Large Linear Multi-output Gaussian Process Learning

Anonymous Authors[1]

Abstract

Gaussian process (GP) models, which put a distribution over arbitrary functions in a continuous domain, can be generalized to the multi-output case; a common way of doing this is to use a linear model of coregionalization (Alvarez et al., 2012). Such models can learn correlations across the multiple outputs, such as temperature data from disparate regions, which can then be exploited to share knowledge across the multiple outputs. While model learning can be performed efficiently for single-output GPs, the multi-output case still requires approximations for large n, the total number of observations across all outputs. In this work, we describe a new method, Large Linear GP (LLGP), which estimates covariance hyperparameters for multi-dimensional outputs and one-dimensional inputs. We show results of our approach applied to real time series data, and compare these results against related methods. Finally, we discuss extensions of our approach to multidimensional inputs.

1. Introduction

Gaussian processes (GPs) are a popular nonlinear regression method that innately capture function smoothness across inputs as defined by their covariance function (Williams & Rasmussen, 1996). GPs seamlessly extend to multi-output learning, picking up on latent cross-output processes, where in multi-output learning the objective is to build a probabilistic regression model over vector-valued observations.

Such an extension is especially important in time series data, where multi-output GPs have been applied to the problem of imputing missing temperature readings for sensors in different locations and reconstructing missing foreign exchange rates and commodity prices given rates and prices for other goods over time (Osborne et al., 2008; Alvarez et al., 2010).

Exact GP inference comes at a cost: to compute the gradients for kernel hyperparameters, an exact computation requires an $O(n^3)$ decomposition of an $O(n^2)$ matrix in memory for n samples (Williams & Rasmussen, 2006). Each hyperparameter's scalar partial derivative then requires another $O(n^2)$ computation. While these operations may be performed in parallel, the memory use and runtime is prohibitive for problems with a large number of outputs, since the number of hyperparameters grows accordingly. For this reason, an accurate approximate approach has been an important goal of machine learning research.

For a single output, variational approaches that work on $m \ll n$ inducing points, or locations at which the true kernel is sampled, letting users trade off accuracy for efficiency (Hensman et al., 2013).

Nguyen and Bonilla observed in Collaborative Multi-output Gaussian Processes (COGP) that multi-output GP algorithms can further share an internal representation of the covariance structure among all outputs at once (2014). We analogously exploit sharing in the covariance structure.

1.1. Contributions

Our approach makes the following contributions to approximate inference in multi-output GPs. First, we identify a block-Toeplitz structure induced by the linear model of coregionalization (LMC) kernel on a grid. Next, we adapt a previously identified kernel structure based on Semiparametric Latent Factor Models (SLFM) (Seeger et al., 2005). Both of these structures coordinate for fast matrix-vector multiplication $K\mathbf{y}$ with the covariance matrix K.

When inputs do not lie on a uniform grid (i.e., for time series, the length of time between observations differs), we apply a technique proposed in Massively Scalable Gaussian Processes (MSGP) (Wilson et al., 2015). For single-output kernels, MSGP leverages the structured kernel interpolation (SKI) framework to make inputs act

[1] Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

as if they are on a grid (Wilson & Nickisch, 2015). In a single-output case, this opens up the possibility for a matrix-free GP, avoiding construction of the explicit Gram covariance matrix. Because LMC kernels exhibit the previously-identified block-Toeplitz structure, they harmonize with SKI. While direct application of SKI is not possible, we can make an extension that makes matrix-free LMC kernel learning viable even on non-grid inputs.

Our paper is organized as follows. In Section 2 we give a background on single-output and multi-output GPs, as well as some history in structured linear algebra in GPs. Section 3 details both related work that was built upon in LLGP and existing methods for multi-output GPs. Section 4 describes our method, including a matrix-free heuristic stopping criterion. Then, in Section 5 we compare the performance of LLGP to existing methods and offer concluding remarks in Section 6.

2. Background

2.1. General Gaussian processes

First, we give an overview of the theoretical foundation for GPs. A GP is a set of random variables (RVs) $\{f_{\mathbf{x}}\}_{\mathbf{x}}$ indexed by $\mathbf{x} \in \mathcal{X}$, with the property that for any finite collection $X \subset \mathcal{X}$, the RVs are jointly Gaussian with a prescribed mean function $\boldsymbol{\mu} : \mathcal{X} \to \mathbb{R}$ and covariance $K : \mathcal{X}^2 \to \mathbb{R}$, i.e., $f_X \sim N(\boldsymbol{\mu}_X, K_{X,X})$ (Williams & Rasmussen, 1996). As usual, we drop the mean with $\mathbf{y} \triangleq f_X - \boldsymbol{\mu}_X$ and denote vectorization $f_X = (f_{\mathbf{x}})_{\mathbf{x}\in X}$ or matricization $K_{X,Z} = (K(\mathbf{x},\mathbf{z}))_{\mathbf{x}\in X,\mathbf{z}\in Z}$ with subscripts.

Given a set of n observations at each X of the $\mathbb{R}^n$-valued RV $\mathbf{y}$, inference at a single point $* \in \mathcal{X}$ of an $\mathbb{R}$-valued RV y_* is performed by marginalization (Williams & Rasmussen, 2006):

$$y_*|\mathbf{y} \sim N\left(K_{*,X}K_{X,X}^{-1}\mathbf{y}, K_{*,*} - K_{*,X}K_{X,X}^{-1}K_{X,*}\right).$$

Accuracy is often sensitive to a particular parameterization of our kernel, so model estimation is performed by maximizing data log likelihood with respect to parameters $\boldsymbol{\theta}$ of K:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \log p(\mathbf{f}_X|X,\boldsymbol{\theta}) &(1)\\ &= -\frac{1}{2}\mathbf{y}^\top K_{|\boldsymbol{\theta}}^{-1}\mathbf{y} - \frac{1}{2}\log\left|K_{|\boldsymbol{\theta}}\right| - \frac{n}{2}\log 2\pi. &(2)\end{aligned}$$

As such, the heart of GP learning lies in optimization of $\mathcal{L}$. Gradient-based optimization methods then require the gradient with respect to every parameter θ_j using $\boldsymbol{\alpha} = K_{|\boldsymbol{\theta}}^{-1}\mathbf{y}$:

$$\partial_{\theta_j}\mathcal{L} = \frac{1}{2}\boldsymbol{\alpha}^\top\partial_{\theta_j}K_{|\boldsymbol{\theta}}\boldsymbol{\alpha} - \frac{1}{2}\operatorname{tr}\left(K_{|\boldsymbol{\theta}}^{-1}\partial_{\theta_j}K_{|\boldsymbol{\theta}}\right). \quad (3)$$

2.2. Multi-output linear GPs

We build multi-output GP models as instances of general GPs, where a multi-output GP model identifies correlations between outputs through a shared input space (Alvarez et al., 2012).

Here, for D outputs, we write our indexing set as $\mathcal{X}' = [D]\times\mathcal{X}$: an input is in a point from a shared domain coupled with an output tag. Then, if we make observations at $X_d \subset \mathcal{X}$ for output $d \in [D]$, we can set:

$$\mathbf{X} = \{(d,x)\,|\,d\in[D], x\in X_d\} \subset \mathcal{X}';\quad n = |\mathbf{X}|\,.$$

An LMC kernel K is of the form

$$K([i,\mathbf{x}_i],[j,\mathbf{x}_j]) = \sum_{q=1}^{Q} b_{ij}^{(q)} k_q(\|\mathbf{x}_i - \mathbf{x}_j\|) + \epsilon_i\delta_{ij}, \quad (4)$$

where $k_q : \mathbb{R} \to \mathbb{R}$ is a stationary kernel on $\mathcal{X}$. Typically, the positive semi-definite (PSD) matrices $B_q \in \mathbb{R}^{D\times D}$ formed by $b_{ij}^{(q)}$ are parameterized as $A_qA_q^\top + \boldsymbol{\kappa}_q I_D$, with $A_q \in \mathbb{R}^{D\times R_q}, \boldsymbol{\kappa}_q \in \mathbb{R}_+^D$ and R_q a preset rank. Importantly, even though each k_q is stationary, K is only stationary on $\mathcal{X}'$ if B_q is Toeplitz. Settings of B_q that arise in practice, where we wish to capture covariance across outputs as an arbitrary D^2-dimensional latent process, prevent stationarity in K and therefore prohibit direct application of SKI.

2.3. Structured covariance matrices

If we can identify structure in the covariance matrices K, then we can recover fast in-memory representations and efficient matrix-vector multiplications (MVMs) for K.

The Kronecker product $A\otimes B$ of matrices of order a, b is a block matrix of order ab with ij-th block $A_{ij}B$. We can represent it by keeping representations of A and B separately, rather than the product. Then, the corresponding MVMs can be computed in time $O(a\text{MVM}(B) + b\text{MVM}(A))$, where $\text{MVM}(\cdot)$ is the runtime of a MVM. For GPs on uniform dimension-P grids, this approximately reduces the runtime and representation costs by a $(1/P)$-th power (Gilboa et al., 2015).

Symmetric Toeplitz matrices T are constant along their diagonal and fully determined by their top row $\{T_{1j}\}_j$, yielding an $O(n)$ representation. Such matrices arise naturally when we examine the covariance matrix induced by a stationary kernel k applied to a one-dimensional grid of inputs. Since the difference in adjacent inputs $t_{i+1}-t_i$ is the same for all i, we have the Toeplitz property that:

$$T_{(i+1)(j+1)} = k(|t_{i+1}-t_{j+1}|) = k(|t_i-t_j|) = T_{ij}$$

Furthermore, we can embed T in the upper-left corner of a circulant matrix C of twice its size, which enables MVMs $C\begin{pmatrix}\mathbf{x}\\ \mathbf{0}\end{pmatrix} = \begin{pmatrix}T\mathbf{x}\\ \mathbf{0}\end{pmatrix}$ in $O(n\log n)$ time. This approach has been used for fast inference in single-output GP time series with uniformly spaced inputs (Cunningham et al., 2008).

3. Related Work

3.1. Approximate inference methods

To cope with the lack of tractable GP inference methods, inducing point approaches create a tractable model to use instead of the classic GP. Such approaches fix or estimate inducing points $\mathbf{U}$ and claim that the data f_X is conditionally deterministic (deterministic training conditional, or DTC), independent (fully independent training conditional, or FITC), or partially independent (partially independent training conditional, or PITC) given RVs $f_{\mathbf{U}}$ (Quiñonero-Candela & Rasmussen, 2005). These approaches are agnostic to kernel stationarity, and their quality can be improved by increasing the number of inducing points at the cost of a longer runtime. Setting the inducing points $\mathbf{U} = \mathbf{X}$ recovers the original exact GP.

In the context of LMC, a framework for variational DTC, DTCVAR, provided a matrix-free method where each iteration has complexity $O(n(Dm)^2)$, where m is the average number of inducing points per output, $|\mathbf{U}|/D$ (Alvarez et al., 2010).

By sharing the m inducing points across all outputs, Collaborative Multi-output Gaussian Processes (COGP) improves runtime per iteration to $O(m^3)$. Moreover, COGP only requires $O(Dnm)$ memory (Nguyen et al., 2014). To ensure that $O(m^3)$ is computationally tractable for COGP, m must be sufficiently small. COGP makes large multi-output GP estimation feasible through a variational approximation, the evidence lower bound, to the log likelihood.

3.2. Approaches for stationary kernels

3.2.1. Structured Kernel Interpolation (SKI)

SKI abandons the inducing-point approach: instead of using an intrinsically sparse model, SKI approximates the original $K_{X,X}$ directly (Wilson & Nickisch, 2015). To do this efficiently, SKI relies on the differentiability of K. For $\mathbf{x}, \mathbf{z}$ within a grid U, $|U| = m$, and $W_{\mathbf{x},U} \in \mathbb{R}^{1\times m}$ as the cubic interpolation weights (Keys, 1981), $|K_{\mathbf{x},\mathbf{z}} - W_{\mathbf{x},U}K_{U,\mathbf{z}}| = O(m^{-3})$. The simultaneous interpolation $W \triangleq W_{X,U} \in \mathbb{R}^{n\times m}$ then yields the SKI approximation: $K_{X,X} \approx WK_{U,U}W^\top$. Importantly, W has only $4^d n$ nonzero entries, with $\mathcal{X} = \mathbb{R}^d$.

In order to adapt SKI to our context of multiple outputs, we build grid $\mathbf{U} \subset \mathcal{X}'$ out of a common subgrid $U \subset \mathcal{X}$ that extends to all outputs with $\mathbf{U} = [D] \times U$. Since the LMC kernel evaluated between two sets of outputs K_{X_i,X_j} is differentiable, as long as U contains each $\{X_d\}_{d\in[D]}$, the corresponding SKI approximation $K_{\mathbf{X},\mathbf{X}} \approx WK_{\mathbf{U},\mathbf{U}}W^\top$ holds with the same asymptotic convergence cubic in $1/m$.

Massively Scalable Gaussian Processes (MSGP) observes that the kernel $K_{U,U}$ on a grid exhibits Kronecker and Toeplitz matrix structure (Wilson et al., 2015). Drawing on previous work on structured GPs (Cunningham et al., 2008; Gilboa et al., 2015), MSGP uses linear conjugate gradient descent as a method for evaluating $K_{|\boldsymbol{\theta}}^{-1}\mathbf{y}$ efficiently for Eq. 1. In addition, (Wilson et al., 2014) mentions an efficient eigendecomposition that carries over to the SKI kernel for the remaining $\log\left|K_{|\boldsymbol{\theta}}\right|$ term in Eq. 1.

While evaluating $\log\left|K_{|\boldsymbol{\theta}}\right|$ is not feasible in the LMC setting (because the LMC sum breaks Kronecker and Toeplitz structure), the general notion of creating structure with SKI carries over to LLGP.

4. Matrix-free LMC Learning

We propose a linear model of coregionalization (LMC) method based on recent structure-based optimizations for GP estimation instead of variational approaches. Critically, the accuracy of the method need not be reduced by keeping m low because its reliance on structure allows better asymptotic performance. For simplicity, our work focuses on multi-dimensional outputs, one-dimensional inputs, and Gaussian likelihoods.

Although Eq. 1 does not have a hyperprior $\log p(\boldsymbol{\theta})$ term, it can easily be augmented with one.

4.1. Matrix-free GP Learning

Gibbs and MacKay describe the algorithm for GP model learning in terms of only MVMs with the covariance matrix (1996). In particular, they note that we can solve for $\boldsymbol{\alpha}$ satisfying $K_{|\boldsymbol{\theta}}\boldsymbol{\alpha} = \mathbf{y}$ in Eq. 3 using linear conjugate gradient descent (LCG). Moreover, they develop a stochastic approximation by introducing RV $\mathbf{r}$ with $\operatorname{cov}\mathbf{r} = I$:

$$\operatorname{tr}\left(K_{|\boldsymbol{\theta}}^{-1}\partial_{\theta_j}K_{|\boldsymbol{\theta}}\right) = \mathbb{E}\left[(K_{|\boldsymbol{\theta}}^{-1}\mathbf{r})^\top\partial_{\theta_j}K_{|\boldsymbol{\theta}}\mathbf{r}\right] \qquad (5)$$

For this approximation, the number of samples need not be large, and the estimate improves as the size of $K_{|\boldsymbol{\theta}}$ increases. As in other work (Cutajar et al., 2016), we let $\mathbf{r} \sim \mathrm{Unif}\{\pm 1\}$.

We depart from Gibbs and MacKay in selecting MINRES instead of LCG as the Krylov-subspace inversion method used to compute inverses from MVMs. MINRES handles numerically semidefinite matrices with more grace (Fong & Saunders, 2012). This is essential in GP optimization, where the diagonal noise matrix $\boldsymbol{\epsilon}$, iid for each output, shrinks over the course of learning, making inversion-based methods require additional iterations (Fig. 1).

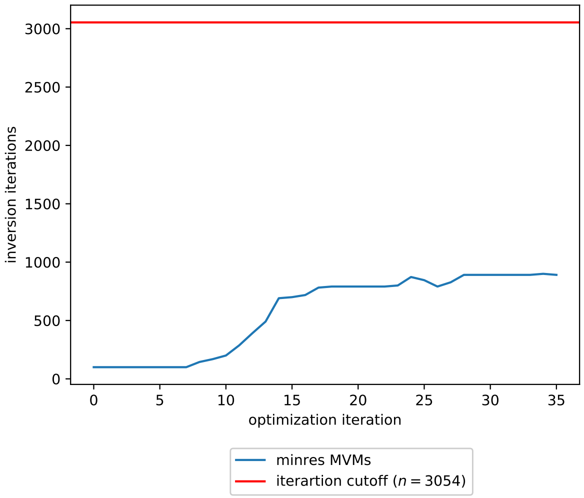

Figure 1: Number of MVMs that MINRES must perform at each optimization iteration for a GP applied to the dataset in Section 5.2. The iteration cutoff is the number of training points n in the dataset.

If we sample N_t instances of $\mathbf{r}$, every iteration needs to perform $N_t + 1$ inversions including $\mathbf{y}$, if we reuse the same samples $K_{|\boldsymbol{\theta}}^{-1}\mathbf{r}$ between derivatives ∂_{θ_j}. These inversions are done in parallel. Then, since $\partial_{\theta_j} K_{|\boldsymbol{\theta}}$ is both structured and smaller than $K_{|\boldsymbol{\theta}}$, each of the terms in the approximation Eq. 5 require $\tilde{O}(n)$ time to compute.

Critically, we cannot access $\mathcal{L}$ itself, only $\nabla\mathcal{L}$, so we choose AdaDelta as the high-level optimization routine (Zeiler, 2012). We considered several stochastic approximations for finding $\log\left|K_{|\boldsymbol{\theta}}\right|$ (Dorn & Enßlin, 2015; Han et al., 2015), but found these too slow and inaccurate for use in optimization.

In other words, all the work is done in the inversions $K_{|\boldsymbol{\theta}}^{-1}\mathbf{r}$, $K_{|\boldsymbol{\theta}}^{-1}\mathbf{y}$, which in turn rely on the MVM operation, discussed in Section 4.2. Since $K_{|\boldsymbol{\theta}}$ only enters computation as an MVM operator, the amount of memory consumed is dictated by its representation, which need not be dense.

4.2. Fast MVMs and parsimonious kernels

When LMC kernels are evaluated on a grid of points for each output, so $X_d = U$, the simultaneous covariance matrix equation without noise Eq. 6 over $\mathbf{U}$ holds for Toeplitz matrices K_q formed by the stationary kernels k_q evaluated at pairs of $U \times U$.

$$K_{\mathbf{U},\mathbf{U}} = \sum_q (A_q A_q^\top + \operatorname{diag} \boldsymbol{\kappa}_q) \otimes K_q \tag{6}$$

The SLFM and COGP models correspond to all $\boldsymbol{\kappa}_q$ set to 0 and $A_q = \mathbf{a}_q \in \mathbb{R}^{D\times 1}$. Moreover, we include D additional kernels for the independent GP components, which can be incorporated using kernels K_d where $A_d = 0$ and $\boldsymbol{\kappa}_d$ as the dth standard basis vector of $\mathbb{R}^D$ for $d \in [D]$. This shows a reduction from SLFM to LMC.

Importantly, the Kronecker structure of Eq. 6 lets us reuse the same grid K_q in a computational sense across the different outputs. Recalling our SKI extension to multiple outputs (Section 3.2.1), we build a corresponding approximation for the differentiable part of our kernel:

$$K_{\mathbf{X},\mathbf{X}} \approx W K_{\mathbf{U},\mathbf{U}} W^\top + \boldsymbol{\epsilon}. \tag{7}$$

We cannot fold $\boldsymbol{\epsilon}$ into the interpolated term $K_{\mathbf{U},\mathbf{U}}$ since it does not correspond to a differentiable kernel, so the SKI approximation fails. But this fact does not prevent efficient representation or multiplication since the matrix is diagonal. In particular, the MVM operation $K_{\mathbf{X},\mathbf{X}}\mathbf{x}$ can be approximated by $W K_{\mathbf{U},\mathbf{U}} W^\top \mathbf{x} + \boldsymbol{\epsilon}\mathbf{x}$, where matrix multiplication by the sparse matrices $\boldsymbol{\epsilon}, W, W^\top$ require $O(n)$ space and time.

We consider different representations of $K_{\mathbf{U},\mathbf{U}}$ from (Eq. 7) to reduce the memory and runtime overhead for performing the multiplication $K_{\mathbf{U},\mathbf{U}}\mathbf{z}$ (where we have computed $\mathbf{z} = W^\top \mathbf{x}$).

4.2.1. SUM: SUM REPRESENTATION

In SUM, we represent $K_{\mathbf{U},\mathbf{U}}$ with a Q-length list. At each index q, B_q is a dense matrix of order D and K_q is a Toeplitz matrix of order m, where only its top row needs to be represented to perform MVMs.

In turn, multiplication $K_{\mathbf{U},\mathbf{U}}\mathbf{z}$ is performed by multiplying each matrix in the list with $\mathbf{z}$ and summing the results, corresponding to . As described before, the Kronecker MVM $(B_q \otimes K_q)\mathbf{z}$ may be expressed as D fast Toeplitz MVMs with K_q and m dense MVMs with B_q. In turn, the runtime for each of the Q terms is $O(Dm \log m)$.

4.2.2. BT: BLOCK-TOEPLITZ REPRESENTATION

In BT, we notice that $K_{\mathbf{U},\mathbf{U}}$ is a block matrix with blocks T_{ij}:

$$\sum_q B_q \otimes K_q = \left(T_{ij}\right)_{i,j\in[D]^2}, \quad T_{ij} = \sum_q b_{ij}^{(q)} K_q.$$

On a one-dimensional grid U, these matrices are Toeplitz since they are linear combinations of Toeplitz matrices. BT requires D^2 m-sized rows to represent each T_{ij}. Then, using usual block matrix multiplication, an MVM $K_{\mathbf{U},\mathbf{U}}\mathbf{z}$ takes $O(D^2 m \log m)$ time.

4.2.3. SLFM: SLFM REPRESENTATION

SLFM uses a rank-based representation. Let $R \triangleq \sum_q {}^{R_q}/_{Q}$ be the average added rank, $R \le D$.

We first rewrite the grid kernel:

$$K_{U,U} = \sum_q \sum_{r=1}^{R_q} \mathbf{a}_q^{(r)} {\mathbf{a}_q^{(r)}}^\top \otimes K_q + \sum_q \operatorname{diag} \boldsymbol{\kappa}_q \otimes K_q.$$

Note $\mathbf{a}_q^{(r)} {\mathbf{a}_q^{(r)}}^\top$ is rank-1. Under some re-indexing $q' \in [RQ]$ which flattens the double sum such that each q' corresponds to a unique (r, q), the term $\sum_q \sum_{r=1}^{R_q} \mathbf{a}_q^{(r)} {\mathbf{a}_q^{(r)}}^\top \otimes K_q$ may be rewritten as

$$\sum_{q'} \mathbf{a}_{q'} \mathbf{a}_{q'}^\top \otimes K_{q'} = \mathbf{A} \operatorname{blockdiag}_{q'} \left(K_{q'}\right) \mathbf{A}^\top;$$

where the second simplification has $\mathbf{A} = \left(\mathbf{a}_{q'}\right)_{q'} \otimes I_m$ with $\left(\mathbf{a}_{q'}\right)_{q'}$ a matrix of horizontally stacked column vectors (Seeger et al., 2005). Next, we rearrange the remaining term $\sum_q \operatorname{diag} \boldsymbol{\kappa}_q \otimes K_q$ as $\operatorname{blockdiag}_d(T_d)$, where $T_d = \sum_q \kappa_{qd} K_q$ is Toeplitz.

Thus, the SLFM representation writes $K_{\mathbf{U},\mathbf{U}}$ as the sum of two block diagonal matrices of block order QR and D, where each block is a Toeplitz order m matrix, so MVMs take $O((QR + D) m \log m)$ time.

4.3. Asymptotic performance

Because the runtimes of BT and SLFM are complimentary in the sense that one performs better than the other when $D^2 > QR$ and vice-versa, an algorithm that uses the aforementioned condition between to decide between which representation to use can minimize runtime. We also found that SUM is efficient in practice for $Q = 1$.

Using the switching condition, MVMs with the original matrix $K_{\mathbf{X},\mathbf{X}}$ altogether have a space and time upper bound of $\tilde{O}(\min(QR, D^2) m + n)$, where the $\min$ is earned thanks to the choice between different representations. Every AdaDelta iteration then takes $\tilde{O}(\sqrt{\kappa_2})$ such matrix-vector products for machine-precision answers, with κ_2 the spectral condition number (Raykar & Duraiswami, 2007).

On a grid of inputs with $\mathbf{X} = \mathbf{U}$, the SKI interpolation vanishes with $W = I$. In this case, using BT alone leads to a faster algorithm—applying the Chan block-Toeplitz preconditioner in a Krylov-subspace based routine has experimentally shown convergence using fewer MVMs (Chan & Olkin, 1994).

4.4. Stopping conditions

For a gradient-only stopping heuristic, we maintain the running maximum gradient ∞-norm. If gradient norms drop below a preset proportion of the running maximum norm more than a pre-set tolerance number of times, we terminate. For example, when applied to the foreign exchange rate prediction (FX2007) dataset in Section 5.2, the heuristic eventually notices that we have slowed down increases in $\mathcal{L}$ because the gradients occasionally drop below the threshold at that point, while not displacing the solution $\boldsymbol{\theta}$ significantly since we must be near a local minimum (Fig. 2).

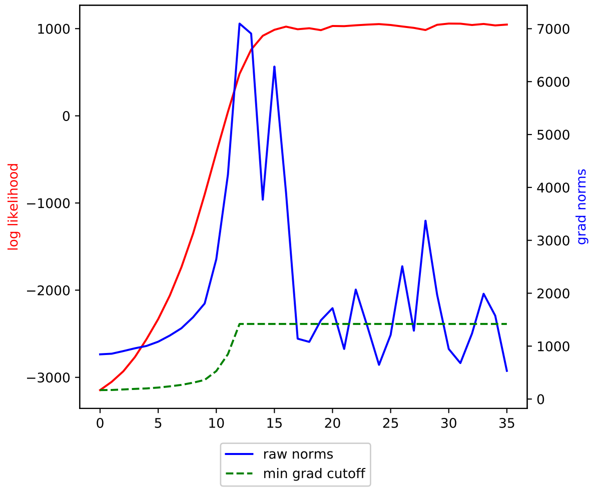

Figure 2: In green, we have 20% of the rolling maximum norm. In red and blue are $\mathcal{L}$ (computed exactly and therefore unavailable during benchmarks) and $\|\nabla\mathcal{L}\|_\infty$, respectively.

4.5. Prediction

The predictive mean can be computed in $O(1)$ time as observed in (Wilson et al., 2015) by $K_{*,X}\boldsymbol{\alpha} \approx W_{*,\mathbf{U}} K_{\mathbf{U},\mathbf{U}} \boldsymbol{\alpha}$.

The full predictive covariance estimate requires finding a new term $K_{*,X} K_{X,X}^{-1} K_{X,*}$. This is done by solving the linear system in a matrix-free manner on-the-fly; in particular, $K_{X,X}^{-1}K_{X,*}$ is computed via MINRES for every new test point $K_{X,*}$. Over several test points, this is embarrassingly parallelizable.

A more efficient predictive variance algorithm is outside the scope of this paper: for a research setting, we expect training time to be the bottleneck. Moreover, one can couple LLGP learning with other prediction mechanisms. One example is the sampling-based approach proposed in (Papandreou & Yuille, 2011), which extends to linear combinations of kernels that allow fast eigendecompositions.

4.6. Hyperparameter Settings

AdaDelta parameters were set to the following on all tests: rate $= 1$, decay $= 0.9$, momentum $= 0.5$, offset $= 0.0001$. The stopping criterion parameters permit the gradient ∞-norm to drop below a threshold of its maximum value so far a small, fixed number of times, 5. The maximum number of iterations was 100.

For learning, we initialize entries A_q according to a unit normal and all $\boldsymbol{\kappa}_q$ to 1. Note that COGP initializes the coregionalization matrix to 1 uniformly. Like COGP, we initialize noise to 0.1 for all outputs.

5. Results

We evaluate the methods on held out data by using standardized mean square error (SMSE) of the test points with the predicted mean, and the negative log predictive density (NLPD) of the Gaussian likelihood of the inferred model. Notably, NLPD takes confidence into account, while SMSE only evaluates the mean prediction. In both cases, lower values represent better performance.

5.1. Representation evaluation

We evaluated the performance of the different kernel representations over various rank and parameterization settings. In particular, we have the following parameters: n total sample size across all outputs, D number of outputs, Q number of kernels k_q, R average added rank, ϵ mean noise, and `ktype` kernel type. Kernel type is one of `mat, periodic, rbf, mix` corresponding to Matérn-3/2, standard periodic[1], and radial basis functions. `mix` refers to a mix of all three kernels.

Each kernel's inverse length scales and periods were selected by sampling uniformly in log space from 1 to 10 with Q samples. Next, we construct a random LMC kernel by sampling entries of each A_q from a standard normal distribution, $\boldsymbol{\kappa}_q$ from an inverse gamma with unit shape and scale, and independent noise $\boldsymbol{\epsilon}$ for every output from an inverse gamma with unit scale and mean ϵ. Inputs and outputs were independently generated from $\text{Unif}[0,1]$ for benchmarking.

As expected from their asymptotic runtime, SUM, BT, and SLFM representations are complimentary in MVM speed for different configurations of D, R, Q—this results in sparse inversion computation that consistently outperforms Cholesky decomposition in runtime (Tab. 1). For inverting systems, all computations were carried out until the residual ℓ_2 norm was at most 10^{-4}.

Table 1: The runtime in seconds for solving $K\mathbf{x} = \mathbf{y}$ for a random kernel K constructed as in Section 5.1 using MINRES for each of the kernel representations. For comparison, the CHOL representation is wallclock time to compute the Cholesky decomposition of the matrix, which must be constructed, and use this decomposition to invert the system. We averaged over five runs. In every run, we use $n = 5000$ simulated data points, `mix` kernels, and $\epsilon = 0.1$.

D	R	Q	CHOLESKY	SUM	BT	SLFM
2	2	10	40.45	40.04	**8.28**	45.07
10	1	10	37.60	34.51	21.93	**9.86**
10	10	1	9.59	**0.42**	2.42	0.90

We next evaluated the accuracy of the gradients for $N_t = 10$ trace samples. Fixing $R = 3, n = 5000, D = 10$, we quantified the accuracy and speed of computing $\nabla\mathcal{L}$. Since, for each partial derivative, LLGP requires only N_t n-sized vector dot products (Eq. 5), it generally runs faster than the exact approach (Fig. 4), which must compute a matrix-matrix dot product (Eq. 3). The gradients between the two, however, are virtually indistinguishable for smooth kernels that induce diagonally dominant covariance matrices (Fig. 3). Kernels such as the single Matérn or periodic kernel with noise on the order of 10^{-4} lead to less accurate gradients, owing to poor MINRES convergence in the inversions (Fig. 5). We will show that the stochastic gradients suffice for optimization in practical examples.

5.2. Foreign exchange rate prediction (FX2007)

We replicate the medium-sized dataset from COGP as an application to evaluate LLGP performance. The dataset consists of ten foreign currency exchange rates—CAD, EUR, JPY, GBP, CHF, AUD, HKD, NZD, KRW, and MXN—and three precious metals—XAU, XAG, and

[1] We define the periodic kernel as $k(r) = \exp\left(\frac{-\gamma}{2}\sin^2\frac{\pi r}{T}\right)$.

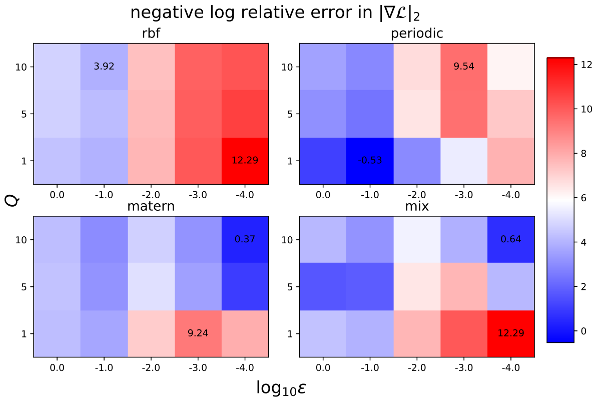

Figure 3: Negative logarithm of the relative error in ℓ_2 norm between exact and LLGP gradients. Higher is better, and extremal values are noted. For each data point, the average was taken over five runs.

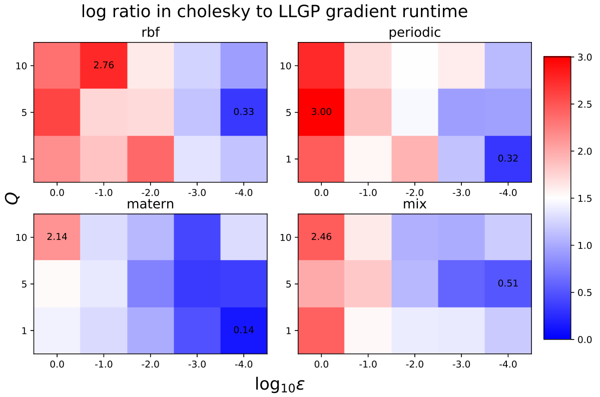

Figure 4: Logarithm of the ratio of time required by the exact approach to that of LLGP for computing the gradient over all parameters from Eq. 3. Higher is better, and extremal values are noted. For each data point, the average was taken over five runs.

XPT—implying that $D = 13$.[2] As in COGP, we retrieved the asset to USD rate, then used its reciprocal in all the results discussed below. The LLGP setting has $Q = 1, R = 2$, as recommended in (Alvarez et al., 2010) for LMC models on this dataset; let this be the LMC model on LLGP. COGP roughly corresponds to the the SLFM model, which has a total of 94 hyperparameters, compared to 53 for LLGP. All kernels are RBF.

The data used in this example are from 2007, and include $n = 3054$ training points and 150 test points. The test points include 50 contiguous points extracted from each of the CAD, JPY, and AUD exchanges.

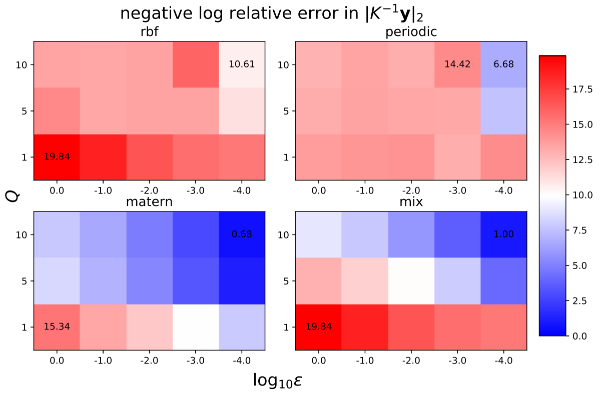

Figure 5: Negative logarithm of the relative error in ℓ_2 norm between exact and MINRES solutions to $K\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$. For each data point, the average was taken over five runs.

For this application, LLGP uses $m = n/D = 238$ interpolating points. We use the COGP settings from the paper.[3] LLGP, for both LMC, outperforms COGP in terms of predictive mean and variance estimation as well as runtime (Tab. 2).

Table 2: Average predictive performance and training time over 10 runs for LLGP and COGP on the FX2007 dataset. Parenthesized values are standard error. LLGP was run with LMC set to $Q = 1$, $R = 2$, and 238 interpolating points. COGP used a $Q = 2$ kernel with 100 inducing points.

METRIC	LLGP	COGP
SECONDS	**64** (**8**)	296 (2)
SMSE	**0.21** (**0.01**)	0.26 (0.03)
NLPD	**-3.62** (**0.07**)	14.52 (3.10)

5.3. Weather dataset

Next, we replicate results from a weather dataset, a large time series used to validate COGP. In this dataset, $D = 4$ weather sensors Bramblemet, Sotonmet, Cambermet, and Chimet record air temperature over five days in five minute intervals, with some dropped records due to equipment failure. Parts of Cambernet and Chimet are dropped for imputation, yielding $n = 15789$ training

[2] Data are from http://fx.sauder.ubc.ca/data.html

[3] COGP hyperparameters for FX2007 were 100 inducing points, 500 iterations, 200 mini-batch size.

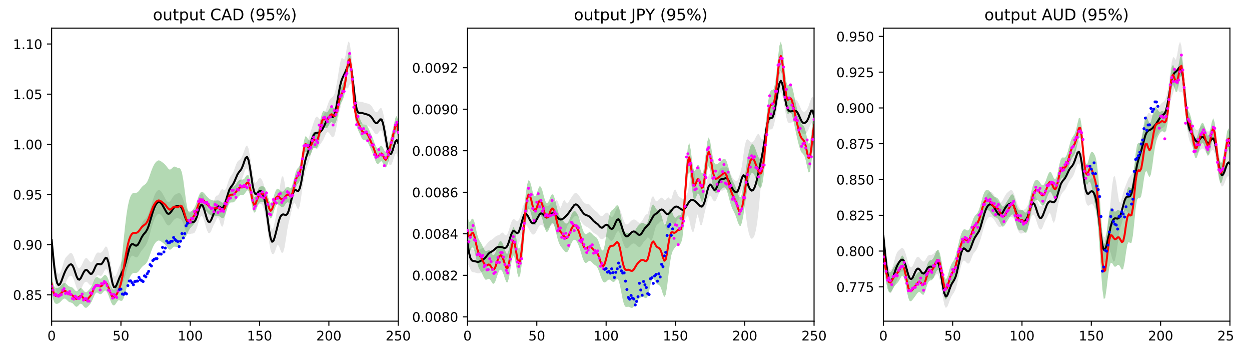

Figure 6: Test outputs for the FX2007 dataset. COGP mean is black, with 95% confidence intervals shaded in grey. LLGP mean is a solid red curve, with light green 95% confidence intervals. Magenta points are in the training set, while blue ones are in the test set. Notice LLGP variance corresponds to an appropriate level of uncertainty on the test set and certainty on the training set, as opposed to the uniform variance from COGP.

measurements and 374 test measurements.

We use the COGP parameters that were set by default in the code provided by the authors.[4] LLGP was run with the same parameters as in FX2007, simulating the SLFM model. We tested LLGP models on different numbers of interpolating points m.

Table 3: Average predictive performance and training time over 10 runs for LLGP and COGP on the weather dataset. Parenthesized values are standard error. Both LLGP and COGP trained the SLFM model. We show LLGP with 500 and 1000 interpolating points and COGP with 200 inducing points.

METRIC	LLGP $m = 500$	LLGP $m = 1000$	COGP
SECONDS	**60** (**14**)	259 (62)	1380 (12)
SMSE	0.09 (0.01)	0.09 (0.01)	**0.08** (**0.00**)
NLPD	2.14 (0.58)	**1.54** (**0.03**)	98.48 (1.30)

LLGP performed slightly worse than COGP in SMSE, but both NLPD and runtime indicate significant improvements (Tbl. 3). Varying the number of interpolating points m from 500 to 1000 constructs a tradeoff frontier between increases in m and NLPD decrease at the cost of additional runtime (Fig. 7). While NLPD improvement diminishes as m increases, LLGP is still an improvement compared to COGP for a range of m by an order of magnitude in runtime and almost two orders of magnitude for NLPD.

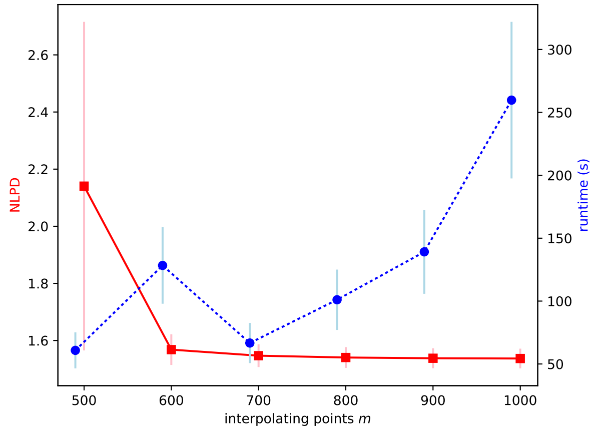

Figure 7: Average and standard error of NLPD and runtime of the SLFM model on LLGP across over varying interpolating points. Every setting was run 10 times.

6. Conclusion

LLGP recovers speedups from SKI (Wilson & Nickisch, 2015) for the problem of multi-output GP regression by recognizing structure unique to LMC kernels, and otherwise not necessarily recoverable in general multi-output kernels. This structure further enables a parsimonious representation that allows even large GPs to be learned without explicit construction of the covariance matrix.

LLGP provides a means to approximate the log-likelihood function gradients through interpolation. We have shown on several datasets that this can be done in a way that is faster and leads to more accurate results than variational approximations.

[4] https://github.com/trungngv/cogp, commit 3b07f621ff11838e89700cfb58d26ca39b119a35. The weather dataset was run on 1500 iterations, mini-batch size 1000.

Future work would extend the inputs to accept multiple dimensions. This can be done without losing internal structure in the kernel (Wilson et al., 2015): Toeplitz covariance matrices become block-Toeplitz with Toeplitz-blocks (BTTB). The cubic interpolation requires and exponential number of terms, so projection into lower dimensions learned in a supervised manner would be essential. Another useful line for investigation would be more informed stopping heuristics. Finally, an extension to non-Gaussian noise is also feasible in a matrix-free manner by following prior work (Cutajar et al., 2016).

References

Alvarez, Mauricio, Luengo, David, Titsias, Michalis, and Lawrence, Neil D. Efficient multioutput Gaussian processes through variational inducing kernels. In *AISTATS*, volume 9, pp. 25–32, 2010.

Alvarez, Mauricio, Rosasco, Lorenzo, Lawrence, Neil, et al. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3): 195–266, 2012.

Chan, Tony and Olkin, Julia. Circulant preconditioners for toeplitz-block matrices. *Numerical Algorithms*, 6 (1):89–101, 1994.

Cunningham, John, Shenoy, Krishna, and Sahani, Maneesh. Fast Gaussian process methods for point process intensity estimation. In *25th international conference on Machine learning*, pp. 192–199. ACM, 2008.

Cutajar, Kurt, Osborne, Michael, Cunningham, John, and Filippone, Maurizio. Preconditioning kernel matrices. In *ICML*, pp. 2529–2538, 2016.

Dorn, Sebastian and Enßlin, Torsten. Stochastic determination of matrix determinants. *Physical Review E*, 92 (1):013302, 2015.

Fong, David and Saunders, Michael. CG versus MINRES: an empirical comparison. *SQU Journal for Science*, 17(1):44–62, 2012.

Gibbs, Mark and MacKay, David. Efficient implementation of Gaussian processes, 1996.

Gilboa, Elad, Saatçi, Yunus, and Cunningham, John. Scaling multidimensional inference for structured Gaussian processes. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):424–436, 2015.

Han, Insu, Malioutov, Dmitry, and Shin, Jinwoo. Large-scale log-determinant computation through stochastic Chebyshev expansions. In *ICML*, pp. 908–917, 2015.

Hensman, James, Fusi, Nicolò, and Lawrence, Neil D. Gaussian processes for big data. In *The Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pp. 282–290. AUAI Press, 2013.

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

Nguyen, Trung, Bonilla, Edwin, et al. Collaborative multi-output Gaussian processes. In *UAI*, pp. 643–652, 2014.

Osborne, Michael, Roberts, Stephen, Rogers, Alex, Ramchurn, Sarvapali, and Jennings, Nicholas. Towards real-time information processing of sensor network data using computationally efficient multi-output Gaussian processes. In *7th international conference on Information processing in sensor networks*, pp. 109–120. IEEE Computer Society, 2008.

Papandreou, George and Yuille, Alan. Efficient variational inference in large-scale bayesian compressed sensing. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pp. 1332–1339. IEEE, 2011.

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

Raykar, Vikas and Duraiswami, Ramani. Fast large scale Gaussian process regression using approximate matrix-vector products. In *Learning workshop*, 2007.

Seeger, Matthias, Teh, Yee-Whye, and Jordan, Michael. Semiparametric latent factor models. Technical report, 2005.

Williams, Christopher and Rasmussen, Carl. Gaussian processes for regression. *Advances in neural information processing systems*, pp. 514–520, 1996.

Williams, Christopher and Rasmussen, Carl. Gaussian processes for machine learning. pp. 13–31. MIT Press, 2006.

Wilson, Andrew and Nickisch, Hannes. Kernel interpolation for scalable structured Gaussian processes (kiss-gp). In *The 32nd International Conference on Machine Learning*, pp. 1775–1784, 2015.

Wilson, Andrew, Gilboa, Elad, Cunningham, John, and Nehorai, Arye. Fast kernel learning for multidimensional pattern extrapolation. In *Advances in Neural Information Processing Systems*, pp. 3626–3634, 2014.

Wilson, Andrew, Dann, Christoph, and Nickisch, Hannes. Thoughts on massively scalable Gaussian processes. *arXiv preprint arXiv:1511.01870*, 2015.

Zeiler, Matthew. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.