Efficient Optimization for Sparse Gaussian Process Regression

Article *in* IEEE Transactions on Pattern Analysis and Machine Intelligence · December 2015

DOI: 10.1109/TPAMI.2015.2424873 · Source: arXiv

CITATIONS
18

READS
157

4 authors, including:

Marcus A. Brubaker
York University
40 PUBLICATIONS **2,470** CITATIONS
SEE PROFILE

David J. Fleet
University of Toronto
203 PUBLICATIONS **17,701** CITATIONS
SEE PROFILE

Efficient Optimization for Sparse Gaussian Process Regression

Yanshuai Cao, Marcus A. Brubaker, David J. Fleet, and Aaron Hertzmann

Abstract—We propose an efficient optimization algorithm to select a subset of training data as the inducing set for sparse Gaussian process regression. Previous methods either use different objective functions for inducing set and hyperparameter selection, or else optimize the inducing set by gradient-based continuous optimization. The former approaches are harder to interpret and suboptimal, whereas the latter cannot be applied to discrete input domains or to kernel functions that are not differentiable with respect to the input. The algorithm proposed in this work estimates an inducing set and the hyperparameters using a single objective. It can be used to optimize either the marginal likelihood or a variational free energy. Space and time complexity are linear in training set size, and the algorithm can be applied to large regression problems on discrete or continuous domains. Empirical evaluation shows state-of-art performance in discrete cases, competitive prediction results as well as a favorable trade-off between training and test time in continuous cases.

Index Terms—Gaussian process regression, low rank, matrix factorization, sparsity

1 Introduction

Gaussian process (GP) learning and inference are computationally prohibitive for large datasets, having time complexities $O(n^3)$ and $O(n^2)$, where n is the number of training points. Sparsification algorithms exist that scale linearly in the training set size (see [12] for a review). They construct a low-rank approximation to the GP covariance matrix over the full dataset using a small set of *inducing points*. Some approaches select inducing points from training points [9], [10], [14], [15]. But these methods select the inducing points using ad hoc criteria; i.e., they use different objective functions to select inducing points and to optimize GP hyperparameters. More powerful sparsification methods [16], [17], [18] use a single objective function and allow inducing points to be free parameters in the input domain, learned via gradient descent. This continuous relaxation is not feasible, however, if the input domain is discrete, or if the kernel function is not differentiable in the input variables. As a result, there are problems in myraid domains, like bio-informatics, linguistics and computer vision where current sparse GP regression methods are inapplicable or ineffective.

We introduce an efficient sparsification algorithm for GP regression. The method optimizes a single objective for joint selection of inducing points and GP hyperparameters. Notably, it can be used to optimize either the marginal likelihood, or a variational free energy [17], exploiting the QR factorization of a partial Cholesky decomposition to efficiently approximate the covariance matrix. Because it chooses inducing points from the training data, it is applicable to problems on discrete or continuous input domains. To our knowledge, it is the first method for selecting discrete inducing points and hyperparameters that optimizes a single principled objective, with linear space and time complexity. It is shown to outperform other methods on discrete datasets from bio-informatics and computer vision. On continuous domains it is competitive with the Pseudo-point GP [16] (a.k.a FITC). On the empirical evaluation framework for approximate GPs introduced by Chalupka et al. [6], we obtain a favourable trade-off of performance versus hyperparameter learning time and test time.

1.1 Previous Work

The computational cost of GP learning and inference can be improved with fast matrix inversion techniques, such as Bo and Sminchisescus greedy block coordinate descent [3]. While this allows for fast computation of the predictive mean, it entails repeatedly solving large linear systems for hyperparameter optimization and for computing the predictive variance. Alternatively, efficient state-of-the-art sparsification methods are $O(m^2n)$ in time and $O(mn)$ in space for learning. They compute the predictive mean and variance in time $O(m)$ and $O(m^2)$. Methods based on continuous relaxation, when applicable, entail learning $O(md)$ continuous parameters, where d is the input dimension. In the discrete case, combinatorial optimization is required to select the inducing points, and this is, in general, intractable. Existing discrete sparsification methods therefore use other criteria to greedily select inducing points [9], [10], [14], [15]. Although their criteria are justified, each in their own way (e.g., [10], [14] take an information theoretic perspective), they are greedy and do not use the same objective to select inducing points and to estimate GP hyperparameters.

The variational formulation of Titsias [17] treats inducing points as variational parameters, and gives a unified objective

- *Y. Cao and D. J. Fleet are with the Department of Computer Science, University of Toronto, Ontario, Canada.*
- *M. A. Brubaker is with the TTI-Chicago institute, Chicago, Illinois.*
- *A. Hertzmann is with the Adobe Research, San Francisco, California and the Department of Computer Science, University of Toronto, Ontario, Canada.*

Manuscript received 28 May 2014; revised 21 Jan. 2015; accepted 9 Mar. 2015. Date of publication 19 Apr. 2015; date of current version 6 Nov. 2015. Recommended for acceptance by C. Sminchisescu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2015.2424873

for discrete and continuous inducing point models. In the continuous case, it uses gradient-based optimization to find inducing points and hyperparameters. In the discrete case, our method optimizes the same variational objective of Titsias [17], but is a significant improvement over greedy forward selection using the variational objective as selection criteria, or some other criteria. In particular, given the cost of evaluating the variational objective on all training points, Titsias [17] evaluates the objective function on a small random subset of candidates at each iteration, and then selects the best element from the subset. This approximation is often slow to achieve good results, as we explain and demonstrate below in Section 4.1. That approach uses greedy forward selection, which provides no way to refine the inducing set after hyperparameter optimization, except to discard all previous inducing points and restart selection. Hence, the objective is not guaranteed to decrease after each restart. By comparison, our formulation considers all candidates at each step, revisiting previous selections is efficient, and it is guaranteed to decrease the objective or terminate.

Our low-rank decomposition is inspired by the *Cholesky with Side Information* (CSI) algorithm for kernel machines [1]. We extend that approach to GP regression. First, we alter the form of the low-rank matrix factorization in CSI to be suitable for GP regression with full-rank diagonal noise term in the covariance. Second, the CSI algorithm selects inducing points in a single greedy pass using an approximate objective. We propose an iterative optimization algorithm that swaps previously selected points with new candidates that are guaranteed to lower the objective. Finally, we perform inducing set selection jointly with gradient-based hyperparameter estimation instead of the grid search in CSI. Our algorithm selects inducing points in a principled fashion, optimizing the variational free energy or the log likelihood. It does so with time complexity $O(m^2 n)$, and in practice provides an improved quality-speed tradeoff over other discrete selection methods.

2 Sparse GP Regression

Let $y \in \mathbb{R}$ be the noisy output of a function, f, on input $\mathbf{x}$. Let $X = \{\mathbf{x}_i\}_{i=1}^n$ denote n training inputs, each belonging to input space $\mathcal{D}$, which is not necessarily Euclidean. Let $\mathbf{y} \in \mathbb{R}^n$ be the corresponding vector of training outputs. Under a zero-mean full GP prior, the covariance between two outputs is

$$\mathbb{E}[y_i y_j] = \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sigma^2 1[i = j], \tag{1}$$

where κ is the kernel function, $1[\cdot]$ is the usual indicator function, and σ^2 is the variance of the observation noise. The predictive distribution over the output $f_\star$ at a test point $\mathbf{x}_\star$ is normally distributed. The mean and variance of the predictive distribution can be expressed as

$$\mu_\star = \boldsymbol{\kappa}(\mathbf{x}_\star)^\mathsf{T}\big(K + \sigma^2 I_n\big)^{-1}\mathbf{y}$$
$$v_\star^2 = \kappa(\mathbf{x}_\star, \mathbf{x}_\star) - \boldsymbol{\kappa}(\mathbf{x}_\star)^\mathsf{T}\big(K + \sigma^2 I_n\big)^{-1}\boldsymbol{\kappa}(\mathbf{x}_\star)$$

where I_n is the $n \times n$ identity matrix, K is the kernel matrix whose $i\,j$th element is $\kappa(\mathbf{x}_i, \mathbf{x}_j)$, and $\boldsymbol{\kappa}(\mathbf{x}_\star)$ is the column vector whose ith element is $\kappa(\mathbf{x}_\star, \mathbf{x}_i)$.

The hyperparameters of a GP, denoted $\boldsymbol{\theta}$, comprise the parameters of the kernel function, and the noise variance σ^2. One natural objective for learning $\boldsymbol{\theta}$ is the negative marginal log likelihood (NMLL) of the training data, $-\log\left(P(\mathbf{y}|X, \boldsymbol{\theta})\right)$, given up to a constant by

$$E_{full}(\boldsymbol{\theta}) = \left(\mathbf{y}^\top\big(K + \sigma^2 I_n\big)^{-1}\mathbf{y} + \log|K + \sigma^2 I_n|\right)/2. \tag{2}$$

The computational bottleneck lies in the $O(n^2)$ storage and $O(n^3)$ inversion of the full covariance matrix, $K + \sigma^2 I_n$. To lower this cost with a sparse approximation, Csató and Opper [7] and Seeger et al. [14] proposed the Projected Process (PP) model, wherein a set of m inducing points are used to construct a low-rank approximation of the kernel matrix. In the discrete case, where the inducing points are a subset of the training data, with indices $\mathcal{I} \subset \{1, 2, \ldots, n\}$, and an arbitrary ordering $(i_1, i_2, \ldots, i_k, \ldots, i_m)$ of these indices[1], this approach amounts to replacing the kernel matrix K with the following Nyström approximation [13]:

$$K \simeq \hat{K} = K[:, \mathcal{I}]\, K[\mathcal{I}, \mathcal{I}]^{-1}\, K[\mathcal{I}, :] \tag{3}$$

where $K[:, \mathcal{I}]$ denotes the sub-matrix of K comprising columns indexed by $\mathcal{I}$, and $K[\mathcal{I}, \mathcal{I}]$ is the sub-matrix of K comprising rows and columns indexed by $\mathcal{I}$. We assume the rank of K is m or higher so we can always find such rank-m approximations. The PP NMLL is then algebraically equivalent to replacing K in Eq. (2) with $\hat{K}$, i.e.,

$$E(\boldsymbol{\theta}, \mathcal{I}) = \left(E^D(\boldsymbol{\theta}, \mathcal{I}) + E^C(\boldsymbol{\theta}, \mathcal{I})\right)/2, \tag{4}$$

with data term $E^D(\boldsymbol{\theta}, \mathcal{I}) = \mathbf{y}^\top(\hat{K} + \sigma^2 I_n)^{-1}\mathbf{y}$, and model complexity $E^C(\boldsymbol{\theta}, \mathcal{I}) = \log|\hat{K} + \sigma^2 I_n|$.

The computational cost reduction from $O(n^3)$ to $O(m^2 n)$ associated with the new likelihood is achieved by applying the Woodbury inversion identity to $E^D(\boldsymbol{\theta}, \mathcal{I})$ and $E^C(\boldsymbol{\theta}, \mathcal{I})$. The objective in (4) can be viewed as an approximate log likelihood for the full GP model, or as the exact log likelihood for an approximate model, called the Deterministically Trained Conditional [12].

The same PP model can also be obtained by a variational argument, as in [17]. That is, the variational free energy objective can be shown to be Eq. (4) plus one extra term; i.e.,

$$F(\boldsymbol{\theta}, \mathcal{I}) = \left(E^D(\boldsymbol{\theta}, \mathcal{I}) + E^C(\boldsymbol{\theta}, \mathcal{I}) + E^V(\boldsymbol{\theta}, \mathcal{I})\right)/2, \tag{5}$$

where $E^V(\boldsymbol{\theta}, \mathcal{I}) = \sigma^{-2}\mathrm{tr}(K - \hat{K})$ arises from the variational formulation. It effectively regularizes the trace norm, also known as the nuclear norm, of the approximation residual of the covariance matrix. The kernel machine of [1] also uses a trace norm regularizer of the form $\lambda\,\mathrm{tr}(K - \hat{K})$, however λ is a free parameter that is set manually. Properties of trace norm regularization have been studied extensively [2].

3 Efficient Optimization

We now outline our algorithm for optimizing the variational free energy (5) to select the inducing set $\mathcal{I}$ and the hyperparameters $\boldsymbol{\theta}$. (The negative log-likelihood (4) is

1. We will relax the notation to use $\mathcal{I}$ to denote both the set and the ordered sequence.

similarly minimized by simply discarding the E^V term.) The algorithm is a form of hybrid coordinate descent that alternates between discrete optimization of inducing points, and continuous optimization of the hyperparameters. We first describe the algorithm to select inducing points, then discuss continuous hyperparameter optimization and termination criteria in Section 3.4.

Finding the optimal inducing set is a combinatorial problem; global optimization is intractable. Instead, the inducing set is initialized to a random subset of the training data, which is then refined by a fixed number of swap updates at each iteration.[2] In a single swap update, a randomly chosen inducing point is considered for replacement. If swapping does not improve the objective, then the original point is retained.

There are $n - m$ potential replacements for each swap update; the key is to efficiently determine which is likely to maximally improve the objective. With the techniques described below, the computation time required to approximately evaluate all possible candidates and swap an inducing point is $O(mn)$. Swapping all inducing points once takes $O(m^2 n)$ time.

3.1 Factored Representation

To support efficient evaluation of the objective and swapping, we use a factored representation of the kernel matrix. Given an inducing set $\mathcal{I}$ of k points, for any $k \leq m$, the low-rank Nyström approximation to the kernel matrix (Eq. 3) can be expressed in terms of a partial Cholesky factorization:

$$\hat{K} = K[:, \mathcal{I}]\, K[\mathcal{I}, \mathcal{I}]^{-1}\, K[\mathcal{I}, :] = L(\mathcal{I}) L(\mathcal{I})^\top, \tag{6}$$

where $L(\mathcal{I}) \in \mathbb{R}^{n \times k}$ is, up to permutation of rows, a lower trapezoidal matrix (i.e., has a $k \times k$ top lower triangular block, again up to row permutation).[3] The proof of the identity in Eq. (6) follows from Proposition 1 in [1], and the fact that, given the ordered sequence of pivots $\mathcal{I}$, the partial Cholesky factorization is unique.

Using this factorization and the Woodbury identities (dropping the dependence on $\boldsymbol{\theta}$ and $\mathcal{I}$ for clarity), the terms of the negative marginal log-likelihood (4), or the variational free energy (5), become

$$E^D = \sigma^{-2}\Big(\mathbf{y}^\top \mathbf{y} - \mathbf{y}^\top L \big(L^\top L + \sigma^2 I\big)^{-1} L^\top \mathbf{y}\Big) \tag{7}$$

$$E^C = \log\Big((\sigma^2)^{n-k} |L^\top L + \sigma^2 I|\Big) \tag{8}$$

$$E^V = \sigma^{-2}(\mathrm{tr}(K) - \mathrm{tr}(L^\top L)). \tag{9}$$

We can further simplify the data term E^D and the complexity term E^C by augmenting the factor matrix as $\widetilde{L} = [L^\top, \sigma I_k]^\top$, where I_k is the $k \times k$ identity matrix, and $\widetilde{\mathbf{y}} = [\mathbf{y}^\mathsf{T}, \mathbf{0}_k^\mathsf{T}]^\mathsf{T}$ is the $\mathbf{y}$ vector with k zeroes appended:

$$E^D = \sigma^{-2}\Big(\mathbf{y}^\top \mathbf{y} - \widetilde{\mathbf{y}}^\top \widetilde{L}\, (\widetilde{L}^\top \widetilde{L})^{-1}\, \widetilde{L}^\top \widetilde{\mathbf{y}}\Big) \tag{10}$$

$$E^C = \log\Big((\sigma^2)^{n-k} |\widetilde{L}^\top \widetilde{L}|\Big). \tag{11}$$

Now, let $\widetilde{L} = QR$ be a QR factorization of $\widetilde{L}$, where $Q \in \mathbb{R}^{(n+k)\times k}$ has orthonormal columns and $R \in \mathbb{R}^{k \times k}$ is invertible. The first two terms in the objective simplify further to

$$E^D = \sigma^{-2}\Big(\|\mathbf{y}\|^2 - \|Q^\top \widetilde{\mathbf{y}}\|^2\Big) \tag{12}$$

$$E^C = (n - k)\log(\sigma^2) + 2\log|R|. \tag{13}$$

3.2 Factorization Update

Here we present the mechanics of the swap update algorithm. See the Appendices for pseudo-code. Suppose we wish to swap inducing point i with candidate point j in $\mathcal{I}_m$, the inducing set of size m. We first modify the factor matrices in order to remove point i from $\mathcal{I}_m$, i.e., to downdate the factors, then update all the key terms using one step of Cholesky and QR factorization with the new point j.

Downdating to remove inducing point i requires that we shift the corresponding columns/rows in the factorization to the right-most columns of $\widetilde{L}$, Q, R and to the last row of R. We can then simply discard these last columns and rows, and modify related quantities. When permuting the order of the inducing points, the underlying GP model is invariant, but the matrices in the factored representation are not. If needed, any two points in $\mathcal{I}_m$, can be permuted, and the Cholesky or QR factors can be updated in time $O(mn)$. This is done with the efficient pivot permutation presented in Algorithm 4 of Appendix B, which is based on the permutation algorithm in the Appendix of [1], with minor modifications to account for the augmented form of $\widetilde{L}$. In this way, downdating and removing i take $O(mn)$ time, as does the updating with point j.

After downdating, we have factors $\widetilde{L}_{m-1}$,Q_{m-1}, R_{m-1}, and inducing set $\mathcal{I}_{m-1}$. To add j to $\mathcal{I}_{m-1}$, and update the factors to rank m, one step of Cholesky factorization following Algorithm 3 of Appendix A is performed with point j,[4] for which, the new column to append to $\widetilde{L}$ is formed as

$$\boldsymbol{\ell}_m = \frac{(K - \hat{K}_{m-1})[:, j]}{\sqrt{(K - \hat{K}_{m-1})[j, j]}}, \tag{14}$$

where $\hat{K}_{m-1} = L_{m-1} L_{m-1}{}^\mathsf{T}$. Then, we set $\widetilde{L}_m = [\widetilde{L}_{m-1} \tilde{\boldsymbol{\ell}}_m]$, where $\tilde{\boldsymbol{\ell}}_m$ is just $\boldsymbol{\ell}_m$ augmented with $\sigma \mathbf{e}_m$, and $\mathbf{e}_m$ being the mth standard basis vector $[0, 0, \ldots, 1]^\top$.

The final updates are $Q_m = [Q_{m-1} \mathbf{q}_m]$, where $\mathbf{q}_m$ is given by Gram-Schmidt orthogonalization step:

$$\mathbf{q}_m = \frac{((I - Q_{m-1} Q_{m-1}^\top)\tilde{\boldsymbol{\ell}}_m)}{\|(I - Q_{m-1} Q_{m-1}^\top)\tilde{\boldsymbol{\ell}}_m\|}. \tag{15}$$

and R_m is updated from R_{m-1} so that $\widetilde{L}_m = Q_m R_m$.

2. The inducing set can be incrementally constructed, as in [1], also shown in Algorithms 2 of Appendix A, however we found no benefit to this.

3. Note that k is a dummy variable used solely to describe that the factorization shown in this section applies to any rank $1 \leq k \leq m$. It is not a parameter of the algorithm.

4. In Algorithm 3, j is related to the permuation by $j = \mathbf{p}[t]$, where $\mathbf{p}$ is the vector used in actual implementation to track the permutation.

3.3 Evaluating Candidates

Next we show how to select candidates for inclusion in the inducing set. We first derive the exact change in the objective due to adding an element to $\mathcal{I}_{m-1}$. Later we provide an approximation to this objective change that can be computed efficiently.

Given an inducing set $\mathcal{I}_{m-1}$, and matrices $\widetilde{L}_{m-1}, Q_{m-1}$, and R_{m-1}, we wish to evaluate the change in Eq. (5) for $\mathcal{I}_m = \mathcal{I}_{m-1} \cup j$. That is, $\Delta F \equiv F(\boldsymbol{\theta}, \mathcal{I}_{m-1}) - F(\boldsymbol{\theta}, \mathcal{I}_m) = (\Delta E^D + \Delta E^C + \Delta E^V)/2$, where, based on the mechanics of the incremental updates above, one can show that

$$\Delta E^D = \frac{\sigma^{-2}(\tilde{\mathbf{y}}^\top (I - Q_{m-1}Q_{m-1}^\top)\tilde{\boldsymbol{\ell}}_m)^2}{\|(I - Q_{m-1}Q_{m-1}^\top)\tilde{\boldsymbol{\ell}}_m\|^2} \tag{16}$$

$$\Delta E^C = \log(\sigma^2) - \log\|(I - Q_{m-1}Q_{m-1}^\top)\tilde{\boldsymbol{\ell}}_m\|^2 \tag{17}$$

$$\Delta E^V = \sigma^{-2}\|\boldsymbol{\ell}_m\|^2. \tag{18}$$

This gives the exact decrease in the objective function after adding point j. For a single point this evaluation is $O(mn)$, so to evaluate all $n - m$ points would be $O(mn^2)$.

3.3.1 Fast Approximate Cost Reduction

While $O(mn^2)$ is prohibitive, computing the exact change is not required. Rather, we only need a ranking of the best few candidates. Thus, instead of evaluating the change in the objective exactly, we use an efficient approximation based on a small number, z, of training points which provide information about the residual between the current low-rank covariance matrix (based on inducing points) and the full covariance matrix. After this approximation proposes a candidate, we use the actual objective to decide whether to include it. The techniques below reduce the complexity of evaluating all $n - m$ candidates to $O(zn)$.

To compute the change in objective for one candidate, we need the new column of the updated Cholesky factorization, $\boldsymbol{\ell}_m$. In Eq. (14) this vector is a (normalized) column of the residual $K - \hat{K}_{m-1}$ between the full kernel matrix and the Nyström approximation. Now consider the full Cholesky decomposition of $K = L^*L^{*\top}$ where $L^* = [L_{m-1}, L(\mathcal{J}_{m-1})]$ is constructed with $\mathcal{I}_{m-1}$ as the first pivots and $\mathcal{J}_{m-1} = \{1, \ldots, n\} \backslash \mathcal{I}_{m-1}$ as the remaining pivots, so the residual becomes

$$K - \hat{K}_{m-1} = L(\mathcal{J}_{m-1})L(\mathcal{J}_{m-1})^\top. \tag{19}$$

We approximate $L(\mathcal{J}_{m-1})$ with a rank $z \ll n$ matrix, L_z, by taking z points from $\mathcal{J}_{m-1}$ and performing a partial Cholesky factorization of $K - \hat{K}_{m-1}$ using these pivots. The residual approximation becomes:

$$K - \hat{K}_{m-1} \approx L_z L_z^\top \tag{20}$$

yielding an approximation to the exact in Eq. (14) by

$$\boldsymbol{\ell}_m \approx \frac{(L_z L_z^\top)[:, j]}{\sqrt{(L_z L_z^\top)[j, j]}}. \tag{21}$$

The pivots used to construct L_z are called *information pivots*. Their selection is discussed in Section 3.3.2.

The approximations to ΔE_k^D, ΔE_k^C and ΔE_k^V, Eqs. (16)-(18), for all candidate points, involve the following terms: $\mathrm{diag}(L_z L_z^\top L_z L_z^\top)$, $\mathbf{y}^\top L_z L_z^\top$, and $(Q_{k-1}[1:n,:])^\top L_z L_z^\top$. The first term can be computed in time $O(z^2 n)$, and the other two in $O(zmn)$ with careful ordering of matrix multiplications.[5] Computing L_z costs $O(z^2 n)$, but can be avoided since information pivots change by at most one when an information pivot is added to the inducing set and needs to be replaced. The techniques in Section 3.2 bring the associated update cost to $O(zn)$ by updating L_z rather than recomputing it. These z information pivots are equivalent to the "look-ahead" steps of Bach and Jordan's CSI algorithm [1], but as described in Section 3.3.2, there is a more effective way to select them.

3.3.2 Ensuring a Good Approximation

Selection of the information pivots determines the approximate objective, and hence the candidate proposal. To make the optimization less likely to be stuck in bad local minima due to the bias in a particular fixed approximation, we randomly select the z pivots, and resample after a random number of swapping iteration (on average once per five swaps).

This is different from the CSI algorithm [1], which greedily selects points to find an approximation of the residual $K - \hat{K}_{m-1}$ in Eq. (14) that is optimal in terms of a bound of the trace norm. The goal, however, is to approximate Eqs. (16)-(18) . By analyzing the role of the residual matrix, we see that the information pivots provide a low-rank approximation to the orthogonal complement of the space spanned by current inducing set. With a fixed set of information pivots, parts of that subspace may never be captured. This suggests that we might occasionally update the entire set of information pivots. Although information pivots are changed when one is moved into the inducing set, we find empirically that this is insufficient. Instead, our proposed randomization works better than optimizing the information pivots as in [1].

From a theoretical perspective, to ensure a good approximation of the residual $K - \hat{K}_{m-1}$, the number of information pivots z has a subtle dependency on the difficulty of the problem. In general, this approximation is good if, in the kernel feature space, the projection of the points $\mathcal{J}_{m-1}$ (including the z information pivots and the other unselected points) onto the orthogonal complement of the subspace spanned by the $\mathcal{I}_{m-1}$ inducing points are "close". This depends both on the properties of the reproducing kernel Hilbert space (RKHS) defined by the kernel and the points themselves. In the case of the square exponential kernel, the characteristic length scale is the property that matters, since a long length scale implies points are generally closer to each other. For other kernels, such as a periodic kernel, this intuition does not necessarily translate to "smoothness" of functions drawn from the RKHS.

In practice however, z as small as 16 works well on many problems, mainly because the overall swapping

5. Both can be further reduced to $O(zn)$ by appropriate caching during the updates of Q,R and $\widetilde{L}$, and L_z.

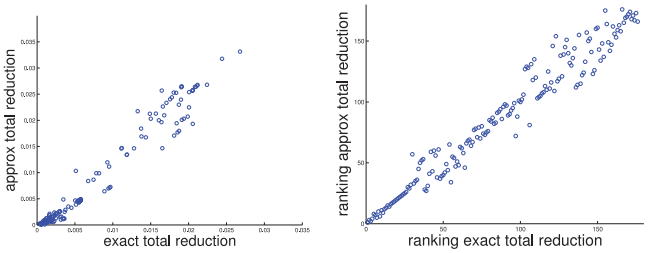


Fig. 1. Exact vs approximate costs, based on the 1D example of Section 4, with $z=10$, $n=200$.

routine only needs a good ranking of the candidates. In fact, as long as the ordering of the top few candidates are the same under the approximate and exact change in objective, the iterative swapping algorithm always picks the optimal replacement candidate. In this case, the approximation will give the same inducing set as the exact case; while slight differences in ranking will generally give a replacement candidate that is still good enough to decrease the objective (albeit "smaller steps"). Fig. 1 illustrates this reasoning by comparing the exact and approximate cost reduction for candidate inducing points (left), and their respective rankings (right). The approximation is shown to work well.

When candidates that do not decrease the objective are proposed based on the approximation, they will be rejected after evaluating the change in the true objective function. This mechanism makes the overall algorithm even more robust to the quality and number of information pivots, as well as the frequency of updates. We find that rejection rates are typically low during early iterations ($<20\%$), but increase as optimization nears convergence (to 30 or 40 percent). Rejection rates also increase for sparser models, where each inducing point plays a more critical role and is harder to replace.

3.4 Hybrid Optimization

The overall hybrid optimization procedure performs block coordinate descent in the inducing points and the continuous hyperparameters as summarized in Algorithm 1.

The discussion so far has focused on the discrete phase of the algorithm, as the continuous phase is mostly straightforward. The Nyström expression for $\hat{K}$ in Eq. (6) allows us to compute the gradient with respect to the hyperparameters analytically. After the continuous steps, the hyperparameters have changed, so all factorization matrices have to be recomputed. This batch re-computation can be done efficiently by first computing $L_m = K[:, \mathcal{I}_m]\, M_m^{-\top}$, where $M_m M_m^\top = K[\mathcal{I}_m, \mathcal{I}_m]$ is the full Cholesky factorization of the m-by-m submatrix of the kernel covariance matrix indexed by $\mathcal{I}_m$; then $\widetilde{L}_m$ follows from its definition in Section 3.1, while Q_m, R_m can be computed using batch QR factorization given $\widetilde{L}_m$.

In practice, because we alternate the discrete and continuous phases for many training epochs, attempting to swap every inducing point in each epoch is unnecessary, just as there is no need to run hyperparameter optimization until convergence at every epoch. As long as all inducing set points are eventually considered we find that optimized models can achieve similar performance with shorter learning times.

Algorithm 1. Hybrid optimization of inducing points and hyperparameters

procedure HYBRID_OPT($\mathcal{I}_m$, $\boldsymbol{\theta}$, $\widetilde{L}_m$, Q_m, R_m)
  while improvement $>$ threshold & time budget > 0 **do**
    Randomly sample $\mathcal{S} \subseteq \mathcal{I}_m$ of predefined size to consider for swapping; ▷ Begin discrete steps
    for all $i \in \mathcal{S}$ **do**
      Save a copy of the factorization matrices $\widetilde{L}_m$, Q_m, R_m, as well as the objective value as F_m^{old};
      Down-date the factorization matrices as described in Section 3.2 to remove i;
      Compute the true objective value F_{m-1} over the down-dated model with $\mathcal{I}_m \backslash \{i\}$, using (12), (13) and (9);
      Select a replacement candidate using the fast approximate cost change from Section 3.3.1;
      Evaluate the exact objective change ΔF, using (16), (17), and (18);
      Get the objective value with the new candidate: $F_m^{new} = F_{m-1} + \Delta F$;
      if $F_m^{new} < F_m^{old}$ **then**
        Include the candidate replacing i in $\mathcal{I}$ and update the matrices as in Section 3.2, set $F_m^{old} = F_m^{new}$;
      else
        Reject the swap proposal and revert to the factorization with i;
      end if
      Update the information pivots if needed as in Sections 3.3.1 and 3.3.2.
    end for
    ▷ Begin continuous steps
    With $\mathcal{I}_m$ fixed, run a fixed number of nonlinear conjugate gradients (CG) steps to optimize the objective in Eq. (5) with respect to $\boldsymbol{\theta}$.
    With the new hyperparameters, batch recompute the factorization matrices $\widetilde{L}_m$, Q_m, R_m, as described in Section 3.4.
  end while
  return $\mathcal{I}_m$, $\boldsymbol{\theta}$, $\widetilde{L}_m$, Q_m, R_m
end procedure

3.5 Variants

There are alternative ways of selecting information pivots, which give slight variants to the main algorithm described above. Here, we define two such variants, and later explore them experimentally in Section 4.1, and in particular in Figs. 3f and 3c.

The first variant, OI, uses optimized information pivots as in the CSI algorithm instead of randomly chosen ones. More specifically, each time a new information pivot needs to be selected, we take the one that has the maximum $\mathbf{d}$ values, where $\mathbf{d}$ (defined in Appendix A), is the amount of prior variance at that point which is not yet explained by the existing factorization. This variant of our algorithm has an interesting connection to the Informative Vector Machine (IVM) [10]. If the likelihood model is isotropic Gaussian, then the maximum reduction in entropy criteria of IVM is equivalent to selection based on $\mathrm{argmax}\{\mathbf{d}\}$. Therefore, this variant actually selects information pivots using the IVM criteria, except that the information pivots are not part of the sparse representation as in IVM.

The second variant actively adapts the size of information pivot set, and is referred to as AA. Initialized to a small size, and given an upper bound z, this variant exponentially grows the information pivot set size whenever a proposed candidate is rejected, and shrinks it linearly whenever one is accepted. The idea behind the AA variant is the following: as in most optimizations, large progress should be easier to achieve at the beginning comparing to later when closer to convergence, hence less computation is needed to construct a useful approximation at early stages.

4 Experiments and Analysis

For the experiments that follow we jointly learn inducing points and hyperparameters, a more challenging task than learning inducing points with known hyperparameters [14], [16]. For all but the 1D example, the number of inducing points swapped per epoch is $\min(60, m)$. The maximum number of function evaluations per epoch in conjugate gradient hyperparameter optimization is $\min(20, \max(15, 2d))$, where d is the number of continuous hyperparameters. Empirically we find the algorithm is robust to changes in these limits. We use two performance measures, (a) standardized mean square error (SMSE), $\frac{1}{N}\Sigma_{t=1}^{N}(\hat{y}_t - y_t)^2/\hat{\sigma}_*^2$, where $\hat{\sigma}_*^2$ is the sample variance of test outputs $\{y_t\}$, and (b) standardized negative log probability (SNLP) defined in [13] as

$$\mathrm{SNLP} = \frac{1}{2N}\Sigma_{t=1}^{N}\left(\log\left(2\pi\hat{\sigma}_t^2\right) + (\hat{y}_t - y_t)^2/\hat{\sigma}_t^2\right) - \mathrm{CST}, \quad (22)$$

where $\hat{\sigma}_t^2$ is the point-wise predictive variance (including the observation noise variance), and $\mathrm{CST} = \frac{1}{2N}\Sigma_{t=1}^{N}(\log(2\pi\hat{\sigma}^2) + (\bar{y} - y_t)^2/\hat{\sigma}^2)$ is the negative log probability under a Gaussian $\mathcal{N}(\bar{y}, \hat{\sigma}^2)$ whose mean and variance are sample estimates from training $\mathbf{y}$.

4.1 Discrete Input Domain

4.1.1 Methods Compared

We first show results on two discrete datasets with kernels that are not differentiable in the input variable x. Because continuous relaxation methods are not applicable, we compare to discrete selection methods, namely, random selection as baseline (Random), greedy subset-optimal selection of Titsias [17] with either 16 or 512 candidates (Titsias-16 and Titsias-512), and Informative Vector Machine [10]. For learning continuous hyperparameters, each method optimizes the same objective using non-linear CG.

For our algorithm we use $z = 16$ information pivots with random selection (CholQR-z16). Later, we show how variants of our algorithm trade-off speed and performance. Additionally, we also compare to least-square kernel regression using CSI (in Fig. 3c).

4.1.2 Implementation of Compared Methods

To ensure fair comparison, all sparse GP methods used in the discrete domain experiments (CholQR, Random, Titsias', and IVM) use the same code for computing the variational free energy objective function and its gradient. For the discrete inducing point selection part of IVM, we use Lawrence's IVM toolbox.[6] Furthermore, care is taken to ensure consistent initialization and termination criteria. For each random seed, all methods start with exactly the same initial hyperparameters and inducing points. All methods except IVM have the same termination criteria: i.e., when they fail to decrease the objective function by a threshold amount, or when they exceed the computational budget. For IVM, because of the inconsistent objectives issue, the variational (or marginal likelihood) objective values highly fluctuate when alternating between the discrete and continuous phases, as demonstrated in Figs. 3d and 3e. Therefore in order to terminate learning at reasonable time for IVM, we make it stop either when there is insufficient change in parameters (no change in inducing points and change in hyperparameters below a predefined threshold), or if the training epoch is larger than 10, and the average relative change in objective function value for the past 10 epochs is below a predefined threshold.

4.1.3 BindingDB Dataset and Graph Kernels

The first discrete dataset, from bindingdb.org, concerns the prediction of binding affinity for a target (Thrombin), from the 2D chemical structure of small molecules (represented as graphs). We do 50-fold random splits to 3,660 training points and 192 test points for repeated runs. We use a compound kernel, comprising 14 different labeled and unlabeled graph kernels [18], [19], [20], [21], [22], [23], [24], [25],[7] Each graph kernel has its own data variance hyperparameter determining its relevance, learned from data during continuous hyperparameter optimization.

4.1.4 HoG Dataset and Histogram Intersection Kernels

The second discrete domain problem comes from the Twin Gaussian Processes work by Bo and Sminchisescu [4], where the task is to predict 3D human joint position from histograms of HoG image features [8]. Training and test sets have 4,819 and 4,811 data points. Because our goal is the general purpose sparsification method for GP regression, we make no attempt at the more difficult problem of modelling the multivariate output structure in the regression as in [4]. Instead, we predict the vertical position of joints independently, using a histogram intersection kernel [11], having four hyperparameters: one noise variance, and three data variances corresponding to the kernel evaluated over the HoG from each of three cameras. We select and show results on the representative left wrist here.

The results in Figs. 2 and 3 show that CholQR-z16 outperforms the baseline methods in terms of test-time predictive power with significantly lower training time. Titsias-16 and Titsias-512 shows similar test performance, but they are two to four orders of magnitude slower than CholQR-z16 (see Figs. 3d and 3e). Indeed, Fig. 3a shows that the training time for CholQR-z16 is comparable to IVM and Random selection, but with much better performance. The poor performance of Random selection highlights the importance of selecting good inducing points, as no amount of hyperparameter optimization can correct for poor inducing points. Fig. 3a also shows IVM to be somewhat slower due

6. From dcs.shef.ac.uk/people/N.Lawrence/ivm

7. Code from mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/Graphkernels

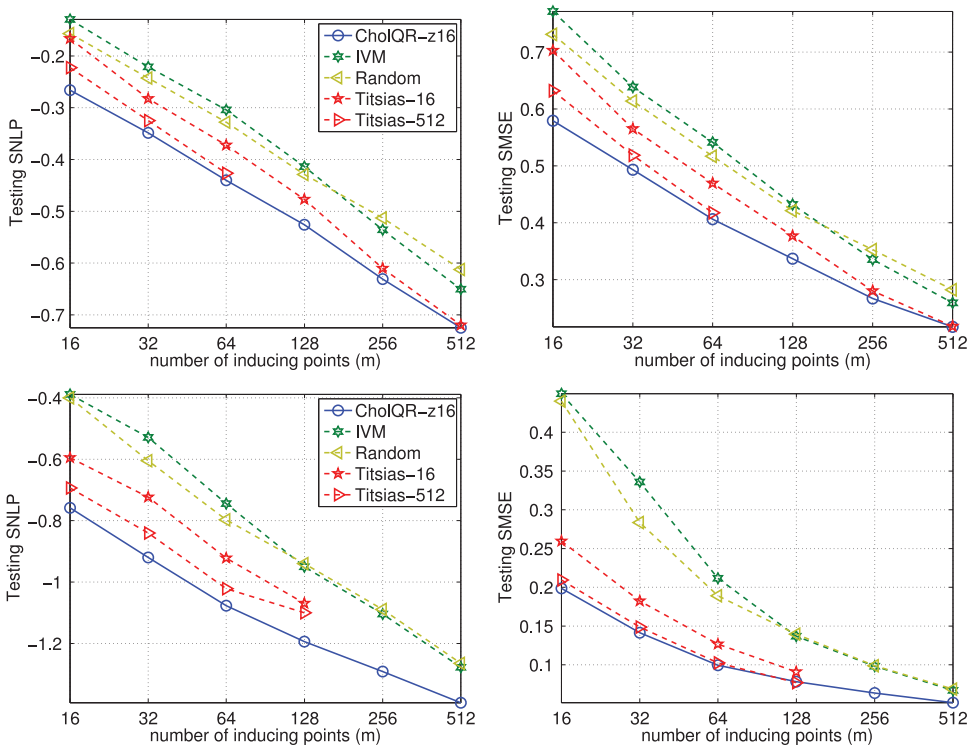


Fig. 2. Test performance on discrete datasets. *(top row)* BindingDB, values at each marker is the average of 150 runs (50-fold random train/test splits times three random initialization); *(bottom row)* HoG dataset, each marker is the average of 10 randomly initialized runs.

to the increased number of iterations needed, even though per epoch, IVM is faster than CholQR. When stopped earlier, IVM test performance further degrades.

Finally, Figs. 3c and 3f show the trade-off between the test SMSE and training time for variants of CholQR, with baselines and CSI kernel regression [1]. For CholQR we

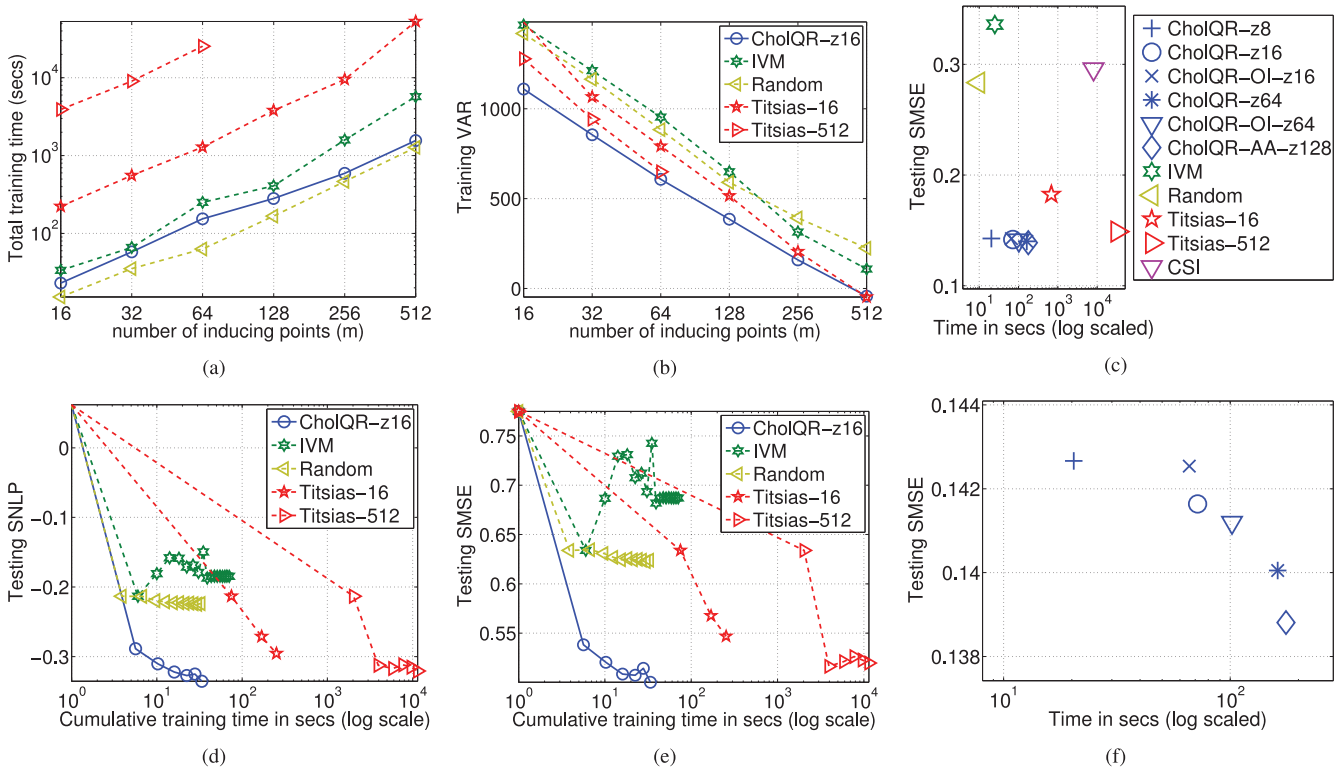
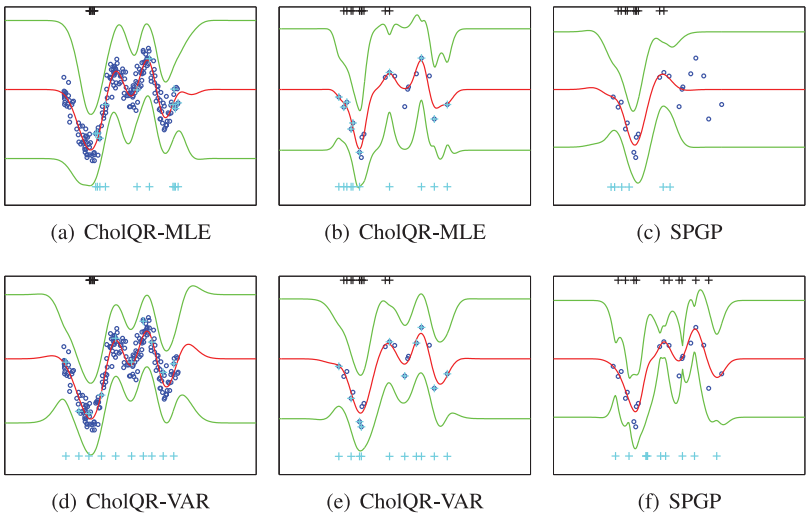


Fig. 3. Training time versus test performance on discrete datasets. (a) the average BindingDB training time; (b) the average BindingDB objective function value at convergence; (d) and (e) show test scores versus training time with $m = 32$ for a single run; (c) shows the trade-off between training time and testing SMSE on the HoG dataset with $m = 32$, for various methods including multiple variants of CholQR and CSI; (f) a zoomed-in version of (c) comparing the variants of CholQR.



(a) CholQR-MLE (b) CholQR-MLE (c) SPGP

(d) CholQR-VAR (e) CholQR-VAR (f) SPGP

Fig. 4. Snelson's 1D example: prediction mean (red curves); one standard deviation in prediction uncertainty (green curves); inducing point initialization (black points at top of each figure); learned inducing point locations (the cyan points at the bottom, also overlaid on data for CholQR).

consider different numbers of information pivots (denoted z8, z16, z64 and z128), and different strategies for their selection including random selection, optimized information pivots (denote OI) and adaptively growing the information pivot set (denoted AA). See Section 3.5 for details about the later two strategies. These variants of CholQR trade-off speed and performance (3(f)), all significantly outperform the other methods (3(c)); CSI, which uses grid search to select hyper-parameters, is slow and exhibits higher SMSE. Interestingly, the variational sparse GP formulation with random inducing points is on par with CSI in terms of accuracy; while with the same sparse GP framework, our hybrid optimization makes CholQR methods significantly better. This suggests that the hybrid optimization is the main source of accuracy improvement over CSI.

4.1.5 Redundant Kernels for Avoiding Local Minimum

In the BindingDB problem, because we try to automatically select relevant kernels from many choices as well as inducing points, the optimization is prone to be trapped in bad local minimum. In particular, bad initialization could lead to local minima where good kernels are dropped while bad ones are kept; this in turn renders selecting good inducing points impossible.

In order to fix this problem when learning relevance from many kernels for sparse GP, we include two kernels that are redundant but useful to facilitate the automatic relevance learning for all the methods. The first redundant kernel is simply the sum of all graph kernels mentioned above; the second one is a constant identity, which is redundant because GP has a diagonal noise term.

The two extra redundant kernels allow us to handle this problem without any modification to the learning algorithms or putting explicit prior over hyper-parameters. The sum kernel forces all individual kernels to be active at the beginning until the data variance (relevance) on this sum kernel is reduced to zero. In a way, it acts as a temporary parameter sharing that is automatically turned off by the optimization once the truly relevant kernels are found. The second constant diagonal kernel reduces the problem of bad local minima where optimization quickly drives all data variance hyper-parameters to zero, and use very large data noise to explain the observations. This is often the case where bad initial hyper-parameters and/or inducing points give a GP model that cannot interpret the data at all.

Most importantly, with the variational energy objective, hyperparameter optimization for all methods learned to reduce the data variance (relevance) on these two redundant kernels to zero after a few training epochs; while without this technique, we obtain good solutions in most cases, but it produces degeneracy with some initializations.

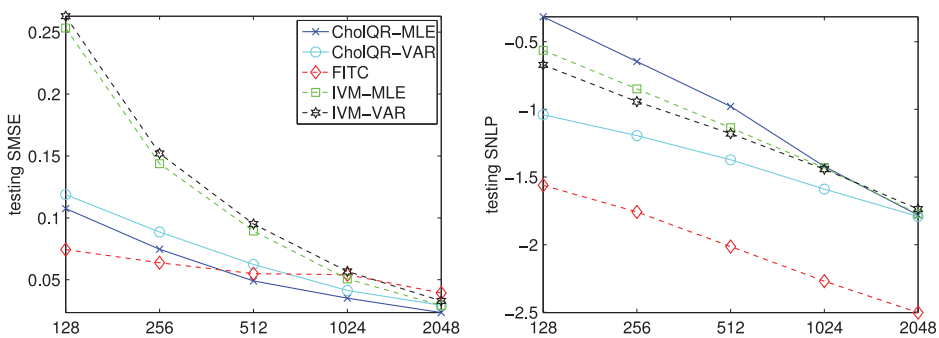


Fig. 5. Test scores on KIN40K as function of number of inducing points: for each number of inducing points the value plotted is averaged over 10 runs from 10 different (shared) initializations.

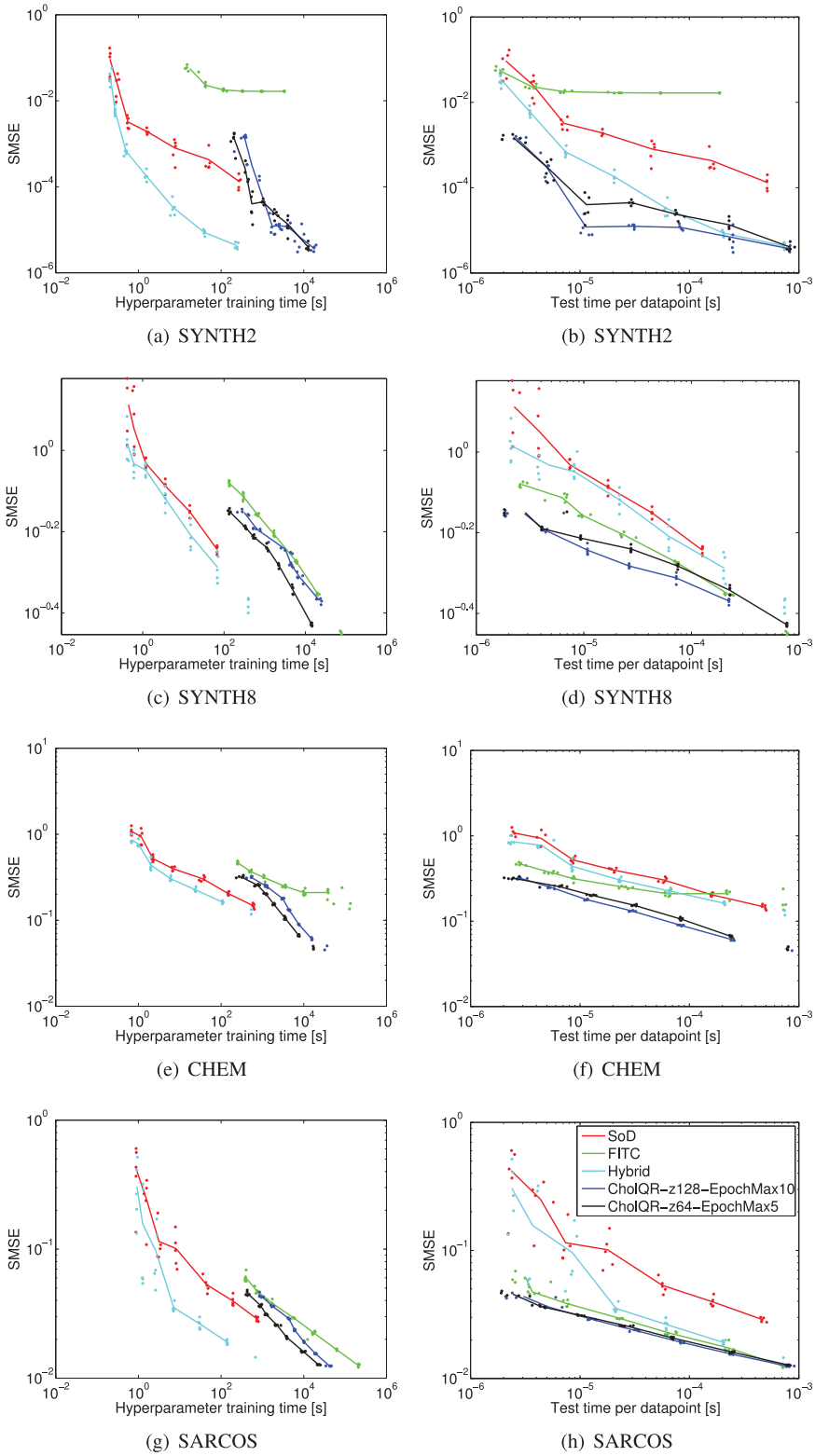


Fig. 6. Comparison of SMSE on the four benchmark datasets: (left column) SMSE as a function of hyperparameter learning time; (right column) SMSE as a function of testing time. Both axes are log-scaled in all figures.

4.2 Continuous Input Domain

Although CholQR was developed for discrete input domains, it can be competitive on continuous domains. To that end, we compare to FITC [16] and IVM [10], using RBF kernels with one length-scale parameter per input dimension; $\kappa(\mathbf{x}_i, \mathbf{x}_j) = c\exp(-0.5\sum_{t=1}^{d} b_t(\mathbf{x}_i^{(t)} - \mathbf{x}_j^{(t)})^2)$. We show results from both the PP log likelihood and variational objectives, suffixed by *MLE* and *VAR*.

We use the 1D toy dataset of [16] to show how the PP likelihood with gradient-based optimization of inducing points is easily trapped in local minima. Figs. 4a and 4d show that for this dataset our algorithm does not get

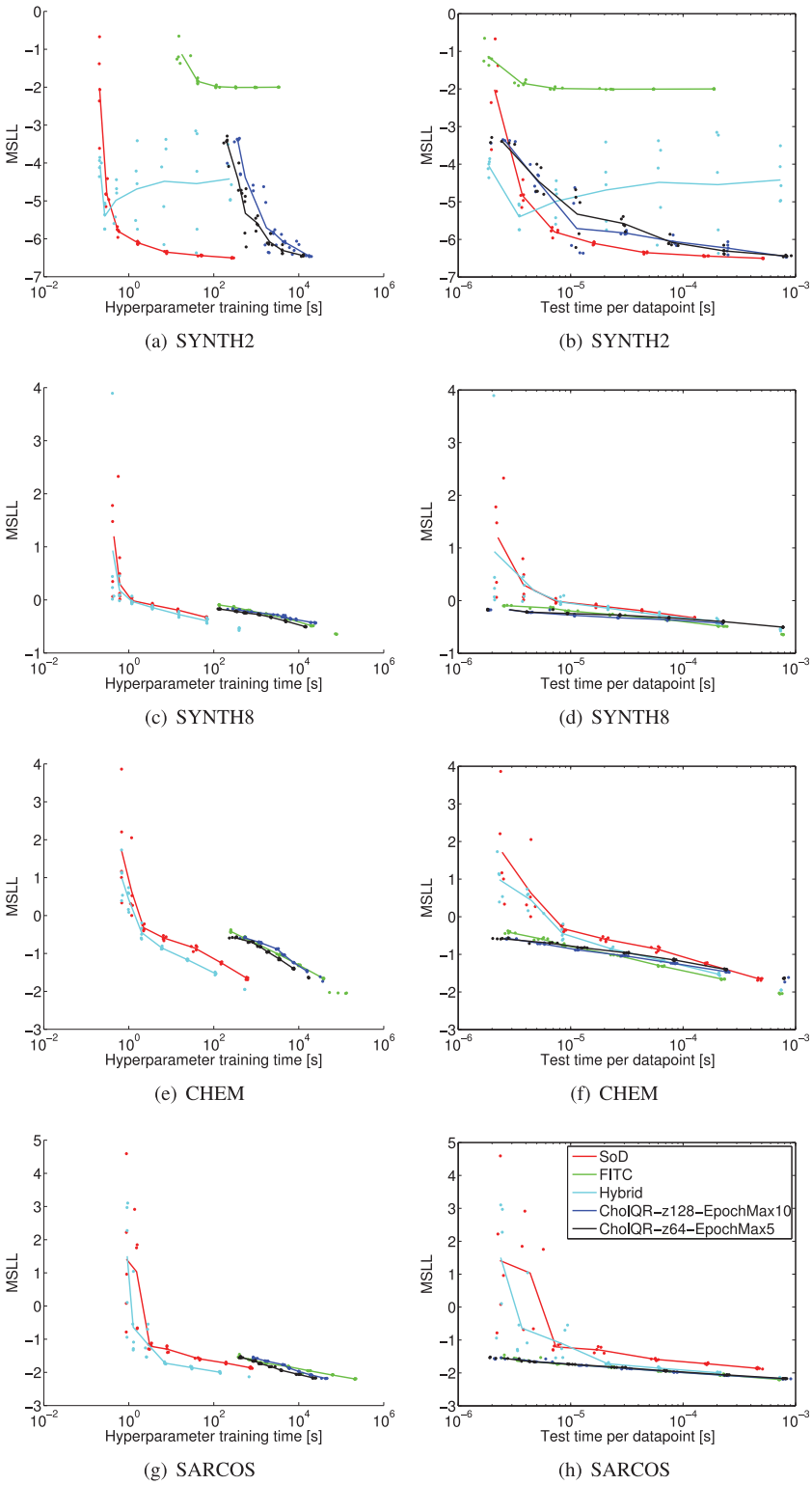


Fig. 7. Comparison of SNLP on the four benchmark datasets: (left column) SNLP as a function of hyperparameter learning time; (right column) SNLP as a function of testing time. Horizontal (time) axes are log-scaled in all figures.

trapped when initialization is poor (as in Fig. 1c of [16]). To simulate the sparsity of data in high-dimensional problems we also down-sample the dataset to 20 points (every 10th point). Here CholQR out-performs FITC (see Figs. 4b, 4e, and 4c). By comparison, Fig. 4f shows FITC learned with a more uniform initial distribution of inducing points avoids this local optima and achieves a better negative log likelihood of 11.34 compared to 14.54 in Fig. 4c.

Finally, we compare CholQR to FITC [16] and IVM [10] on a large dataset. *KIN40K* concerns nonlinear forward kinematic prediction. It has 8D real-valued inputs and scalar outputs, with 10 K training and 30 K test points. We perform linear de-trending and re-scaling as pre-processing.

For FITC we use the implementation of [16]. Fig. 5 shows that CholQR-VAR outperforms IVM in terms of SMSE and SNLP. Both CholQR-VAR and CholQR-MLE outperform FITC in terms of SMSE on KIN40K with large m, but FITC exhibits better SNLP. This disparity between the SMSE and SNLP measures for CholQR-MLE is consistent with findings about the PP likelihood in [17]. CholQR methods use $z = 128$ for this experiment.

4.3 Large Scale Time and Prediction Benchmark

Recently, Chalupka et al. [6] introduced an empirical evaluation framework for approximate GP methods, and showed that subset of data (SoD) often compares favorably to more sophisticated sparse GP methods. Using this framework, we benchmarked CholQR against SoD, FITC, and a heuristic approach called Hybrid by Chalupka et al. [6], which learns hyperparameters using SoD likelihood but uses FITC for prediction with these learned hyperparameters. We use all four large scale datasets from the framework, including two synthetic sets, SYNTH2 and SYNTH8, of $30{,}543$ training points of two and eight dimensional inputs respectively, as well as a 15 dimensional CHEM dataset of $31{,}535$ training points, and the 21 dimensional SARCOS dataset of $44{,}484$ training points. As shown in Figs. 6 and 7, our experiments show that CholQR outperforms FITC in speed and predictive scores. Compared to SoD and Hybrid, CholQR finds a much sparser model. It is slower during training, but, because the model is sparser, it is faster during testing.

5 Conclusion

We describe an algorithm for selecting inducing points for Gaussian Process sparsification. It optimizes principled objective functions, and is applicable to discrete domains and non-differentiable kernels. On such problems it is shown to be as good as or better than competing methods and, for methods whose predictive behavior is similar, our method is several orders of magnitude faster. On continuous domains the method is competitive with state-of-art methods.

APPENDIX A
Incremental Cholesky QR Factorization

Although not directly used in our swap-update algorithm, the m-step incremental partial Cholesky and QR factorization algorithm to be presented in this section lays the ground for later on presenting all the details of the one-step updating, downdating and permuting algorithms used in swap-update algorithm.

For all the algorithms that follow, we frequently refer to columns or rows of K, but K never needs to be precomputed (taking up $O(n^2)$ time and storage). Instead it just needs to return its diagonal and specific column when queried (a function handle for example).

Both Algorithms 2 and 3 work with or without the augmentation trick introduced in Section 3.1. If σ is supplied, they work on the augmented factors, in which case the matrix L in the algorithms is the augmented version $\widetilde{L}$ of Section 3.1; and $L[1:n,:]$ is the non-augmented portion. Q,R are the QR factorization of L. The procedures also returns two vectors, $\mathbf{p}$ and $\mathbf{d}$. $\mathbf{p}$ is a permutation of $(1, 2, \ldots, n)$, and $\mathbf{d}$ stores the diagonal values of the residual matrix between the full K and current partial Cholesky factorization. In our application to kernel covariance matrix, $\mathbf{d}$ is also the point-wise variance that is not yet explained by (the factorization using) existing inducing points. See post-conditions after the algorithm for formal relationships among various quantities.

For ease of description, explicit row pivoting is not performed (consistent with the description in Section 3.1). Instead, the ordering of rows of $L[1:n,:]$ always stays in the original order of data points $(1, 2, \ldots, n)$, and we use $\mathbf{p}$ to keep track of the permutation, and index into the rows of $L[1:n,:]$. The columns are pivoted explicitly during the algorithm. In practical implementation however, we find the equivalent version with explicit row pivoting is slightly faster due to better memory/cache locality.

Assuming that the inducing set $\mathcal{I}_m = [i_1, .., i_k, .., i_m]$ is known, Algorithm 2 CholQR_mStep builds the factors incrementally.

Algorithm 2. m steps of incremental Cholesky and QR factorizations:

procedure CholQR_mStep($\mathcal{I}_m$, n, K, σ)
  $\mathbf{p} \leftarrow [1, 2, \ldots, n]$
  $\mathbf{d} \leftarrow \mathrm{diag}(K)$
  if σ is given **then** ▷Need to do the augmentation
    $L \leftarrow \mathrm{zeros}(n+m, m)$
    $Q \leftarrow \mathrm{zeros}(n+m, m)$
  else
    $L \leftarrow \mathrm{zeros}(n, m)$
    $Q \leftarrow \mathrm{zeros}(n, m)$
  end if
  $R \leftarrow \mathrm{zeros}(m, m)$
  for $k = 1 \rightarrow m$ **do**
    $t \leftarrow$ position of $\mathcal{I}_m[k]$ in $\mathbf{p}$
    $\mathbf{p}, L, Q, R, \mathbf{d} \leftarrow \mathrm{CholQR\_1}(t, k, n, K, \mathbf{p}, L, Q, R, \mathbf{d}, \sigma)$
  end for
  return p, L, Q, R, d
end procedure

Algorithm 3. one step of incremental Cholesky and QR factorizations:

procedure CholQR_1(t, k, n, K, $\mathbf{p}$, L, Q, R, $\mathbf{d}$, σ)
  $\mathbf{p}[t], \mathbf{p}[k] \leftarrow \mathbf{p}[k], \mathbf{p}[t]$ ▷ pivot the indices
  $L[\mathbf{p}[k], k] \leftarrow \sqrt{\mathbf{d}[\mathbf{p}[k]]}$
  $\mathbf{l}_{new} \leftarrow K[\mathbf{p}[(k+1):n], \mathbf{p}[k]]$
  $L[\mathbf{p}[(k+1):n], k] \leftarrow \frac{1}{L[\mathbf{p}[k],k]} * (\mathbf{l}_{new} - L[\mathbf{p}[(k+1):n], 1:(k-1)] * L[\mathbf{p}[k], 1:(k-1)]^\top)$
  $\mathbf{d}[\mathbf{p}[k:n]] \leftarrow \mathbf{d}[\mathbf{p}[k:n]] - (L[\mathbf{p}[k:n], k]).^{\hat{}}2$
  ▷ end of partial Cholesky part
  if σ is given **then** ▷ Need to do the augmentation
    $L[n+k, k] \leftarrow \sigma$
  end if
  ▷ start of QR part
  $R[1:(k-1), k] \leftarrow Q[:, 1:(k-1)]^\top * L[:, k]$
  $Q[:, k] \leftarrow L[:, k] - Q[:, 1:(k-1)] * R[1:(k-1), k]$
  $R[k, k] \leftarrow \|Q[:, k]\|$
  $Q[:, k] \leftarrow Q[:, k] / R[k, k]$
  return p, L, Q, R, d
end procedure

After the CholQR_mStep completes, the following post conditions hold true:

(1) $\mathbf{p}[1:m]$ has the same set of elements as $\mathcal{I}_m$;
(2) $L[\mathbf{p}, 1:m]$ is lower trapezoidal, and it is the rank - m partial Cholesky factor of $K[\mathbf{p}, \mathbf{p}]$;
(3) $L[\mathbf{p}[1:m], 1:m]$ is lower triangular, and it is the (complete) Cholesky factor of $K[\mathbf{p}[1:m], \mathbf{p}[1:m]]$;
(4) $\mathbf{d}[\mathbf{p}[1:m]] = 0$ and $\mathbf{d} = \mathrm{diag}(K - L[1:n, 1:m]\, L[1:n, 1:m]^\top)$
(5) if the augmentation trick is required by supplying σ, then $L[1:m, 1:m] = \sigma I_{m\times m}$, where $I_{m\times m}$ is the rank m identity matrix;
(6) with or without the augmentation, $L[:, 1:k] = Q[:, 1:k]R[1:k, 1:k] \quad \forall k \in \{1, \ldots, m\}$.

APPENDIX B

EFFICIENT PIVOT PERMUTATION AND REMOVAL

Given $k < m$, Algorithm 4 permutes pivot at position k to the right most column of L, Q, and R. Afterward, this pivot at the right most column would be removed by Algorithm 5. If (1)-(6) of the previous section hold as pre-conditions for CholQR_PermuteToRight, then they also hold as post-conditions. The subroutine qr22 used by Algorithm 4 simply computes the QR factorization of a 2 by 2 matrix.

Algorithm 4. Fast permuting a pivot to the right most position

procedure CHOLQR_PERMUTETORIGHT(k, m, n, **p**, L, Q, R, **d**, is_augmented)
  for $s = k \rightarrow (m-1)$ **do**
    $\mathbf{p}[s], \mathbf{p}[s+1] \leftarrow \mathbf{p}[s+1], \mathbf{p}[s]$ ▷ pivot the indices
    $Q1, R1 \leftarrow qr22(L[\mathbf{p}[s:(s+1)], s:(s+1)]^\top)$
    $L[\mathbf{p}[s:n], s:(s+1)] \leftarrow L[\mathbf{p}[s:n], s:(s+1)] * Q1$
    $L[\mathbf{p}[s], s+1] \leftarrow 0$
    $R[1:m, s:(s+1)] = R[1:m, s:(s+1)] * Q1$
    $Q2, R2 \leftarrow qr22(R[s:(s+1), s:(s+1)])$
    $R[s:(s+1), 1:m] \leftarrow Q2^\top * R[s:(s+1), 1:m]$
    $Q[:, s:(s+1)] \leftarrow Q[:, s:(s+1)] * Q2$
    $R[s+1, s] \leftarrow 0$
    if is_augmented **then**
      $Q[n+(s:(s+1)), 1:m] \leftarrow Q1^\top * Q[n+(s:(s+1)), 1:m]$
    end if
  end for
  return p, L, Q, R, **d**
end procedure

Algorithm 5. Remove pivot at the last position from the factors

procedure CHOLQR_REMOVELAST(m, n, **p**, L, Q, R, **d**)
  $\mathbf{d}[\mathbf{p}[m:n]] \leftarrow \mathbf{d}[\mathbf{p}[m:n]] + (L[\mathbf{p}[m:n], m]).^2$
  $L[:, m] \leftarrow 0$
  $Q[:, m] \leftarrow 0$
  $R[1:m, m] \leftarrow 0$
  $R[m, 1:m] \leftarrow 0$
  return L, Q, R, **d**
end procedure

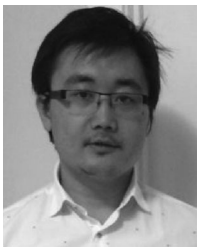
ACKNOWLEDGMENTS

This work has been supported by NSERC Canada, through grants to DJF, AH and a fellowship to MAB, and by the Canadian Institute for Advanced Research through the Program on Neural Computation and Adaptive Perception. We thank TaeHyung Kim for the advice on data collection for the BindingDB dataset. Y. Cao is the corresponding author.

REFERENCES

[1] F. R. Bach and M. I. Jordan, "Predictive low-rank decomposition for kernel methods," in *Proc. 22nd Int. Conf. Mach. Learn.*, 2005, pp. 33–40.
[2] F. R. Bach, "Consistency of trace norm minimization," *J. Mach. Learn. Res.*, vol. 9, pp. 1019–1048, Jun. 2008.
[3] L. Bo and C. Sminchisescu, "Greedy block coordinate descent for large scale Gaussian process regression," in *Proc. 24th Conf. Uncertainty Artif. Intell.*, 2008, pp. 43–52.
[4] L. Bo and C. Sminchisescu, "Twin Gaussian processes for structured prediction," *Int. J. Comput. Vis.*, vol. 87, pp. 28–52, 2010.
[5] A. J. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," in *Proc. 17th Int. Conf. Mach. Learn.*, 2000, pp. 911–918.
[6] K. Chalupka, C. K. I. Williams, and I. Murray, "A framework for evaluating approximation methods for Gaussian process regression," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 333–350, Feb. 2013.
[7] L. Csató and M. Opper, "Sparse on-line Gaussian processes," *Neural Comput.*, vol. 14, pp. 641–668, 2002.
[8] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2005, pp. 886–893.
[9] S. S. Keerthi and W. Chu, "A matching pursuit approach to sparse Gaussian process regression," in *Proc. Adv. Neural Inform. Process. Syst. 18*, 2006, pp. 643–650.
[10] N. D. Lawrence, M. Seeger, and R. Herbrich, "Fast sparse Gaussian process methods: The informative vector machine," in *Proc. Adv. Neural Inform. Process. Syst. 15*, 2003, pp. 609–616.
[11] J. J. Lee. (2008). Libpmk: A pyramid match toolkit. TR: MIT-CSAIL-TR-2008-17, MIT CSAIL [Online]. [Available: http://hdl.handle.net/1721.1/41070
[12] J. Quiñonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *J. Mach. Learn. Res.*, vol. 6, pp. 1939–1959, 2005.
[13] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning* (Adaptive Computation and Machine Learning). Cambridge, MA, USA: MIT Press, 2006.
[14] M. Seeger, C. K. I. Williams, and N. D. Lawrence, "Fast forward selection to speed up sparse Gaussian process regression," *Artif. Intell. Statist. 9*, 2003.
[15] A. J. Smola and P. Bartlett, "Sparse greedy Gaussian process regression," in *Proc. Adv. Neural Inform. Process. Syst. 13*, 2001, pp. 619–625.
[16] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," *Proc. Adv. Neural Inform. Process. Syst. 18*, 2006, pp. 1257–1264.
[17] M. K. Titsias, "Variational learning of inducing variables in sparse Gaussian processes," *J. Mach. Learn. Res.*, vol. 5, pp. 567–574, 2009.
[18] C. Walder, K. I. Kwang, and B. Schölkopf, "Sparse multiscale Gaussian process regression," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1112–1119.
[19] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proc. Int. Conf. Data Mining*, 2005, pp. 74–81.
[20] T. Gaertner, P. A. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Proc. 16th Annu. Conf. Comput. Learn. Theory 7th Kernel Workshop*, pp. 129–143, 2003.
[21] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proc. 20th Int. Conf. Mach. Learn.*, 2003, pp. 321–328.
[22] J. Ramon and T. Gaertner, "Expressivity versus efficiency of graph kernels," in *Proc. 1st Int. Workshop Mining Graphs, Trees Sequences*, 2003, pp. 65–74.

[23] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, 2011.
[24] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2009, pp. 488–495.
[25] S. V. N. Vishwanathan, N. N. Schraudolph, I. R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, pp. 1201–1242, 2010.



Yanshuai Cao received the BS degree in computer science and mathematics & statistics from the University of Toronto in 2010. He is currently working towards the PhD degree in the Department of Computer Science at the University of Toronto, under supervision of David Fleet and Aaron Hertzmann. His research interests include computer vision, machine learning, and statistics.

Marcus A. Brubaker received the PhD degree in computer science from the University of Toronto in 2011 and from 2011 to 2014, he was a postdoctoral fellow at TTI-Chicago. He is currently a postdoctoral fellow at the University of Toronto and consults with Cadre Research Labs. His research interests span statistics, machine learning and computer vision and includes applications in computational biology and forensics.

David J. Fleet received the PhD degree in computer science from the University of Toronto in 1991. He was on the faculty at Queen's University in Kingston from 1991 to 1998, and then an area manager and research scientist at the Palo Alto Research Center (PARC) from 1999 to 2003. In 2004, he joined the University of Toronto as a professor of computer science. His research interests include computer vision, image processing, visual perception, and visual neuroscience. He has published research articles, book chapters, and one book on various topics including the estimation of optical flow and stereoscopic disparity, probabilistic methods in motion analysis, modeling appearance in image sequences, motion perception and human stereopsis, hand tracking, human pose tracking, latent variable models, physics-based models for human motion analysis, and large-scale image retrieval. He received an Alfred P. Sloan Research Fellowship in 1996. He received paper awards at ICCV 1999, CVPR 2001, UIST 2003, and BMVC 2009. In 2010, he received the Koenderink Prize for his work with Michael Black and Hedvig Sidenbladh on human pose tracking. He has served as an area chair for numerous computer vision and machine learning conference. He was a program co-chair for CVPR 2003 and for ECCV 2014. He has been an associate editor and associate editor-in-chief for *IEEE Transactions on Pattern Analysis and Machine Intelligence*, and currently serves on the TPAMI Advisory Board. He is a senior fellow of the Canadian Institute for Advanced Research.

Aaron Hertzmann received the BA degree in computer science and art/art history from Rice University in 1996, and the PhD degree in computer science from New York University in 2001, respectively. He is a senior research scientist at Adobe Systems. He was a professor at the University of Toronto for 10 years, and has also worked at Pixar Animation Studios, University of Washington, Microsoft Research, Mitsubishi Electric Research Lab, Interval Research Corporation, and NEC Research. He is a fellow of the Canadian Institute for Advanced Research, and an associate editor for *ACM Transactions on Graphics*. His awards include the MIT TR100 in 2004, an Ontario Early Researcher Award in 2005, a Sloan Foundation Fellowship in 2006, a Microsoft New Faculty Fellowship in 2006, the CACS/AIC Outstanding Young CS Researcher Award in 2010, and the Steacie Prize for Natural Sciences in 2010.