GP-GRIEF: Scalable Gaussian Processes with Grid-Structured Eigenfunctions

Trefor W. Evans and Prasanth B. Nair

University of Toronto

NIPS

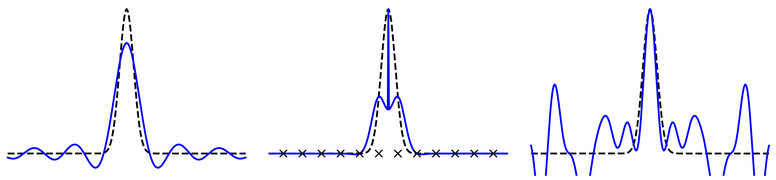
Overview

We introduce a kernel approximation strategy that enables Gaussian process training and inference in $\mathcal{O}(dnp)$ time and $\mathcal{O}(dn)$ storage for a d-dimensional dataset of size n. Our GRIEF (GRId-structured Eigen-Function) kernel consists of p eigenfunctions approximated on a dense Cartesian tensor product grid of inducing points. We show that by exploiting algebraic properties of Kronecker and Khatri-Rao tensor products, computational complexity of the training procedure can be *independent* of the number of inducing points, allowing us to use arbitrarily many to achieve a globally accurate kernel approximation. We benchmark our algorithms on real-world datasets with as many as two-million training points and up to $\mathbf{10^{32}}$ inducing points.

Eigenfunction Kernel

We approximate an exact kernel as a finite sum of eigenfunctions using a Nyström approximation from a set of inducing points [1]. This type of kernel representation is attractive since

- eigenfunctions give the most compact representation among orthogonal functions;
- our eigenfunctions live in a reproducing kernel Hilbert space, unlike some other kernel expansions whose bases have a pre-specified (e.g. trigonometric) form; and
- our approximate eigenfunctions converge in the limit of large n [2].



(a) **Eigenfunction Kernel** (b) FITC, x's show 12 inducing points (c) Random Fourier Features

Figure: Comparison of kernel approximations using $p = 12$ basis functions. Exact kernel shown in black.

We approximate an "exact" kernel k using p eigenfunctions to give the kernel $\widetilde{k}$:

$$\widetilde{k}(\mathbf{x},\mathbf{z}) = \sum_{i=1}^{p} \left(\frac{1}{\sqrt{\lambda_i}}\mathbf{K}_{\mathbf{x},\mathrm{U}}\mathbf{q}_i\right)\left(\frac{1}{\sqrt{\lambda_i}}\mathbf{K}_{\mathbf{z},\mathrm{U}}\mathbf{q}_i\right) = \mathbf{K}_{\mathbf{x},\mathrm{U}}\mathbf{Q}\mathbf{S}_p^T\mathbf{\Lambda}_p^{-1}\mathbf{S}_p\mathbf{Q}^T\mathbf{K}_{\mathrm{U},\mathbf{z}} \approx k(\mathbf{x},\mathbf{z}), \quad (1)$$

where $\mathbf{x},\mathbf{z} \in \mathbb{R}^d$ are d-dimensional inputs; $\mathrm{U} = \{\mathbf{u}_i\}_{i=1}^m$ refers to the set of m inducing point locations; $\mathbf{K}_{\_,\_}$ refers to a matrix of exact kernel evaluations between the two sets in the subscript; $\mathbf{\Lambda},\mathbf{Q} \in \mathbb{R}^{m\times m}$ are diagonal and unitary matrices containing the eigenvalues and eigenvectors of $\mathbf{K}_{\mathrm{U},\mathrm{U}}$, respectively; λ_i and $\mathbf{q}_i$ denote the ith largest eigenvalue and corresponding eigenvector of $\mathbf{K}_{\mathrm{U},\mathrm{U}}$, respectively; $\mathbf{S}_p \in \mathbb{R}^{p\times m}$ is a sparse selection matrix where $\mathbf{S}_p(i,:)$ contains one value set to unity in the column corresponding to the index of the ith largest value on the diagonal of $\mathbf{\Lambda}$; and we use the shorthand notation $\mathbf{\Lambda}_p = \mathbf{S}_p\mathbf{\Lambda}\mathbf{S}_p^T \in \mathbb{R}^{p\times p}$ to denote a diagonal matrix containing the p largest eigenvalues of $\mathbf{K}_{\mathrm{U},\mathrm{U}}$, sorted in descending order.

We write the covariance matrix on a training set with inputs $\mathrm{X} = \{\mathbf{x}_i\}_{i=1}^n$ as

$$\widetilde{\mathbf{K}}_{\mathrm{X},\mathrm{X}} = \mathbf{K}_{\mathrm{X},\mathrm{U}}\mathbf{Q}\mathbf{S}_p^T\mathbf{\Lambda}_p^{-1}\mathbf{S}_p\mathbf{Q}^T\mathbf{K}_{\mathrm{U},\mathrm{X}} \approx \mathbf{K}_{\mathrm{X},\mathrm{X}}. \quad (2)$$

The quality of this kernel approximation depends on the quantity and distribution of inducing points which we discuss next.

Gridded Inducing Points

Quantity and distribution of inducing points is crucial for an accurate kernel approximation. We place inducing points on a Cartesian grid to fill out the input space while allowing many more inducing points then training points ($m \gg n$). The grid contains $\overline{m} = \sqrt[d]{m} \approx \mathcal{O}(10)$ points along each dimension. Our covariance matrix then inherits the Kronecker product ($\otimes$) structure $\mathbf{K}_{\mathrm{U},\mathrm{U}} = \bigotimes_{i=1}^d \mathbf{K}_{\mathrm{U},\mathrm{U}}^{(i)}$, enabling efficient Kronecker matrix algebra to be exploited [3]. For instance,

$\mathbf{K}_{\mathrm{U},\mathrm{U}} = \bigotimes_{i=1}^d \mathbf{K}_{\mathrm{U},\mathrm{U}}^{(i)}$ storage $\rightarrow \mathcal{O}(d\overline{m}^2)$

$\mathbf{K}_{\mathrm{U},\mathrm{U}} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ factoring $\rightarrow \mathcal{O}(d\overline{m}^3)$

$\mathbf{Q} = \bigotimes_{i=1}^d \mathbf{Q}^{(i)}$ MVM $\rightarrow \mathcal{O}(d\overline{m}^{d+1})$

Exponential Scaling

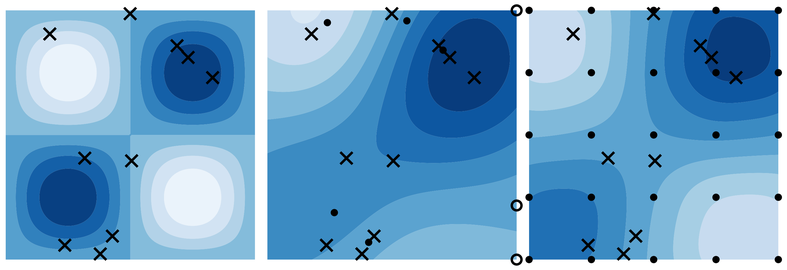
In low-dimensions, exploiting gridded inducing point structure can be greatly advantageous, however, we can immediately see in the block to the left that complexity of MVMs with $\widetilde{\mathbf{K}}_{\mathrm{X},\mathrm{X}}$ increases *exponentially* in d! MVMs also require storing a length $\overline{m}^d$ vector so memory requirements also scale exponentially. This poor scaling poses a serious impediment to the successful application of the proposed approach, or SKI [3], to high-dimensional datasets. We next discuss how to overcome this computational bottleneck.

Linear Scaling

Here, we show how to massively decrease time and storage requirements from exponential to *linear* in d by identifying further matrix structure. From $\widetilde{\mathbf{K}}_{\mathrm{X},\mathrm{X}}$ in eq. (2), we find $\mathbf{K}_{\mathrm{X},\mathrm{U}}$ admits a row-partitioned Khatri-Rao product ($*$) structure

$$\mathbf{K}_{\mathrm{X},\mathrm{U}} = \bigast_{i=1}^{d} \mathbf{K}_{\mathrm{X},\mathrm{U}}^{(i)} = \begin{pmatrix} \mathbf{K}_{\mathrm{X},\mathrm{U}}^{(1)}(1,:)\otimes\mathbf{K}_{\mathrm{X},\mathrm{U}}^{(2)}(1,:)\otimes\cdots\otimes\mathbf{K}_{\mathrm{X},\mathrm{U}}^{(d)}(1,:) \\ \mathbf{K}_{\mathrm{X},\mathrm{U}}^{(1)}(2,:)\otimes\mathbf{K}_{\mathrm{X},\mathrm{U}}^{(2)}(2,:)\otimes\cdots\otimes\mathbf{K}_{\mathrm{X},\mathrm{U}}^{(d)}(2,:) \\ \vdots \quad\quad\quad \vdots \quad\quad\quad \ddots \quad\quad\quad \vdots \\ \mathbf{K}_{\mathrm{X},\mathrm{U}}^{(1)}(n,:)\otimes\mathbf{K}_{\mathrm{X},\mathrm{U}}^{(2)}(n,:)\otimes\cdots\otimes\mathbf{K}_{\mathrm{X},\mathrm{U}}^{(d)}(n,:) \end{pmatrix}, \quad (3)$$

Next, we observe that $\mathbf{K}_{\mathrm{X},\mathrm{U}}\mathbf{Q} = \bigast_{i=1}^{d} \mathbf{K}_{\mathrm{X},\mathrm{U}}^{(i)}\mathbf{Q}^{(i)}$ is also a row-partitioned Khatri-Rao product matrix, and that $\mathbf{S}_p^T$ can be written as a column-partitioned Khatri-Rao product matrix. It can then be shown that a matrix-vector product with $(\mathbf{K}_{\mathrm{X},\mathrm{U}}\mathbf{Q})\mathbf{S}_p^T$ (a matrix product of row- and column-partitioned Khatri-Rao matrices) can be made in $\mathcal{O}(dnp)$ time and using no more than $\mathcal{O}(n)$ additional memory using algorithm `mvKRrowcol`. We can then train our GP-GRIEF model in $\mathcal{O}(dnp)$ time using a conjugate gradient solver. Also, since $\mathbf{K}_{\mathrm{X},\mathrm{U}}^{(i)}$ are only of size $n \times \overline{m}$, our storage requirements have decreased to $\mathcal{O}(dn\overline{m}) \approx \mathcal{O}(dn)$.



(a) Test Data. $f(x,y) = \sin(x)\sin(y)$ (b) FITC $m = 8$. RMSE $= 0.47$ (c) GP-GRIEF m=25, p=4. RMSE $= 0.34$

Figure: Regression comparison of FITC vs GP-GRIEF with a squared-exponential kernel. GP-GRIEF (with $p = 4$) uses only half the basis functions as FITC (with $m = 8$), however, achieves much better generalization on a test set. In fact, GP-GRIEF matches the test error of an exact GP. Crosses denote the $n = 10$ training point positions whose responses are corrupted with $\mathcal{N}(0, 0.1)$ noise. Dots denote inducing point locations within bounds and circles show the direction of those outside bounds.

Algorithm `mvKRrowcol`

Computes the tensor product $\mathbf{RCb}$ where $\mathbf{R} \in \mathbb{R}^{n\times m}$, $\mathbf{C} \in \mathbb{R}^{m\times p}$ are Khatri-Rao products of row- and column-partitioned matrices, respectively. Requires $\mathcal{O}(dnp)$ time if we assume that one of $\mathbf{R}$ or $\mathbf{C}$ are dense and the other is sparse with one non-zero per row. $\circ$ is the Hadamard product.

Output: $\mathbf{f} = \mathbf{RCb} \in \mathbb{R}^n$
for $j = 1$ **to** n **do**
  $\mathbf{t} = \mathbf{R}^{(1)}(j,:)\mathbf{C}^{(1)}$
  for $i = 2$ **to** d **do**
    $\mathbf{t} = \mathbf{t} \circ \mathbf{R}^{(i)}(j,:)\mathbf{C}^{(i)}$
  end for
  $\mathbf{f}(j) = \mathbf{t}\mathbf{b}$
end for

Covariance Reconstruction

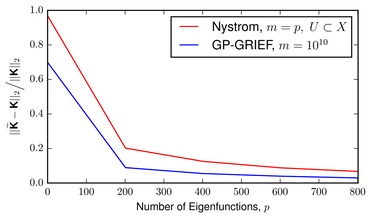


Figure: Covariance matrix reconstruction error of GP-GRIEF compared to the "Nyström method" of Williams and Seeger [4] that uses $m = p$ inducing points randomly sampled from the training set. We use $n = 10000$ randomly distributed training points in 10-dimensional space and a squared-exponential kernel.

UCI Regression Datasets

We present early results on large UCI regression datasets. Observe that:

- Complexity independence on m enables use of 10^{32} inducing points on *Pumadyn*.
- GP-GRIEF takes just one hour to train on the two-million point dataset *Electric*.
- Just p=100 basis functions yield a very high quality model on *Electric*.
- We demonstrate linear scaling with respect to the number of eigenfunctions, p.
- GP-GRIEF shows test errors comparable to [5]. On *Electric* it does much better.

			GP-GRIEF				Yang et al. [5]
Dataset	n	d	p	$m = \overline{m}^d$	Time (mins)	RMSE	RMSE
Pumadyn	8192	32	100	$\mathbf{10^{32}}$	1.6	0.21 ± 0.00	0.20 ± 0.00
			1000	$\mathbf{10^{32}}$	9.3	0.20 ± 0.00	
Elevators	16599	18	100	$\mathbf{5^{18}}$	0.7	0.097 ± 0.001	0.090 ± 0.001
			100	$\mathbf{10^{18}}$	0.8	0.096 ± 0.001	
			1000	$\mathbf{10^{18}}$	6	0.092 ± 0.002	
			5000	$\mathbf{10^{18}}$	30.9	0.091 ± 0.001	
Protein	45730	9	100	$\mathbf{10^{9}}$	0.9	0.63 ± 0.01	0.53 ± 0.01
			5000	$\mathbf{10^{9}}$	34.2	0.58 ± 0.01	
Electric	**2049280**	11	100	$\mathbf{10^{11}}$	65.6	0.068 ± 0.002	0.120 ± 0.120

Table: Mean and standard deviation of test error and average training time (including hyperparmater estimation) from 10-fold cross validation on UCI regression datasets using a squared-exponential ARD (SE-ARD) kernel. We compare our results with Yang et al. [5] who use the same train test splits and approximates an SE-ARD kernel using Fastfood finite basis function expansions. m is the number of inducing points used and p is the number of eigenfunctions used.

Acknowledgements: Work funded by an NSERC Discovery Grant and the Canada Research Chairs program.

[1] H. Peng and Y. Qi. "EigenGP: Gaussian Process Models with Adaptive Eigenfunctions." In: *International Joint Conference on Artificial Intelligence*. 2015, pp. 3763–3769.

[2] C. T. H. Baker. *The numerical treatment of integral equations*. Oxford: Clarendon press, 1977.

[3] A. G. Wilson and H. Nickisch. "Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP)". In: *Proceedings of The 32nd International Conference on Machine Learning*. 2015, pp. 1775–1784.

[4] C. K. I. Williams and M. Seeger. "Using the Nyström method to speed up kernel machines". In: *Advances in Neural Information Processing Systems*. 2001, pp. 682–688.

[5] Z. Yang, A. J. Smola, L. Song, and A. G. Wilson. "À la Carte – Learning Fast Kernels". In: *Artificial Intelligence and Statistics*. 2015, pp. 1098–1106.