorithm of Moore et al. (1997) to GP regression. This method builds a kd-tree of the training data, and assumes that groups of kernel values are approximately equal. The use of more advanced algorithms and data structures have also been proposed (Gray, 2004; Freitas et al., 2006), but the key underlying assumption is the same.

When making a test prediction these algorithms dynamically partition the training set into subsets S_c. The subsets are chosen so that the kernel between each member and the test location is close to some constant k_c. The mean prediction can then be approximated as follows:

$$\bar{f}(x_*) = \sum_i \alpha_i \, k(x_i, x_*) \;\approx\; \sum_c \left[\textstyle\sum_{i \in S_c} \alpha_i\right] k_c \,. \qquad (2)$$

Using cached statistics it is possible to compute the sum over the α_i weights for groups of nearby training points efficiently. It is also possible to bound the error



(a) (b)

Figure 1. **(a)** Average error over 4096 test data points, against length-scale. The training data was 4096 points taken from the same draw from a 2-dimensional GP with true lengthscale $\ell = 1$, and $\sigma_n = 10^{-3}$. Each method had both absolute and relative errors set to $\epsilon = 10^{-3}$, except the IFGT, which had its relative error tolerance set to 10^{-6}. **(b)** Time against length-scale for the predictions in (a).

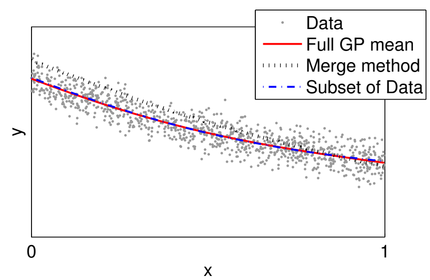


Figure 2. A synthetic 1D data set. The GP mean predictors using all the data and a subset of every other point are very close. The merging method described in the text gives much worse results. In this example, the IFGT can produce a mean predictor indistinguishable from that of the full GP in over an order of magnitude less time.

from this procedure and make the subsets sufficiently small to guarantee any given error criterion. However, there is no guarantee that the algorithm will actually provide a speed-up.

In one experiment we generated a 2-dimensional synthetic dataset drawn from a GP prior using a squared-exponential kernel. The inputs were drawn uniformly within a unit square, and the lengthscale was set to $\ell = 1$. Making accurate predictions depends on using a length-scale close to the correct value (figure 1(a)). Figure 1(b) shows that the time taken for a test MVM operation using several methods. The 'Full GP' simply used MATLAB's multiply operation. The kd-tree results used the library provided by Lang et al. (2006). Unfortunately, the kd-tree method saturated at its worst running time at the optimal lengthscale. Performing the MVM directly would have been considerably faster. This is representative of our experience with several datasets.

There are many free choices in implementing a fast method, both in how the data-structures are built from the data and in the recursions that are run on them to compute the MVM. Lang et al.'s code is primarily provided as a demonstration of techniques for researchers, and is not as heavily optimized as it could be. Our own code also failed to provide speedups, although results using any particular implementation cannot prove that speedups are impossible.

To examine whether speedups are possible for *any* implementation we simplify the situation further. We focus only on the core approximation: replacing similar kernel values with a single estimate. We test this idea on a well-behaved 1D regression problem, figure 2. None of the kernel values are close to zero here, so we must merge large but numerically close values. We consider combining pairs of adjacent training points:

$$\alpha_i\, k(\mathbf{x}_*, \mathbf{x}_i) \;+\; \alpha_{i+1}\, k(\mathbf{x}_*, \mathbf{x}_{i+1}) \approx (\alpha_i + \alpha_{i+1})\, k(\mathbf{x}_*, (\mathbf{x}_i + \mathbf{x}_{i+1})/2). \qquad (3)$$

This won't be an optimal merging, but these points are all *very* close and should be reasonable merging candidates. Besides, we would like to merge many more points: the speedup here will be less than two-fold and more dramatic speedups would be required to make a big impact on the applicability of GP methods.

It is surprising and very disappointing that merging pairs gives a significantly worse mean predictor. It is worth comparing to a simpler approach: if we had simply thrown away half the data from the start, the resulting mean predictor is hard to distinguish from that found using all the data. In this case, attempting to use more data by introducing an approximation is worse than simply ignoring the extra data.

The algorithms that have been proposed adaptively merge points. They are able to merge two adjacent weights when used for a far-away test point, but not for nearby test points. This would be useful when some kernel values are close to zero, which is not the case in our one-dimensional example. However, a dataset with a short lengthscale, such as a time series, may benefit. High-dimensional problems with approximately sparse covariance matrices could also potentially benefit, but the training points with near-zero covariances are not necessarily clustered in a way that is easily captured by a space-partitioning data structure. To our knowledge, no existing work has demonstrated a kd-tree-based GP method in high dimensions.

Approximate kernel computations could potentially work if the $\boldsymbol{\alpha}$ weights and test MVMs systematically used the same effective kernel. The difficulty is that simple piecewise-constant kernels are not positive-semidefinite, and cannot be used for GP regression. An approximation that would generally apply is moving groups of points to lie exactly on top of each other. This is a pre-processing step after which points with the same input location could be analytically combined. There would be no need for recursions on a multi-resolution data structure.

3.3. Expansion of the Gaussian kernel

Merging kernel values is a crude approximation. More sophisticated analytical approximations can be made for particular kernel functions. The Fast Gauss Transform (FGT) (Greengard & Strain, 1991; Strain, 1991) and the Improved Fast Gauss Transform (IFGT) (Yang et al., 2005) offer fast MVM operations when using the Gaussian kernel. The most up-to-date description of the IFGT is provided in a tech-report and associated open-source code (Raykar et al., 2005; Raykar et al., 2006).

Results for the IFGT using the authors' code and the FGT using code from Lang et al. (2006) were included in figure 1. Both can provide a speedup on low-dimensional problems with minimal impact on accuracy. Raykar and Duraiswami (2007) have already applied the IFGT to Gaussian process mean prediction. Given that both the IFGT code and the datasets they used are available, we were able to confirm that their results are broadly reproducible. Unlike the kd-tree methods, there is significant, accessible and easily reproducible evidence that expansion-based methods can provide speedups on some real GP regression problems, without impacting accuracy.

However, the usefulness of the expansion-based methods is limited. In low dimensional problems, the test performance often saturates for practical purposes after training on a manageable subset of the training data. The FGT and IFGT have scaling problems on high-dimensional problems where fast methods are really needed. Maintaining a speedup with the IFGT requires the bandwidth or lengthscale to scale proportional to $\sqrt{D}$, where D is the input dimensionality (Raykar et al., 2006). However, our experience is that lengthscales much wider than the width of the data set are not learned for relevant features. Raykar and Duraiswami (2007) state that the current version of the IFGT does not accelerate GP regression on the 21-dimensional SARCOS dataset.

Even on low dimensional problems, using these codes is not without practical difficulties. The (I)FGT performance curves do not extend to small lengthscales in figure 1(b) because some aspect of the codes failed or crashed in these parameter regimes. Even if long lengthscales are appropriate, this limitation makes parameter searches harder to implement.

4. Iterative Methods

All Gaussian process computations can take advantage of fast matrix-vector multiplications through conjugate gradient methods (Gibbs, 1997). Alternative iterative methods may be useful (Li et al., 2007; Liberty et al., 2007), but these are also based on matrix-vector multiplications.

We have tried conjugate gradients and Li et al.'s method, although performing a careful comparison is involved. The conclusion on the datasets we tried was that a working fast matrix-vector multiply code is essential for these methods to have a strong impact. It would be best to see fast MVMs working *convincingly* for the simplest task of test-time mean prediction, before complicating the analysis significantly with outer loop iterations to perform other tasks.

Raykar and Duraiswami (2007) have already explored combining the IFGT with CG methods for training Gaussian process mean prediction. Speedups on some low- to medium-dimensional regression problems were demonstrated. Although in these cases only moderate-sized datasets are needed for high accuracy predictions, so they could be dealt with naively. Obtaining speedups on high-dimensional problems with mathematical numerical methods is an open problem.

5. Discussion

Much of the work on fast kernel matrix computations has focussed on kernel density estimation (KDE). While local kernel regression has a similar flavor to KDE, Gaussian process regression does not. Although some mathematical expressions look familiar, typical kernel lengthscales are in a range seemingly designed to give poor performance with kd-tree and FGT methods. Results with the IFGT show that speedups are possible, although not in the most useful regimes.

Moderate dimensional datasets such as SARCOS give approximately sparse covariance matrices. However, we are unaware of a robust procedure for leveraging this into a significant speedup.

Acknowledgments

This abstract describes experiences during a broader study with Joaquin Quiñonero Candela, Carl Rasmussen, Edward Snelson, and Chris Williams.

References

Freitas, N. D., Wang, Y., Mahdaviani, M., & Lang, D. (2006). Fast Krylov methods for N-body learning. *Advances in Neural Information Processing Systems (NIPS*18): Proceedings of the 2005 Conference*. MIT Press.

Gibbs, M. (1997). *Bayesian Gaussian processes for classification and regression*. PhD thesis, University of Cambridge.

Gray, A. (2004). *Fast kernel matrix-vector multiplication with application to Gaussian process learning* (Technical Report CMU-CS-04-110). School of Computer Science, Carnegie Mellon University.

Gray, A. G., & Moore, A. W. (2001). 'N-body' problems in statistical learning. *Advances in Neural Information Processing Systems (NIPS*13): Proceedings of the 2000 Conference*. MIT Press.

Greengard, L., & Strain, J. (1991). The fast Gauss transform. *SIAM J. Sci. Stat. Comput.*, *12*, 79–94.

Lang, D., Klaas, M., Hamze, F., & Lee, A. (2006). N-body methods code and Matlab binaries. `http://www.cs.ubc.ca/~awll/nbody_methods.html`.

Li, W., Lee, K.-H., & Leung, K.-S. (2007). Large-scale RLSC learning without agony. *Proceedings of the 24th international conference on Machine learning* (pp. 529–536). ACM Press New York, NY, USA.

Liberty, E., Woolfe, F., Martinsson, P.-G., Rokhlin, V., & Tygert, M. (2007). Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, *104*, 20167–72.

Moore, A., Schneider, J., & Deng, K. (1997). Efficient locally weighted polynomial regression predictions. *Fourteenth International Conference on Machine Learning* (pp. 236–244).

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian Processes for machine learning*. MIT Press.

Raykar, V. C., & Duraiswami, R. (2007). Fast large scale Gaussian process regression using approximate matrix-vector products. *Learning workshop 2007, San Juan, Peurto Rico*. Available from: `http://www.umiacs.umd.edu/~vikas/publications/raykar_learning_workshop_2007_full_paper.pdf`.

Raykar, V. C., Yang, C., & Duraiswami, R. (2006). *Improved fast Gauss transform: user manual* (Technical Report). Department of Computer Science, University of Maryland, CollegePark. `http://www.umiacs.umd.edu/~vikas/Software/IFGT/IFGT_code.htm`.

Raykar, V. C., Yang, C., Duraiswami, R., & Gumerov, N. (2005). *Fast computation of sums of Gaussians in high dimensions* (Technical Report CS-TR-4767). Department of Computer Science, University of Maryland, CollegePark.

Shen, Y., Ng, A., & Seeger, M. (2006). Fast Gaussian process regression using KD-trees. *Advances in Neural Information Processing Systems (NIPS*18): Proceedings of the 2005 Conference*. MIT Press.

Strain, J. (1991). The fast Gauss transform with variable scales. *SIAM J. Sci. Stat. Comput*, *12*, 1131–1139.

Yang, C., Duraiswami, R., & Davis, L. (2005). Efficient kernel machines using the improved fast Gauss transform. *Advances in Neural Information Processing Systems (NIPS*17): Proceedings of the 2004 Conference* (pp. 1561–1568). MIT Press.