Product Kernel Interpolation for Scalable Gaussian Processes

Jacob R. Gardner[1], **Geoff Pleiss**[1], **Ruihan Wu**[1,2], **Kilian Q. Weinberger**[1], **Andrew Gordon Wilson**[1]
[1]Cornell University, [2]Tsinghua University

Abstract

Recent work shows that inference for Gaussian processes can be performed efficiently using iterative methods that rely only on matrix-vector multiplications (MVMs). Structured Kernel Interpolation (SKI) exploits these techniques by deriving approximate kernels with very fast MVMs. Unfortunately, such strategies suffer badly from the curse of dimensionality. We develop a new technique for MVM based learning that exploits product kernel structure. We demonstrate that this technique is broadly applicable, resulting in *linear* rather than exponential runtime with dimension for SKI, as well as state-of-the-art asymptotic complexity for multi-task GPs.

1 INTRODUCTION

Gaussian processes (GPs) provide a powerful approach to regression and extrapolation, with applications as varied as time series analysis (Wilson and Adams, 2013; Duvenaud et al., 2013), blackbox optimization (Jones et al., 1998; Snoek et al., 2012), and personalized medicine and counterfactual prediction (Dürichen et al., 2015; Schulam and Saria, 2015; Herlands et al., 2016; Gardner et al., 2015). Historically, one of the key limitations of Gaussian process regression has been the computational intractability of inference when dealing with more than a few thousand data points. This complexity stems from the need to solve linear systems and compute log determinants involving an $n \times n$ symmetric positive definite *covariance matrix* K. This task is commonly performed by computing the Cholesky decomposition of K (Rasmussen and Williams, 2006), incurring $\mathcal{O}(n^3)$ complexity. To reduce this complexity, *inducing point methods* make use of a small set of $m < n$ points to form a rank m approximation of K (Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2006; Hensman et al., 2013; Titsias, 2009). Using the matrix inversion and determinant lemmas, inference can be performed in $\mathcal{O}(nm^2)$ time (Snelson and Ghahramani, 2006).

Recently, however, an alternative class of inference techniques for Gaussian processes have emerged based on iterative numerical linear algebra techniques (Wilson and Nickisch, 2015; Dong et al., 2017). Rather than explicitly decomposing the full covariance matrix, these methods leverage Krylov subspace methods (Golub and Van Loan, 2012) to perform linear solves and log determinants using only matrix-vector multiples (MVMs) with the covariance matrix. Letting $\mu(K)$ denote the time complexity of computing $K\mathbf{v}$ given a vector $\mathbf{v}$, these methods provide excellent approximations to linear solves and log determinants in $\mathcal{O}(r\mu(K))$ time, where r is typically some small constant Golub and Van Loan (2012).[1] This approach has led to scalable GP methods that differ radically from previous approaches – the goal shifts from computing efficient Cholesky decompositions to computing efficient MVMs. Structured kernel interpolation (SKI) (Wilson and Nickisch, 2015) is a recently proposed inducing point method that, given a regular grid of m inducing points, allows for MVMs to be performed in an impressive $\mathcal{O}(n + m \log m)$ time.

These MVM approaches have two fundamental drawbacks. First, Wilson and Nickisch (2015) use Kronecker factorizations for SKI to take advantage of fast MVMs, constraining the number of inducing points m to grow exponentially with the dimensionality of the inputs, limiting the applicability of SKI to problems with fewer than about 5 input dimensions. Second, the computational benefits of iterative MVM inference methods come at the cost of reduced modularity. If all we know about a kernel is that it decomposes as $K = K_1 \circ K_2$, it is not obvious how to efficiently perform MVMs with K, even if we have access to fast MVMs with both K_1 and K_2. In order for MVM infer-

Proceedings of the 21^{st} International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84.

[1] In practice, r depends on the conditioning of K, but is independent of n.

ence to be truly modular, we should be able to perform inference equipped with nothing but the ability to perform MVMs with K. One of the primary advantages of GPs is the ability to construct very expressive kernels by composing simpler ones (Rasmussen and Williams, 2006; Gönen and Alpaydın, 2011; Durrande et al., 2011; Duvenaud et al., 2013; Wilson, 2014). One of the most common kernel compositions is the element-wise product of kernels. This composition can encode different functional properties for each input dimension (e.g., Rasmussen and Williams, 2006; Gönen and Alpaydın, 2011; Duvenaud et al., 2013; Wilson, 2014), or express correlations between outputs in multi-task settings (MacKay, 1998; Bonilla et al., 2008; Álvarez and Lawrence, 2011). Moreover, the RBF and ARD kernels – arguably the most popular kernels in use – decompose into product kernels.

In this paper, we propose a single solution which addresses both of these limitations of iterative methods – improving modularity while simultaneously alleviating the curse of dimensionality. In particular:

1. We demonstrate that MVMs with product kernels can be approximated efficiently by computing the Lanczos decomposition of each component kernel. If MVMs with a kernel K can be performed in $\mathcal{O}(\mu(K))$ time, then MVMs with the element-wise product of d kernels can be approximated in $\mathcal{O}(dr\mu(K) + r^3 n \log d)$ time, where r is typically a very small constant.

2. Our fast product-kernel MVM algorithm, entitled *SKIP*, enables the use of structured kernel interpolation with product kernels without resorting to the exponential complexity of Kronecker products. SKIP can be applied even when the product kernels use different interpolation grids, and enables GP inference and learning in $\mathcal{O}(dn + dm \log m)$ for products of d kernels.

3. We apply SKIP to high-dimensional regression problems by expressing d-dimensional kernels as the product of d one-dimensional kernels. This formulation affords an *exponential improvement* over the standard SKI complexity of $\mathcal{O}(n + dm^d \log m)$, and achieving state of the art performance over popular inducing point methods (Hensman et al., 2013; Titsias, 2009).

4. We demonstrate that SKIP can reduce the complexity of multi-task GPs (MTGPs) to $\mathcal{O}(n + m \log m + s)$ for a problem with s tasks. We exploit this fast inference, developing a model that discovers cluster of tasks using Gibbs sampling.

5. We make our GPU implementations available as easy to use code as part of a new package for Gaussian processes, GPyTorch, available at `https://github.com/cornellius-gp/gpytorch`.

2 BACKGROUND

In this section, we provide a brief review of Gaussian process regression and an overview of iterative inference techniques for Gaussian processes based on matrix-vector multiplies.

2.1 Gaussian Processes

A Gaussian process generalizes multivariate normal distributions to distributions over functions that are specified by a prior *mean function* and a prior *covariance function* $f(\mathbf{x}) \sim \mathcal{GP}\left(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right)$. By definition, the function values of a GP at any finite set of inputs $[\mathbf{x}_1, ..., \mathbf{x}_n]$ are jointly Gaussian distributed:

$$\mathbf{f} = [f(\mathbf{x}_1), ..., f(\mathbf{x}_n)] \sim \mathcal{N}\left(\mu_X, K_{XX}\right)$$

where $\mu_X = [\mu(\mathbf{x}_1), ..., \mu(\mathbf{x}_n)]^\top$ and $K_{XX} = [k(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n$. Generally, K_{AB} denotes a matrix of cross-covariances between the sets A and B.

Under a Gaussian noise observation model, $p(y(\mathbf{x}) \mid f(\mathbf{x})) \sim \mathcal{N}(y(\mathbf{x}); f(\mathbf{x}), \sigma^2)$, the predictive distribution at $\mathbf{x}^*$ given data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is

$$p(f(\mathbf{x}^*) \mid \mathcal{D}) \sim \mathcal{GP}\left(\mu_{f|\mathcal{D}}(\mathbf{x}^*), k_{f|\mathcal{D}}(\mathbf{x}^*, \mathbf{x}^{*'})\right),$$

$$\mu_{f|\mathcal{D}}(\mathbf{x}) = \mu(\mathbf{x}^*) + K_{\mathbf{x}^* X} \hat{K}_{XX}^{-1} \mathbf{y}, \tag{1}$$

$$k_{f|\mathcal{D}}(\mathbf{x}^*, \mathbf{x}^*) = K_{\mathbf{x}^*\mathbf{x}^*} - K_{\mathbf{x}^* X} \hat{K}_{XX}^{-1} K_{\mathbf{x}^* X}^\top, \tag{2}$$

where $\hat{K}_{XX} = K_{XX} + \sigma^2 I$ and $\mathbf{y} = (y(\mathbf{x}_1), \ldots, y(\mathbf{x}_n))^\top$. All kernel matrices implicitly depend on hyperparameters θ. The log *marginal likelihood* of the data, conditioned only on these hyperparameters, is given by

$$\log p(\mathbf{y} \mid \theta) = -\frac{1}{2}\mathbf{y}^\top \hat{K}_{XX}^{-1}\mathbf{y} - \frac{1}{2}\log|\hat{K}_{XX}| + \mathrm{c}\,, \tag{3}$$

which provides a utility function for kernel learning.

2.2 Inference with matrix-vector multiplies

In order to compute the predictive mean in (1), the predictive covariance in (2), and the marginal log likelihood in (3), we need to perform linear solves (i.e. $[K_{XX} + \sigma^2 I]^{-1}\mathbf{v}$) and log determinants (i.e. $\log|K_{XX} + \sigma^2 I|$). Traditionally, these operations are achieved using the Cholesky decomposition of K_{XX} (Rasmussen and Williams, 2006). Computing this decomposition requires $\mathcal{O}(n^3)$ operations and storing the result requires $\mathcal{O}(n^2)$ space. Given the Cholesky decomposition, linear solves can be computed in $\mathcal{O}(n^2)$ time and log determinants in $\mathcal{O}(n)$ time.

There exist alternative approaches (e.g. Wilson and Nickisch, 2015) that require only matrix-vector multiplies (MVMs) with $[K_{XX} + \sigma^2 I]$. To compute linear solves, we use the method of *conjugate gradients*

(CG). This technique exploits that the solution to $A\mathbf{x} = \mathbf{b}$ is the unique minimizer of the quadratic function $\frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{x}^\top\mathbf{b}$, which can be found by iterating a simple three term recurrence. Each iteration requires a single MVM with the matrix A (Shewchuk et al., 1994). Letting $\mu(A)$ denote the time complexity of an MVM with A, p iterations of CG requires $O(p\mu(A))$ time. If A is $n \times n$, then CG is exact when $p = n$. However, the linear solve can often be approximated by $p < n$ iterations, since the magnitude of the residual $\mathbf{r} = A\mathbf{x} - \mathbf{b}$ often decays exponentially. In practice the value of k required for convergence to high precision is a small constant that depends on the conditioning of A rather than n (Golub and Van Loan, 2012). A similar technique known as *stochastic Lanczos quadrature* exists for approximating log determinants in $O(p\mu(A))$ time (Dong et al., 2017; Ubaru et al., 2017). In short, inference and learning for GP regression can be done in $\mathcal{O}(p\mu(K_{XX}))$ time using these iterative approaches.

Critically, if the kernel matrices admit fast MVMs – either through the structure of the data (Saatçi, 2012; Cunningham et al., 2008) or the structure of a general purpose kernel approximation (Wilson and Nickisch, 2015) – this iterative approach offers massive scalability gains over conventional Cholesky-based methods.

2.3 Structured kernel interpolation

Structured kernel interpolation (SKI) (Wilson and Nickisch, 2015) replaces a user-specified kernel $k(\mathbf{x}, \mathbf{x}')$ with an approximate kernel that affords very fast matrix-vector multiplies. Assume we are given a set of m *inducing points* U that we will use to approximate kernel values. Instead of computing kernel values between data points directly, SKI computes kernel values between inducing points and *interpolates* these kernel values to approximate the true data kernel values. This leads to the approximate SKI kernel:

$$k(\mathbf{x}, \mathbf{z}) \approx \mathbf{w}_\mathbf{x} K_{UU} \mathbf{w}_\mathbf{z}^\top, \tag{4}$$

where $\mathbf{w}_\mathbf{x}$ is a sparse vector that contains interpolation weights. For example, when using local cubic interpolation (Keys, 1981), $\mathbf{w}_x$ contains four nonzero elements. Applying this approximation for all data points in the training set, we see that:

$$K_{XX} \approx W_X K_{UU} W_X^\top \tag{5}$$

With arbitrary inducing points U, matrix-vector multiplies with $[WK_{UU}W^\top]\mathbf{v}$ require $\mathcal{O}(n + m^2)$ time. In one dimension, we can reduce this running time by instead choosing U to be a regular grid, which results in K_{UU} being *Toeplitz*. In higher dimensions, a multi-dimensional grid results in K_{UU} being the Kronecker product of Toeplitz matrices. This decompositions enables matrix-vector multiplies in at most $\mathcal{O}(n + m \log m)$ time, and $\mathcal{O}(n + m)$ storage. However, a Kronecker decomposition of K_{UU} leads to an exponential time complexity in d, the dimensionality of the inputs $\mathbf{x}$ (Wilson and Nickisch, 2015).

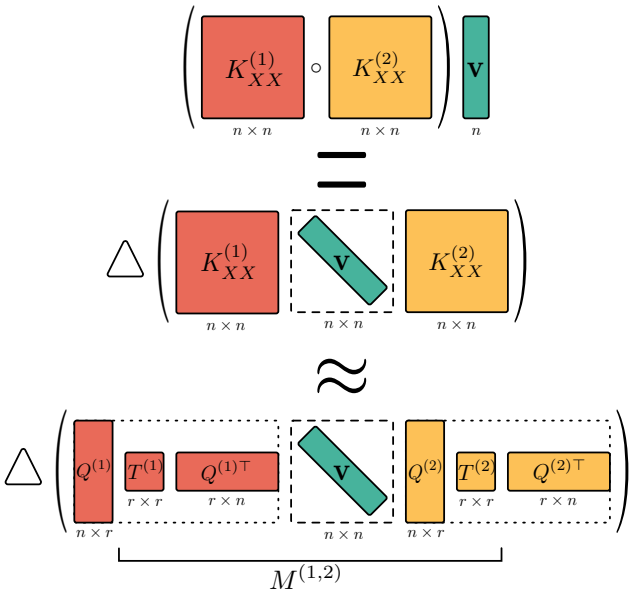

Figure 1: Computing fast matrix-vector multiplies (MVMs) with the product kernel $K_{XX}^{(1)} \circ K_{XX}^{(2)}$. **1:** Rewrite the element-wise product as the diagonal $\Delta(\cdot)$ of a product of matrices. **2:** Compute the rank-r Lanczos decomposition of $K_{XX}^{(1)}$ and $K_{XX}^{(2)}$.

3 MVMs WITH PRODUCT KERNELS

In this section we derive an approach to exploit product kernel structure for fast MVMs, towards alleviating the curse of dimensionality in SKI. Suppose a kernel separates as a product as follows:

$$k(\mathbf{x}, \mathbf{x}') = \prod_{i=1}^{d} k^{(i)}(\mathbf{x}, \mathbf{x}'). \tag{6}$$

Given a training data set $\mathbf{X} = [\mathbf{x}_1, \ldots \mathbf{x}_n]$, the kernel matrix K resulting from the product of kernels in (6) can be expressed as $K = K_{XX}^{(1)} \circ \cdots \circ K_{XX}^{(d)}$, where $\circ$ represents element-wise multiplication. In other words:

$$\left[K_{XX}^{(1)} \circ K_{XX}^{(2)}\right]_{ij} = \left[K_{XX}^{(1)}\right]_{ij} \left[K_{XX}^{(2)}\right]_{ij}. \tag{7}$$

The key limitation we must deal with is that, unlike a sum of matrices, vector multiplication does not distribute over the elementwise product:

$$\left(K^{(1)} \circ K^{(2)}\right)\mathbf{v} \neq \left(K^{(1)}\mathbf{v}\right) \circ \left(K^{(2)}\mathbf{v}\right). \tag{8}$$

We will assume we have access to fast MVMs for each component kernel matrix $K^{(i)}$. Without fast MVMs, there is a trivial solution to computing the elementwise matrix vector product: explicitly compute the kernel matrix K in $\mathcal{O}(dn^2)$ time and then compute $K\mathbf{v}$. We further assume that $K^{(i)}$ admits a low rank approximation, following prior work on inducing point methods following prior work on inducing point methods (Snelson and Ghahramani, 2006; Titsias, 2009; Wilson and Nickisch, 2015; Hensman et al., 2013).

A naive algorithm for a two-kernel product. We initially assume for simplicity that there are only $d = 2$ components kernels in the product. We will then show how to extend the two kernel case to arbitrarily sized product kernels. We seek to perform matrix vector multiplies:

$$(K_{XX}^{(1)} \circ K_{XX}^{(2)})\mathbf{v} \tag{9}$$

Eq. (9) may be expressed in terms of matrix-matrix multiplication using the following identity:

$$K\mathbf{v} = (K_{XX}^{(1)} \circ K_{XX}^{(2)})\mathbf{v} = \Delta(K_{XX}^{(1)} \, D_{\mathbf{v}} \, K_{XX}^{(2)\top}), \tag{10}$$

where $D_{\mathbf{v}}$ is a diagonal matrix whose elements are $\mathbf{v}$ (Figure 1), and $\Delta(M)$ denotes the diagonal of M. Because $D_{\mathbf{v}}$ is an $n \times n$ matrix, computing the entries of $K\mathbf{v}$ naively requires n matrix-vector multiplies with $K_{XX}^{(1)}$ and $K_{XX}^{(2)}$. The time complexity to compute (10) is therefore $\mathcal{O}(n\mu(K_{XX}^{(1)}) + n\mu(K_{XX}^{(2)}))$. This reformulation does not naively offer any time savings.

Exploiting low-rank structure. Suppose however that we have access to rank-r approximations of $K_{XX}^{(1)}$ and $K_{XX}^{(2)}$:

$$K_{XX}^{(1)} \approx Q^{(1)}T^{(1)}Q^{(1)\top}, \quad K_{XX}^{(2)} \approx Q^{(2)}T^{(2)}Q^{(2)\top},$$

where $Q^{(1)}$, $Q^{(2)}$ are $n \times r$ and $T^{(1)}$, $T^{(2)}$ are $r \times r$ (Figure 1). This rank decomposition makes the MVM significantly cheaper to compute. Plugging these decompositions in to (10), we derive:

$$K\mathbf{v} = \Delta\left(Q^{(1)}T^{(1)}Q^{(1)\top} \, D_{\mathbf{v}} \, Q^{(2)}T^{(2)}Q^{(2)\top}\right). \tag{11}$$

We prove the following key lemma in the supplementary materials about (11):

Lemma 3.1. *Suppose that $K_{XX}^{(1)} = Q^{(1)}T^{(1)}Q^{(1)\top}$ and $K_{XX}^{(2)} = Q^{(1)}T^{(1)}Q^{(1)\top}$, where $Q^{(1)}$ and $Q^{(2)}$ are $n \times r$ matrices and $T^{(1)}$ and $T^{(2)}$ are $r \times r$. Then $(K_{XX}^{(1)} \circ K_{XX}^{(2)})\mathbf{v}$ can be computed with (11) in $\mathcal{O}(r^2 n)$ time.*

Therefore, if we can efficiently compute low-rank decompositions of $K^{(1)}$ and $K^{(2)}$, then we immediately apply Lemma 3.1 to perform fast MVMs.

Computing low-rank structure. With Lemma 3.1, we have reduced the problem of computing MVMs with K to that of constructing low-rank decompositions for $K_{XX}^{(1)}$ and $K_{XX}^{(2)}$. Since we are assuming we can take fast MVMs with these kernel matrices, we now turn to the *Lanczos decomposition* (Lanczos, 1950; Paige, 1972). The Lanczos decomposition is an iterative algorithm that takes a symmetric matrix A and probe vector b and returns Q and T such that $A \approx QTQ^\top$, with Q orthogonal.

This decomposition is exact after n iterations. However, if we only compute $r < n$ columns of Q, then $Q_r T_r Q_r^\top$ is an effective low-rank approximation of A (Nickisch et al., 2009; Simon and Zha, 2000). Unlike standard low rank approximations (such as the singular value decomposition), the algorithm for computing the Lanczos decomposition $K_{XX}^{(i)} = Q^{(i)}T^{(i)}Q^{(i)\top}$ requires only r MVMs, leading to the following lemma:

Lemma 3.2. *Suppose that MVMs with $K_{XX}^{(i)}$ can be computed in $\mathcal{O}(\mu(K_{XX}^{(i)}))$ time. Then the rank-r Lanczos decomposition $K_{XX}^{(i)} \approx Q_r^{(i)}T_r^{(i)}Q_r^{(i)\top}$ can be computed in $\mathcal{O}(r\mu(K_{XX}^{(i)}))$ time.*

The above discussion motivates the following algorithm for computing $(K_{XX}^{(1)} \cdot K_{XX}^{(2)})\mathbf{v}$, which is summarized by Figure 1: First, compute the rank-r Lanczos decomposition of each matrix; then, apply (11). Lemmas 3.2 and 3.1 together imply that this takes $\mathcal{O}(r\mu(K_{XX}^{(1)}) + r\mu(K_{XX}^{(2)}) + r^2 n)$ time.

Extending to product kernels with three components. Now consider a kernel that decomposes as the product of three components, $k(\mathbf{x}, \mathbf{x}') = k^{(1)}(\mathbf{x}, \mathbf{x}')k^{(2)}(\mathbf{x}, \mathbf{x}')k^{(3)}(\mathbf{x}, \mathbf{x}')$. An MVM with this kernel is given by $K\mathbf{v} = (K_{XX}^{(1)} \circ K_{XX}^{(2)} \circ K_{XX}^{(3)})\mathbf{v}$. Define $\tilde{K}_{XX}^{(1)} = K_{XX}^{(1)} \circ K_{XX}^{(2)}$ and $\tilde{K}_{XX}^{(2)} = K_{XX}^{(3)}$. Then

$$K\mathbf{v} = (\tilde{K}_{XX}^{(1)} \circ \tilde{K}_{XX}^{(2)})\mathbf{v}\,, \tag{12}$$

reducing the three component problem back to two components. To compute the Lanczos decomposition of $\tilde{K}_{XX}^{(1)}$, we use the method described above for computing MVMs with $K_{XX}^{(1)} \circ K_{XX}^{(2)}$.

Extending to product kernels with many components. The approach for the three component setting leads naturally to a divide and conquer strategy. Given a kernel matrix $K = K_{XX}^{(1)} \circ \cdots \circ K_{XX}^{(d)}$ we define

$$\tilde{K}_{XX}^{(1)} = K_{XX}^{(1)} \circ \cdots \circ K_{XX}^{(\frac{d}{2})} \tag{13}$$

$$\tilde{K}_{XX}^{(2)} = K_{XX}^{(\frac{d}{2}+1)} \circ \cdots \circ K_{XX}^{(d)}, \tag{14}$$

which lets us rewrite $K = \tilde{K}_{XX}^{(1)} \circ \tilde{K}_{XX}^{(2)}$. By applying this splitting recursively, we can compute matrix-

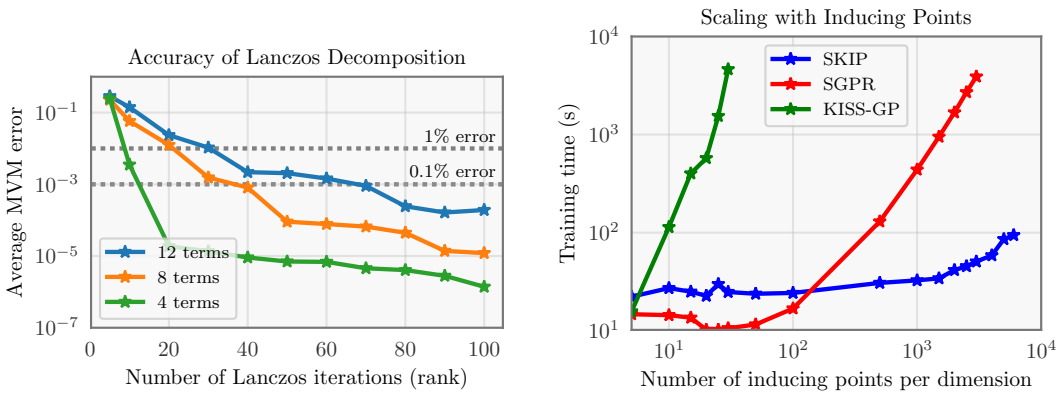

Figure 2: **Left:** Relative error of MVMs computed using SKIP compared to the exact value $K\mathbf{v}$. **Right:** Training time as a function of the number of inducing points *per dimension*. KISS-GP (SKI with Kronecker factorization) scales well with the *total* number of inducing points, but badly with number of inducing points *per dimension*, because the required total number of inducing points scales exponentially with the number of dimensions.

vector multiplies with K, leading to the following running time complexity:

Theorem 3.3. *Suppose that $K = K_{XX}^{(1)} \circ \cdots \circ K_{XX}^{(d)}$, and that computing a matrix-vector multiply with any $K_{XX}^{(i)}$ requires $\mathcal{O}(\mu(K_{XX}^{(1)}))$ operations. Computing an MVM with K requires $\mathcal{O}(dr\mu(K^{(i)}) + r^3 n \log d + r^2 n)$ time, where r is the rank of the Lanczos decomposition used.*

Sequential MVMs. If we are computing many MVMs with the same matrix, then we can further reduce this complexity by caching the Lanczos decomposition. The terms $\mathcal{O}(dr\mu(K_{XX}^{(i)}) + r^3 n \log d)$ represent the time to construct the Lanczos decomposition. However, note that this decomposition is not dependent on the vector that we wish to multiply with. Therefore, if we save the decomposition for future computation, we have the following corollary:

Corollary 3.4. *Any subsequent MVMs with K require $\mathcal{O}(r^2 n)$ time.*

If matrix-vector multiplications with $K_{XX}^{(i)}$ can be performed with significantly fewer than n^2 operations, this results in a significant complexity improvement over explicitly computing the full kernel matrix K.

3.1 Structured kernel interpolation for products (SKIP)

So far we have assumed access to fast MVMs with each constituent kernel matrix of an elementwise (Hadamard) product: $K = K_{XX}^{(1)} \circ \cdots \circ K_{XX}^{(d)}$. To achieve this, we apply the SKI approximation (Section 2.3) to each component:

$$K_{XX}^{(i)} = W^{(i)} K_{UU} W^{(i)\top}. \tag{15}$$

When using SKI approximations, the running time of our product kernel inference technique with p iterations of CG becomes $\mathcal{O}(dr(n + m \log m) + r^3 n \log d + pr^2 n)$. The running time of SKIP is compared to that of other inference techniques in Table 2.

4 MVM ACCURACY AND SCALABILITY

We first evaluate the accuracy of our proposed approach with product kernels in a controlled synthetic setting. We draw 2500 data points in d dimensions from $\mathcal{N}(0, I)$ and compute an RBF kernel matrix with lengthscale 1 over these data points. We evaluate the relative error of SKIP compared exact MVMs as a function of r – the number of Lanczos iterators. We perform this test for for 4, 8, and 12 dimensional data, resulting in a product kernel with 4, 8, and 12 components respectively. The results, averaged over 100 trials, are shown in Figure 2 (left). Even in the 12 dimensional setting, an extremely small value of r is sufficient to get very accurate MVMs: less than 1% error is achieved when $k = 30$. For a discussion of increasing error with dimensionality, see Section 7. In future experiments, we set the maximum number of Lanczos iterations to 100, but note that the convergence criteria is typically met far sooner. In the right side of Figure 2, we demonstrate the improved scaling of our method with the number of inducing points per dimension over KISS-GP. To do this, we use the $d = 4$ dimensional Power dataset from the UCI repository, and plot inference step time as a function of m. While our method clearly scales better with m than both KISS-GP and SGPR, we also note that because SKIP only applies the inducing point approximation to one-dimensional kernels, we anticipate ultimately needing significantly fewer inducing points than either SGPR or KISS-GP which need to cover the full d dimensional

space with inducing points.

5 APPLICATION 1: AN EXPONENTIAL IMPROVEMENT TO SKI

Wilson and Nickisch (2015) use a Kronecker decomposition of K_{UU} to apply SKI for $d > 1$ dimensions, which requires a fully connected multi-dimensional grid of inducing points U. Thus if we wish to have m distinct inducing point values for each dimension, the grid requires m^d inducing points – i.e. MVMs with the SKI approximate K_{XX} require $\mathcal{O}(n + dm^d \log m)$ time. It is therefore computationally infeasible to apply SKI with a Kronecker factorization, referred to in Wilson and Nickisch (2015) as KISS-GP, to more than about five dimensions. However, using the proposed SKIP method of Section 3, we can reduce the running time complexity of SKI in d dimensions from exponential $\mathcal{O}(n + dm^d \log m)$ to linear $\mathcal{O}(dn + dm \log m)$! If we express a d-dimensional kernel as the product of d one-dimensional kernels, then each component kernel requires only m grid points, rather than m^d. For the RBF and ARD kernels, decomposing the kernel in this way yields the same kernel function.

Datasets. We evaluate SKIP on six benchmark datasets. The precipitation dataset contains hourly rainfall measurements from hundreds of stations around the country. The remaining datasets are taken from the UCI machine learning dataset repository. KISS-GP (SKI with a Kronecker factorization) is not applicable when $d > 5$, and the full GP is not applicable on the four largest datasets.

Methods. We compare against the popular sparse variational Gaussian processes (SGPR) (Titsias, 2009; Hensman et al., 2013) implemented in GPflow (Matthews et al., 2017). We also compare to our GPU implementation of KISS-GP where possible, as well as our GPU implementation of the full GP on the two smallest datasets. All experiments were run on an NVIDIA Titan Xp. We evaluate SGPR using 200, 400 and 800 inducing points. All models use the RBF kernel and a constant prior mean function. We optimize hyperparameters with ADAM using default optimization parameters.

Discussion. The results of our experiments are shown in Table 1. On the two smallest datasets, the Full GP model outperforms all other methods in terms of speed. This is due to the overhead added by inducing point methods significantly outweighing simple solves with conjugate gradients with such little data. SKIP is able to match the error of the full GP model on elevators, and all methods have comparable error on the Pumadyn dataset.

On the precipitation dataset, inference with standard KISS-GP is still tractable due to the low dimensionality, and KISS-GP is both fast and accurate. Using SKIP results in higher error than KISS-GP, because we were able to use significantly fewer Lanczos iterations for our approximate MVMs than on other datasets due to the space complexity. We discuss the space complexity limitation further in the discussion section. Nevertheless, SKIP still performs better than SGPR. SGPR results with 400 and 800 inducing points are unavailable due to GPU memory constraints. On the remaining datasets, SKIP is able to achieve comparable or better overall error than SGPR, but with a significantly lower runtime.

6 APPLICATION 2: MULTI-TASK LEARNING

We demonstrate how the fast elementwise matrix vector products with SKIP can also be applied to accelerate multi-task Gaussian processes (MTGPs). Additionally, because SKIP provides cheap marginal likelihood computations, we extend standard MTGPs to construct an interpretable and robust multi-task GP model which discovers latent clusters among tasks using Gibbs' sampling. We apply this model to a particularly consequential child development dataset from the Gates foundation.

Motivating problem. The Gates foundation has collected an aggregate longitudinal dataset of child development, from studies performed around the world. We are interested in predicting the future development for a given child (as measured by weight) using a limited number of existing measurements. Children in the dataset have a varying number of measurements (ranging from 5 to 30), taken at irregular times throughout their development. We therefore model this problem with a multitask approach, where we treat each child's development as a task. This approach is the basis of several medical forecasting models (Alaa et al., 2017; Cheng et al., 2017; Xu et al., 2016).

Multi-task learning with GPs. The common multi-task setup involves s datasets corresponding to a set of different tasks, $\mathcal{D}_i : \left\{(\mathbf{x}_1^{(i)}, y_1^{(i)}), ..., (\mathbf{x}_{n_i}^{(i)}, y_{n_i}^{(i)})\right\}_{i=1}^{s}$. The multi-task Gaussian process (MTGP) of Bonilla et al. (2008) extends standard GP regression to share information between several related tasks. MTGPs assume that the covariance between data points factors as the product of kernels over (e.g. spatial or temporal) inputs and

Table 1: Comparison of SKIP and other methods on higher dimensional datasets. In this table, m is the *total* number of inducing points, rather than number of inducing points per dimension. (*We use $m = 100$ for SKIP on all datasets except precipitation, where we use $m = 120$K.)

Dataset	Metric	Full GP	SGPR ($m = 200$)	SGPR ($m = 400$)	SGPR ($m = 800$)	KISS-GP ($m = 120$K)	SKIP ($m = 100$)*
Pumadyn ($n = 8192$, $d = 32$)	**Test MAE**	**0.721**	0.766	0.766	0.766	–	0.766
	Train Time (s)	**4**	28	67	235	–	65
Elevators ($n = 16599$, $d = 18$)	**Test MAE**	**0.072**	0.157	0.157	0.157	–	**0.072**
	Train Time (s)	**12**	46	122	425	–	23
Precipitation ($n = 628474$, $d = 3$)	**Test MAE**	–	14.79	–	–	**9.81**	14.08
	Train Time (s)	–	1432	–	–	615	**34.16**
KEGG ($n = 48827$, $d = 22$)	**Test MAE**	–	0.101	0.093	0.087	–	**0.065**
	Train Time (s)	–	116	299	9926	–	**66**
Protein ($n = 45730$, $d = 9$)	**Test MAE**	–	7.219	4.97	4.72	–	**1.97**
	Train Time (s)	–	139	397	1296	–	**35**
Video ($n = 68784$, $d = 16$)	**Test MAE**	–	6.836	6.463	6.270	–	**5.621**
	Train Time (s)	–	113	334	1125	–	**57**

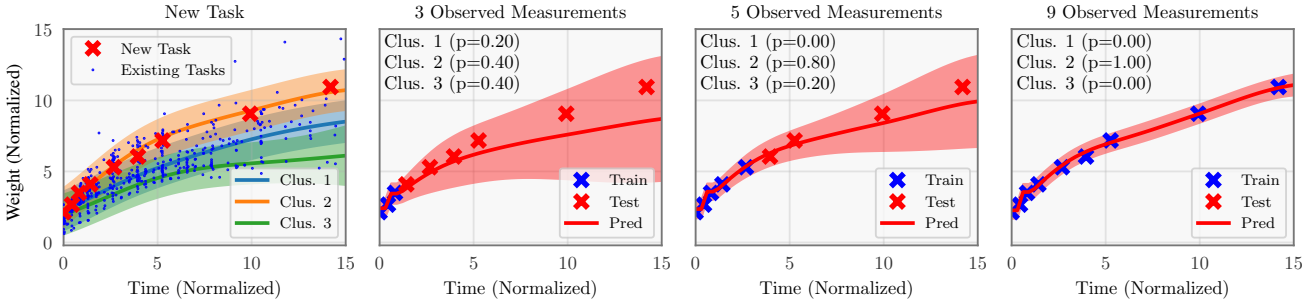

Figure 3: Applying the cluster-based MTGP model to new tasks.

Table 2: Asymptotic complexities of a single calculation of Equation 3 with n data points, m inducing points, r Lanczos iterations and p CG iterations. The first two rows correspond to an exact GP with Cholesky and CG.

Method	Complexity of 1 Inference Step
GP (Chol)	$\mathcal{O}(n^3)$
GP (MVM)	$\mathcal{O}(pn^2)$
SVGP	$\mathcal{O}(nm^2 + m^3 + dnm)$
KISS-GP	$\mathcal{O}(pn + pdm^d \log m)$
SKIP	$\mathcal{O}(drn + drm \log m + r^3 n \log d + pr^2 n)$

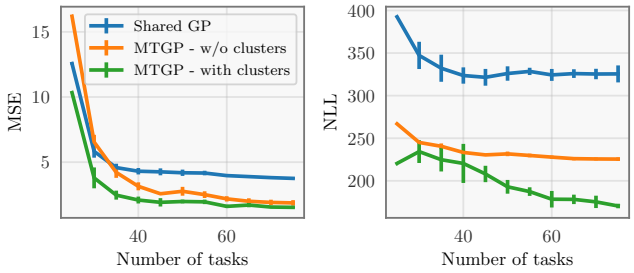

Figure 4: Predictive performance on childhood development dataset as a function of the number of tasks.

tasks. Specifically, given data points $\mathbf{x}$ and $\mathbf{x}'$ from tasks i and j, the MTGP kernel is given by

$$k((\mathbf{x}, i), (\mathbf{x}', j)) = k_{\text{input}}(\mathbf{x}, \mathbf{x}) k_{\text{task}}(i, j), \tag{16}$$

where k_{input} is a kernel over inputs, and $k_{\text{task}}(i, j)$ – the *coregionalization kernel* – is commonly parameterized by a low-rank covariance matrix $M = BB^\top \in \mathbb{R}^{s \times s}$ that encodes pairwise correlations between all pairs of tasks. The entries of B are learned by maximizing (3). We can express the covariance matrix K_{multi} for all n measurements as

$$K_{\text{multi}} = K_{XX}^{(\text{data})} \circ \left(VBB^\top V \right),$$

where V is an $n \times s$ matrix with one-hot rows: $V_{ij} = 1$ if the i^{th} observation belongs to task j. We can apply SKIP to multi-task problems by using a SKI approximation of $K^{(\text{data})}$ and computing its Lanczos decomposition. If B is rank-q, with $q < n$, then we do not need to decompose $VBB^\top V^\top$ since the matrix affords $\mathcal{O}(n + sq)$ MVMs.[2] For one-dimensional inputs, the time complexity of an MVM with K_{multi} is $\mathcal{O}(n + m \log m + sq)$ – a substantial improvement over standard inducing-point methods with MTGPs, which

[2] MVMs are $\mathcal{O}(n + sq)$ because V has $\mathcal{O}(n)$ nonzero elements and B is an $s \times q$ matrix.

typically require at least $\mathcal{O}(nm^2q)$ time (Bonilla et al., 2008; Álvarez and Lawrence, 2011). For $n = 4000$, SKIP speeds up marginal likelihood computations by a factor of 20.

Learning clusters of tasks. Motivated by the work of Rasmussen and Ghahramani (2002), Shi et al. (2005), Schulam and Saria (2015), Hensman et al. (2015), and Xu et al. (2016), we propose a modification to the standard MTGP framework. We hypothesize that similarities between tasks can be better expressed through c latent subpopulations, or clusters, rather than through pairwise associations. We place an independent uniform categorical prior over $\lambda_i \in [1, \dots, c]$, the cluster assignment for task i. Given measurements $\mathbf{x}_i, \mathbf{x}'_j$ for tasks i and j, we propose a kernel consisting of product and sum structure that captures cluster-level trends and individual-level trends:

$$k(\mathbf{x}_i, \mathbf{x}'_j) = k_{\text{cluster}}(\mathbf{x}, \mathbf{x}')\delta_{\lambda_i=\lambda_j} + k_{\text{indiv}}(\mathbf{x}, \mathbf{x}')\delta_{i=j}.$$

Here, k_{cluster} and k_{indiv} are both Matérn kernels ($\nu = \frac{5}{2}$) operating on $\mathbf{x}$, and the δ terms represent indicator functions. Both terms can be easily expressed as product kernels. We infer the posterior distribution of cluster assignments through Gibbs sampling. Given λ_{-i}, the cluster assignments for all tasks except the i^{th}, we sample an assignment for the i^{th} task from the marginal posterior distribution

$$p(\lambda_i|\mathbf{y}, \lambda_{-i}) \propto p(\mathbf{y} \mid \lambda_{-i}, \lambda_i = a\theta)p(\lambda_{-i}, \lambda_i = a)$$

Drawing a sample for the full vector λ requires $\mathcal{O}(cs)$ calculations of (3), an operation made relatively inexpensive by applying SKIP to the underlying model.

Results. We compare the cluster-based MTGP against two baselines: 1) a single-task GP baseline, which treats all available data as a single task, and 2) the standard MTGP. In Figure 4, we measure the extrapolation accuracy for 25 children as additional children (tasks) are added to the model. As the models are supplied with data from additional children, they are able to refine the extrapolations on all children. The predictions of the cluster model slightly outperform the standard MTGP, and significantly outperform the single-task model. Perhaps the key advantage of the clustering approach is interpretability: in Figure 3 (left), we see three distinct development types: above-average, average, and below average. We then demonstrate that as more data is observed when we apply the model to a new child with limited measurements, the model becomes increasingly certain that the child belongs to the above-average subpopulation.

7 DISCUSSION

It is our hope that this work highlights a question of foundational importance for scalable GP inference: *given the ability to compute* $A\mathbf{v}$ *and* $B\mathbf{v}$ *quickly for matrices* A *and* B*, how do we compute* $(A \circ B)\mathbf{v}$ *efficiently?* We have shown an answer to this question can *exponentially* improve the scalability and general applicability of MVM-based methods for fast Gaussian processes.

Stochastic diagonal estimation. Our method relies primarily on quickly computing the diagonal in Equation (10). Techniques exist for stochastic diagonal estimation (Fitzsimons et al., 2016; Hutchinson, 1990; Selig et al., 2012; Bekas et al., 2007). We found that these techniques converged slower than our method in practice, but they may be more appropriate for kernels with high rank structure.

Higher-order product kernels. A fundamental property of the Hadamard product is that $\text{rank}(A \circ B) \leq \text{rank}(A)\text{rank}(B)$ suggesting that we may need higher rank approximations with increasing dimension. In the limit, the SKI approximation $WK_{UU}W^\top$ can be used in place of the Lanczos decomposition in equation (10), resulting in an exact algorithm with $\mathcal{O}(dnm + dm^2 \log m)$ runtime: simply set $Q_k = W$ and MVMs require $\mathcal{O}(n)$ time instead of $\mathcal{O}(nk)$, and set $T_k = K_{UU}$ and MVMs now require $\mathcal{O}(m \log m)$ instead of $\mathcal{O}(k^2)$. This adaptation is rarely necessary, as the accuracy of MVMs with SKIP increases exponetially in k in practice.

Space complexity. To perform the matrix-vector multiplication algorithm described above, we must store the Lanczos decomposition of each component kernel matrix and intermediate matrices in the merge step for $\mathcal{O}(dkn)$ storage. This is better storage than the $\mathcal{O}(n^2)$ storage required in full GP regression, or $\mathcal{O}(nm)$ storage for standard inducing point methods, but worse than the linear storage requirements of SKI. In practice, we note that GPU memory is indeed often the major limitation of our method, as storing even $k = 20$ or $k = 30$ copies of a dataset in GPU memory can be expensive.

Acknowledgments

JRG, GP, and KQW are supported in part by grants from the National Science Foundation (III-1525919, IIS-1550179, IIS-1618134, S&AS 1724282, and CCF-1740822), the Office of Naval Research DOD (N00014-17-1-2175), and the Bill and Melinda Gates Foundation. AGW is supported by NSF IIS-1563887.

References

Alaa, A. M., Yoon, J., Hu, S., and van der Schaar, M. (2017). Personalized risk scoring for critical care prognosis using mixtures of gaussian processes. *IEEE Transactions on Biomedical Engineering*.

Álvarez, M. A. and Lawrence, N. D. (2011). Computationally efficient convolved multiple output Gaussian processes. *Journal of Machine Learning Research*, 12(May):1459–1500.

Bekas, C., Kokiopoulou, E., and Saad, Y. (2007). An estimator for the diagonal of a matrix. *Applied numerical mathematics*, 57(11):1214–1229.

Bonilla, E. V., Chai, K. M., and Williams, C. (2008). Multi-task Gaussian process prediction. In *NIPS*, pages 153–160.

Cheng, L.-F., Darnell, G., Chivers, C., Draugelis, M. E., Li, K., and Engelhardt, B. E. (2017). Sparse multi-output gaussian processes for medical time series prediction. *arXiv preprint arXiv:1703.09112*.

Cunningham, J. P., Shenoy, K. V., and Sahani, M. (2008). Fast gaussian process methods for point process intensity estimation. In *Proceedings of the 25th international conference on Machine learning*, pages 192–199. ACM.

Dong, K., Eriksson, D., Nickisch, H., Bindel, D., and Wilson, A. G. (2017). Scalable log determinants for gaussian process kernel learning. In *NIPS*.

Dürichen, R., Pimentel, M. A., Clifton, L., Schweikard, A., and Clifton, D. A. (2015). Multitask gaussian processes for multivariate physiological time-series analysis. *IEEE Transactions on Biomedical Engineering*, 62(1):314–322.

Durrande, N., Ginsbourger, D., and Roustant, O. (2011). Additive kernels for gaussian process modeling. *arXiv preprint arXiv:1103.4023*.

Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J. B., and Ghahramani, Z. (2013). Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1166–1174.

Fitzsimons, J. K., Osborne, M. A., Roberts, S. J., and Fitzsimons, J. F. (2016). Improved stochastic trace estimation using mutually unbiased bases. *arXiv preprint arXiv:1608.00117*.

Gardner, J. R., Song, X., Weinberger, K. Q., Barbour, D. L., and Cunningham, J. P. (2015). Psychophysical detection testing with bayesian active learning. In *UAI*, pages 286–295.

Golub, G. H. and Van Loan, C. F. (2012). *Matrix computations*, volume 3. JHU Press.

Gönen, M. and Alpaydın, E. (2011). Multiple kernel learning algorithms. *Journal of machine learning research*, 12(Jul):2211–2268.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*.

Hensman, J., Rattray, M., and Lawrence, N. D. (2015). Fast nonparametric clustering of structured time-series. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):383–393.

Herlands, W., Wilson, A., Nickisch, H., Flaxman, S., Neill, D., Van Panhuis, W., and Xing, E. (2016). Scalable gaussian processes for characterizing multidimensional change surfaces. In *Artificial Intelligence and Statistics*, pages 1013–1021.

Hutchinson, M. F. (1990). A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450.

Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492.

Keys, R. (1981). Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160.

Lanczos, C. (1950). *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA.

MacKay, D. J. (1998). Introduction to Gaussian processes. In Bishop, C. M., editor, *Neural Networks and Machine Learning*, chapter 11, pages 133–165. Springer-Verlag.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). Gpflow: A gaussian process library using tensorflow. *Journal of Machine Learning Research*, 18(40):1–6.

Nickisch, H., Pohmann, R., Schölkopf, B., and Seeger, M. (2009). Bayesian experimental design of magnetic resonance imaging sequences. In *NIPS*, pages 1441–1448.

Paige, C. C. (1972). Computational variants of the lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, 10(3):373–381.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959.

Rasmussen, C. E. and Ghahramani, Z. (2002). Infinite mixtures of gaussian process experts. In *NIPS*, pages 881–888.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*, volume 1. MIT press Cambridge.

Saatçi, Y. (2012). *Scalable inference for structured Gaussian process models*. PhD thesis, University of Cambridge.

Schulam, P. and Saria, S. (2015). A framework for individualizing predictions of disease trajectories by exploiting multi-resolution structure. In *NIPS*, pages 748–756.

Selig, M., Oppermann, N., and Enßlin, T. A. (2012). Improving stochastic estimates with inference methods: Calculating matrix diagonals. *Physical Review E*, 85(2):021134.

Shewchuk, J. R. et al. (1994). An introduction to the conjugate gradient method without the agonizing pain.

Shi, J. Q., Murray-Smith, R., and Titterington, D. (2005). Hierarchical gaussian process mixtures for regression. *Statistics and computing*, 15(1):31–41.

Simon, H. D. and Zha, H. (2000). Low-rank matrix approximation using the lanczos bidiagonalization process with applications. *SIAM Journal on Scientific Computing*, 21(6):2257–2274.

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In *NIPS*, pages 1257–1264.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

Titsias, M. K. (2009). Variational learning of inducing variables in sparse gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 567–574.

Ubaru, S., Chen, J., and Saad, Y. (2017). Fast estimation of tr (f (a)) via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099.

Wilson, A. and Adams, R. (2013). Gaussian process kernels for pattern discovery and extrapolation. In *ICML*, pages 1067–1075.

Wilson, A. G. (2014). Covariance kernels for fast automatic pattern discovery and extrapolation with gaussian processes. *University of Cambridge*.

Wilson, A. G. and Nickisch, H. (2015). Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *ICML*, pages 1775–1784.

Xu, Y., Xu, Y., and Saria, S. (2016). A bayesian nonparametric approach for estimating individualized treatment-response curves. In *Machine Learning for Healthcare Conference*, pages 282–300.

Supplementary Materials for: Product Kernel Interpolation for Scalable Gaussian Processes

Jacob R. Gardner[1], Geoff Pleiss[1], Ruihan Wu[1,2], Kilian Q. Weinberger[1], Andrew Gordon Wilson[1]
[1]Cornell University, [2]Tsinghua University

S1 Proof of Lemma 3.1

Letting $\mathbf{q}_i^{(1)}$ denote the i^{th} row of $Q^{(1)}$ and $\mathbf{q}_i^{(2)}$ denote the i^{th} row of $Q^{(2)}$, we can express the i^{th} entry $K\mathbf{v}$, $[K\mathbf{v}]_i$ as:

$$[K\mathbf{v}]_i = \mathbf{q}_i^{(1)} T^{(1)} Q^{(1)\top} \, D_{\mathbf{v}} \, Q^{(2)} T^{(2)} \mathbf{q}_i^{(2)\top}$$

To evaluate this for all i, we first once compute the $k \times k$ matrix:

$$M^{(1,2)} = T^{(1)} Q^{(1)\top} \, D_{\mathbf{v}} \, Q^{(2)} T^{(2)}.$$

This can be done in $O(nk^2)$ time. $T^{(1)}Q^{(1)\top}$ and $Q^{(2)}T^{(2)}$ can each be computed in $O(nk^2)$ time, as the Q matrices are $n \times k$ and the T matrices are $n \times k$. Multiplying one of the results by $D_{\mathbf{v}}$ takes $\mathcal{O}(nk)$ time as it is diagonal. Finally, multiplying the two resulting $n \times k$ matrices together takes $\mathcal{O}(nk^2)$ time.

After computing $M^{(1,2)}$, we can compute each element of the matrix-vector multiply as:

$$[K\mathbf{v}]_i = \mathbf{q}_i^{(1)} M^{(1,2)} \mathbf{q}_i^{(2)\top}.$$

Because $M^{(1,2)}$ is $k \times k$, each of these takes $\mathcal{O}(k)$ time to compute. Since there are n entries to evaluate in the MVM $K\mathbf{v}$ in total, the total time requirement after computing $M^{(1,2)}$ is $\mathcal{O}(kn)$ time. Thus, given low rank structure, we can compute $K\mathbf{v}$ in $\mathcal{O}(k^2 n)$ time total.

S2 Proof of Theorem 3.3

Given the Lanczos decompositions of $\tilde{K}^{(1)} = K_{XX}^{(1)} \circ \cdots \circ K_{XX}^{(a)}$ and $\tilde{K}^{(2)} = K_{XX}^{(a+1)} \circ \cdots \circ K_{XX}^{(d)}$, we can compute matrix-vector multiplies with $\tilde{K}^{(1)} \circ \tilde{K}^{(2)}$ in $\mathcal{O}(k^2 n)$ time each. This lets us compute the Lanczos decomposition of $\tilde{K}^{(1)} \circ \tilde{K}^{(2)}$ in $\mathcal{O}(k^3 n)$ time.

For clarity, suppose first that $d = 3$, i.e., $K = K_{XX}^{(1)} \circ K_{XX}^{(2)} \circ K_{XX}^{(3)}$. We first Lanczos decompose $K_{XX}^{(1)}$, $K_{XX}^{(2)}$ and $K_{XX}^{(3)}$. Assuming for simplicty that MVMs with each matrix take the same amount of time, This takes $\mathcal{O}(k\mu(K_{XX}^{(i)}))$ time total. We then use these Lanczos decompositions to compute matrix-vector multiples with $\tilde{K}_{XX}^{(1)}$ in $\mathcal{O}(k^2 n)$time each. This allows us to Lanczos decompose it in $\mathcal{O}(k^3 n)$ time total. We can then compute matrix-vector multiplications $K\mathbf{v}$ in $\mathcal{O}(k^2 n)$ time.

In the most general setting where $K = K_{XX}^{(1)} \circ \cdots \circ K_{XX}^{(d)}$, we first Lanczos decompose the d component matrices in $\mathcal{O}(dk\mu(K^{(i)}))$ and then perform $\mathcal{O}(\log d)$ merges as described above, each of which takes $\mathcal{O}(k^3 n)$ time. After computing all necessary Lanczos decompositions, matrix-vector multiplications with K can be performed in $\mathcal{O}(k^2 n)$ time.

As a result, a single matrix-vector multiply with K takes $\mathcal{O}(dk\mu(K^{(i)}) + k^3 n \log d + k^2 n)$ time. With the Lanczos decompositions precomputed, multiple MVMs in a row can be performed significantly faster. For example, running p iterations of conjugate gradients with K takes $\mathcal{O}(dk\mu(K^{(i)}) + k^3 n \log d + pk^2 n)$ time.