Tucker Gaussian Process for Regression and Collaborative Filtering

Hyunjik Kim, Xiaoyu Lu, Seth Flaxman, Yee Whye Teh
Department of Statistics
University of Oxford
{hkim,xiaoyu.lu,flaxman,y.w.teh}@stats.ox.ac.uk

Abstract

We introduce the Tucker Gaussian Process (TGP), an approach to scalable GP learning based on low-rank tensor decompositions. We show that our model is applicable to general regression problems, and is particularly well-suited to grid-structured data and problems where the dependence on covariates is close to being separable. Furthermore, when applied to collaborative filtering, our model provides an effective GP based method that has a low-rank matrix factorisation at its core, and gives a natural and elegant method for incorporating side information.

1 Introduction

Gaussian processes (GPs) are a popular class of Bayesian nonparametric priors over functions [Rasmussen and Williams, 2005], and have been used across a range of machine learning tasks, e.g. classification and regression, dimensionality reduction [Lawrence, 2004], and collaborative filtering [Lawrence and Urtasun, 2009, Yu et al., 2006]. The computational cost of naïvely learning GPs is $O(N^3)$ where $N$ is the data set size; at its core GP learning involves the solution of a linear system of equations of size $N$. In recent years a range of techniques have been developed for scaling up learning in GPs. One class of techniques uses random feature expansions [Rahimi and Recht, 2007, Lázaro-Gredilla et al., 2010], which are based on the “weight space view” of GPs, i.e. that a GP simply amounts to (nonparametric) Bayesian linear regression with input vectors replaced by (infinite-dimensional) feature vectors. One can use Bochner’s theorem to form finite-dimensional Monte Carlo approximations of the feature vectors [Rahimi and Recht, 2007], or can learn these approximations [Lázaro-Gredilla et al., 2010]. A second class constitutes the Nyström and inducing point techniques [Seeger et al., 2003, Shielsen and Ohahramani, 2003, Titsias, 2009], which are based on the more common “function space view” of GPs. These approximate a GP by evaluating it only on $n \ll N$ inducing points and, assuming that these inducing points capture all correlations in the GP, approximate the kernel matrix with a low-rank matrix. Yet another class makes use of specialised kernel structures to efficiently solve the linear system of equations. Examples include Kronecker structure when the inputs lie on a grid [Saatci, 2011], and Toeplitz structure [Storkey, 1999] when the grid is regularly spaced and the kernel is stationary.

In this paper, we propose a novel scalable approximation scheme for GPs (Section 2). Our starting point is the Kronecker structure that arises naturally when working with kernels that are products of simpler constituent kernels (say each dependent on one covariate dimension). Coupled with the weight space view of GPs, we can represent a draw from the GP as a product between a weight tensor and a collection of feature vectors (one for each constituent kernel). The weight tensor can be very large for high dimensional problems, and our proposal is to approximate it using a low-rank Tucker decomposition [Tucker, 1966] instead. This reduces the effective number of parameters that need to be learnt, and acts as a regulariser towards simple regression functions that are linear combinations of separable functions (Section 2.4). Our approach can take advantage of the Kronecker structure even when the data do not lie on a grid, by using random feature maps (Section 2.2).
We also show an elegant application of our method to the problem of collaborative filtering (Section 3). A natural and direct approach to collaborative filtering, particularly with side information, is to model the rating that user $u_i$ gives to item $v_j$ as a function $f(u_i, v_j)$, where $u_i$ and $v_j$ can include both the identities of the user/item as well as relevant side information. Using a GP prior on $f$, the kernel can be designed to capture similarities among users/items and how ratings depend on side information. However a naïve implementation of this approach is not scalable, and most of the literature on collaborative filtering has instead been based on low-rank matrix factorisation [Rennie and Srebro 2005; Salakhutdinov and Mnih 2008a,b]. We show that our approach naturally leads to a representation of the GP function using a low-rank matrix factorisation, and incorporates dependence on side information neatly into the framework.

2 Tucker Gaussian Process Regression

2.1 Tucker GP Regression

Consider a regression problem with inputs $x_1, \ldots, x_N \in \mathcal{X}$ and corresponding observations $y_1, \ldots, y_N \in \mathbb{R}$. We assume $y_i | x_i \sim \mathcal{N}(f(x_i), \sigma^2)$ for some $f : \mathcal{X} \to \mathbb{R}$ and that the observations are independent. The aim is to learn $f$. One approach is to put a Gaussian Process (GP) prior on $f$, with zero mean and covariance $k$. The training then consists of computing the posterior GP. The problem is that training costs $O(N^3)$ operations, making inference infeasible for large $N$.

The weight space view of GPs offers a natural way of dealing with the problem: suppose there exists a feature map $\phi : \mathcal{X} \to \mathbb{R}^n$ (where $n$ is the number of features) such that $k(x, x') = \phi(x)\top \phi(x') \forall x, x' \in \mathcal{X}$. Then the GP is equivalent to Bayesian Linear Regression with feature vectors used for each row of the design matrix [Rasmussen and Williams 2005]:

$$y|x \overset{iid}{\sim} \mathcal{N}(f(x), \sigma^2) \quad f(x) = \theta\top \phi(x) \quad \theta \sim \mathcal{N}(0, I), \theta \in \mathbb{R}^n \quad (1)$$

Now training takes $O(Nn^2)$ time, and is scalable for $n \ll N$.

Consider the case of product kernels, where the kernel can be written as follows:

$$k(x_i, x_j) = \prod_{d=1}^D k_d(x_i, x_j) \quad (2)$$

and suppose there are feature maps $\phi_d : \mathcal{X} \to \mathbb{R}^n$ such that $k_d(x_i, x_j) = \phi_d(x_i)\top \phi_d(x_j)$. Then we can write $k(x_i, x_j) = \phi(x_i)\top \phi(x_j)$ where $\phi(x) = \otimes_{d=1}^D \phi_d(x)$ is the Kronecker product of the $\phi_d$. Returning to (1),

$$f(x) = \theta\top \phi(x) = \theta\top (\otimes_{d=1}^D \phi_d(x)) = \theta \times_{d=1}^D \phi_d(x) \quad (3)$$

where $\theta$ has been reshaped as a $D$-dimensional tensor in $\mathbb{R}^{n \times \cdots \times n}$ in the rightmost expression as in Figure 1. We refer to this as the full-rank model, and use $\theta$ as a tensor for the rest of the paper.

\[1\theta \times_{d=1}^D \phi_d := \text{vec}(\theta)\top \otimes_{d=1}^D \phi_d = \sum_{i_1, \ldots, i_D = 1}^n \theta_{i_1, \ldots, i_D} \prod_{d=1}^D (\phi_d)_{i_d}.\]
This full-rank model is problematic in high dimensions: the size of $\theta$ grows as $n^D$, so the function computation becomes infeasible. Thus we introduce the novel Tucker Gaussian Process (TGP) model, where we circumvent this problem by approximating $\theta$ using a low-rank Tucker decomposition [Tucker 1966]. This is a tensor-matrix product between a low rank core tensor $W \in \mathbb{R}^{r_1 \times \cdots \times r_D}$ of dimension $D$ and matrices $U^{(1)}, \ldots, U^{(D)} \in \mathbb{R}^{n \times r_i}$, as in Figure 1b. We denote $\theta \approx W \times_{d=1}^D U^{(d)^T}$ where the $(i_1, \ldots, i_D)$th entry is $W_{i_1 \ldots i_D} U^{(d)}_{i_d}$ with $U^{(d)}_{i_d}$ a column vector representing the $i_d$th row of $U^{(d)}$. $n$ is the number of features in each dimension and $r$ is the rank. Note that we are free to use a different $n$ and $r$ for each dimension, but assume these are the same for convenience of notation.

We must also place suitable priors on $W$ and $U^{(d)}$ to match the first two moments of $\theta$. This technique can be applied to arbitrary features where the number of features is too large. See Drineas and Mahoney, 2005, which use a set of feasible even for large approximation $2007$ for stationary kernels, where $V$ $E$ $φ$ inner product of finite feature vectors. In this case we use random feature maps $\Phi$. In most cases the data does not lie on a grid, nor can $K_d$ be expressed as the inner product of feature vectors. Another case where we can write kernels as inner products of features is with identity $k_d(x_i, x_j) = \delta_{ij}$. The features are unit vectors: $\phi_d(x_i) = e_i := (0, \ldots, 0, 1, 0, \ldots)$ with the non-zero at the $i$th entry, hence $U^{(d)} \phi_d(x_i) = U^{(d)}_{i}$. However this implies $U^{(d)} \in \mathbb{R}^{N \times r}$ or $\mathbb{R}^{n_d \times r}$ for inputs on a grid, so for $N$ or $n_d$ too big, computations can become too costly both in time and memory. A workaround is to use feature hashing [Weinberger et al., 2009] to obtain shorter features whose inner products are unbiased estimates of inner products of the original features. This technique can be applied to arbitrary features where the number of features is too large. See Appendix C for details.

### 2.2 Choice of Feature Map

So far, we have assumed that the kernels $k_d$ can be written as the inner product of feature vectors: $k_d(x_i, x_j) = \phi_d(x_i)^T \phi_d(x_j)$. We investigate the situations where this assumption holds. When this doesn’t hold, we explore other choices of $\phi$ that approximate $k_d$.

**Cholesky features** Consider data with inputs lying on a $D$-dimensional grid: $x_i \in X = \times_{d=1}^D X^{(d)}$, $|X^{(d)}| = n_d$ finite, where $k_d(x_i, x_j)$ only depends on the values that $x_i, x_j$ take in $X^{(d)}$. The $X^{(d)}$ can be, for example, a finite set of points in Euclidean space, or the set of values a categorical variable can take. Then the Gram matrix $K$, containing the values of the kernel evaluated at each pair of points on the full grid, can be written as $K = \otimes_{d=1}^D K^{(d)}$, a Kronecker product of the Gram matrices $K^{(d)} \subset \mathbb{R}^{n_d \times n_d}$ on each dimension [Saatçi, 2011]. The same holds for the Cholesky factor $L$ where $K = LL^T$; we have $L = \otimes_{d=1}^D L^{(d)}$ where $L^{(d)} = L^{(d)} L^{(d)^T} \subset \mathbb{R}^{n_d \times n_d}$. Then we define $\phi_d(x_i)$ to be the $i$th row of $L^{(d)}$, so that $k_d(x_i, x_j) = K^{(d)}_{ij} = \phi_d(x_i)^T \phi_d(x_j)$. In general a Cholesky decomposition for an $m$ by $m$ matrix takes $O(m^3)$ to compute. Thus $\phi_d(x_i)$ for $i = 1, \ldots, N$ require $O(n_d^3)$ to compute in total. Hence the computation of features become feasible even for large $N$ as long as the $n_d$ are reasonably small.

**Identity features** Another case where we can write kernels as inner products of features is with identity kernels $k_d(x_i, x_j) = \delta_{ij}$. The features are unit vectors: $\phi_d(x_i) = e_i := (0, \ldots, 0, 1, 0, \ldots)^T$. This is $U^{(d)} = U^{(d)}_{i}$. However this implies $U^{(d)} \in \mathbb{R}^{N \times r}$ or $\mathbb{R}^{n_d \times r}$ for inputs on a grid, so for $N$ or $n_d$ too big, computations can become too costly both in time and memory. A workaround is to use feature hashing to obtain shorter features whose inner products are unbiased estimates of inner products of the original features. This technique can be applied to arbitrary features where the number of features is too large. See Appendix C for details.

**Random feature maps** In most cases the data does not lie on a grid, nor can $k_d$ be expressed as the inner product of finite feature vectors. In this case we use random feature maps $\phi_d : X \to \mathbb{R}^n$ where $E[\phi_d(x)^T \phi_d(x')] = k_d(x, x')$. An example is random Fourier features (RFF) [Rahimi and Recht, 2007] for stationary kernels, where $\forall [\phi_d(x)^T \phi_d(x')] = O(\frac{1}{n})$. So we are introducing a further approximation $k_d(x, x') \approx \phi_d(x)^T \phi_d(x')$, with more accurate approximations for larger $n$. This is feasible even for large $N$ as $\phi_d(x)$ only takes $O(n)$ computation. See Appendix D for details. For non-stationary kernels, we can obtain features by Nyström methods [Williams and Seeger, 2001] [Drineas and Mahoney, 2005], which use a set of $n$ inducing points to approximate $K$. The kernel is evaluated for each pair of inducing points and also between the inducing points and the data, giving matrices $K_{nn}$ and $K_{Nn}$. Then $K \approx K_{Nn} K_{nn}^{-1} K_{Nn}^T = \Phi^T \Phi$ where $\Phi = L_{nn}^{-1} K_{Nn}^T$. Hence the columns of $\Phi$ can be defined to be the Nyström features.
2.3 Learning

In TGP we would like to learn the posterior distribution of $U$ and $W$. The simplest and fastest method of learning is Maximum a Posteriori (MAP), whereby we approximate the posterior with point estimates $\hat{U}, \hat{W} = \arg \max_{U,W} p(U,W|y)$. For the optimisation we may use stochastic gradient descent (SGD) with mini-batches to approximate the full gradient.

Let us examine how the computation is made efficient. We are interested in computing the derivative of the log joint distribution $\log p(U,W|y) = \log p(W) + \sum_{d=1}^{D} \log p(U^{(d)}) + \sum_{i=1}^{N} \log p(y_i|x_i, U, W)$ with respect to each $U^{(k)}$ and $W$. Now $\log p(y_i|x_i, U, W) \propto (y_i - f(x_i))^2$, hence the costly operations will be computing $f(x_i)$ and its derivatives. $f(x_i)$ can be computed efficiently by first computing $\psi_d(x_i) = U^{(d)\top} \phi_d(x_i) \in \mathbb{R}^r\forall d$ in $O(nrD)$ time, then computing $W \times_{d=1}^{D} \psi_d(x_i)$ in $O(rD)$ operations. The derivative of $f(x_i)$ with respect to $U^{(k)}$ is the outer product of $\phi_k(x_i)$ and $W \times_{d\neq k} \psi_d(x_i)$, the latter taking $O(rD)$ having computed the $\psi_d$. So the derivatives $\forall k$ take $O(r^2D)$. The derivative with respect to $\psi(x)$ is $\otimes_{d=1}^{D} \psi_d(x_i)$, which takes $O(rD)$. So all gradient computations of $\log p(y_i|U, W)$ take $O(nrD + r^2D)$ operations, giving us $O(m(nrD + r^2D))$ operations for computing the stochastic gradient on a mini-batch of size $m$. See Appendix A for a details.

The problem with a MAP estimate for $U, W$ is that only the posterior mode is used, and the uncertainty encoded in the shape of the posterior distribution is ignored. In a Bayesian setting, we wish to use samples from the posterior and average predictions over samples. For data where we can afford an $O(N)$ runtime, we may use sampling algorithms such as Hamiltonian Monte Carlo (HMC) [Duane et al., 1987, Neal, 2011]. The runtime for each HMC leapfrog step is $O(N(nrD + r^2D))$, the same time complexity as a step of full-batch gradient descent.

2.4 Discussion and Related Work

Recall that the regression function $f(x)$ in (4) can be seen as $W \times_{d=1}^{D} \psi_d(x)$, where $\psi_d(x) = U^{(d)\top} \phi_d(x)$ are lower-dimensional features in $\mathbb{R}^r$ (i.e. the $U^{(d)}$ multiplied by $\phi_d(x)$ in Figure 1b).

With this new formulation, we have:

$$f(x) = W \times_{d=1}^{D} \psi_d(x) = \sum_{i_1, \ldots, i_D=1}^{r} W_{i_1, \ldots, i_D} \prod_{d=1}^{D} (\psi_d(x))_{i_d} \tag{5}$$

Hence learning $W$ and $(U^{(d)})_{d=1}^{D}$ can be interpreted as learning features $\psi_d$ as well as their weights for the regression function, which is a linear combination of products of these features. In the case where $\psi_d(x)$ is only a function of the $d^{th}$ dimension of $x$, each $\prod_{d=1}^{D} (\psi_d(x))_{i_d}$ is separable in each dimension. Modelling data with sums of separable functions has been studied in [Beylkin et al., 2009], and its effectiveness for regression is shown by promising results on various synthetic and real data. Such additive models arise frequently in the context of ensemble learning, such as boosting and BART [Chipman et al., 2010], where a linear combination of many weak learners is used to build a single strong learner. We may interpret our model in this framework where $\prod_{d=1}^{D} (\psi_d(x))_{i_d}$ are the weak learners that share parameters, and $W_{i_1, \ldots, i_D}$ are the corresponding weights.

With this alternative interpretation in mind, we may expect our model to perform well in cases where the data displays an additive structure, with the additive components arising from a product of features on each dimension. Based on Section 2.3, we also see that our model is particularly well-suited to modelling grid-structured data. The difference between our model and [Saatçi, 2011] is that we have Kronecker structure on the features $\phi$, whereas [Saatçi, 2011] exploits Kronecker structure on the data. Moreover, our model can deal with data not on a grid, as well as data on a grid with many missing observations, since observations are not needed for constructing the features.

3 Collaborative Filtering

3.1 TGP for Collaborative Filtering

Collaborative filtering (CF) tackles the problem of making predictions about the preferences of a user on a set of items, based on the user’s ratings on other items and the ratings of other users. It is a
well-studied problem in machine learning, and an example of where the data arises in the form of a tensor (in this case as a ratings matrix $R$), often accompanied by side information about users and items. Let $R_{ij}$ be the rating of user $u_i$ on movie $v_j$, for $i = 1, \ldots, n_1$, $j = 1, \ldots, n_2$. It is natural to model this as a supervised regression problem with $R_{ij} \sim \mathcal{N}(f(u_i, v_j), \sigma^2)$ with prior $f \sim \text{GP}(0, k) [Yu et al., 2006]$. This is particularly suitable with side information, since kernels can be interpreted as measures of similarity; we can design $k$ to encode similarities between users/movies given by the side information. In particular we use a product kernel $k((u_i, v_j), (u_i', v_j')) = k_1(u_i, u_i')k_2(v_j, v_j')$ since we expect similar ratings for two user/movie pairs if the users are similar and the movies are similar. When there is no side information, it is sensible to use identity kernels $k_1(u_i, u_i') = \delta_{u_i, u_i'}$, $k_2(v_j, v_j') = \delta_{v_j, v_j'}$, i.e. that distinct users and movies are not similar a priori. With side information, we may add on further kernels $\kappa_1, \kappa_2$ modelling similarity between users/movies: $k_1(u_i, u_i') = a_1^2\delta_{u_i, u_i'} + b_1^2\kappa_1(u_i, u_i')$, $k_2(v_j, v_j') = a_2^2\delta_{v_j, v_j'} + b_2^2\kappa_2(v_j, v_j')$, where $a$ and $b$ are parameters controlling the extent to which similarity in side information leads to similarity in preference.

However, it is not clear how this single GP framework relates to the matrix factorisation approach [Billus and azzani, 1998], which forms the basis for many successful CF models. This model $R$ as a product of two low rank matrices $R \approx UV^T$. We show that our proposed TGP forms a natural connection between these two approaches, and that we recover classic matrix factorisation models as a special case. To apply TGP, first note that we have $D = 2$, and the Tucker Decomposition is simply a low-rank matrix decomposition. Using the notation $U, V$ instead of $U^{(1)}, U^{(2)}$, we have that $\theta \approx UVW^T$, hence $f(u_i, v_j) = \phi_1(u_i)^T UW V^T \phi_2(v_j)$ with the identity kernel. We have unit vector features $\phi_1(u_i) = e_i \in \mathbb{R}^{n_1}$ and $\phi_2(v_j) = e_j \in \mathbb{R}^{n_2}$. TGP therefore simplifies to:

$$R_{ij} \overset{\text{ind}}{\sim} \mathcal{N}(f(u_i, v_j), \sigma^2) \quad f(u_i, v_j) = U_i^T W V_j$$ (6)

with iid $\mathcal{N}(0, \sigma^2)$ priors on each entry of $U, V$ where $U_i$, $V_j$ are column vectors representing the $i^{th}$ and $j^{th}$ row of $U$ and $V$ respectively. Note that with $W = I$ fixed, we recover Probabilistic Matrix Factorization (PMF) [Salakhutdinov and Mnih, 2008a], a particularly effective Bayesian model in the matrix factorization framework. An extension is Bayesian PMF (BPMF) [Salakhutdinov and Mnih, 2008b] where priors are placed on the prior mean and covariance of $U_i, V_j$. In Appendix E, we show that TGP in [6] with a dense $W$ can capture the effects of the hierarchical extension of BPMF.

Returning to the case with side information, suppose it is given in the form of vectors $\omega_1(u_i), \omega_2(v_j)$, and that we expect users/movies with similar $\omega$ to show similar preferences/be preferred by similar users. For example we can encode the user age into $\omega_1$ and the movie genre into $\omega_2$ and define $\kappa_1(u_i, u_i') = \omega_1(u_i)^T \omega_1(u_i')$, $\kappa_2(v_j, v_j') = \omega_2(v_j)^T \omega_2(v_j')$. The feature vector is now $\phi_d(u_i) = [a_d e_i^T, b_d \omega_2(u_i)]^T$ for $d = 1, 2$, and we have $f(u_i, v_j) = \phi_1(u_i)^T UVW^T \phi_2(v_j)$.

### 3.2 Discussion and Related Work

Modelling data in the form of matrices and tensors has been studied in the field of multi-way data analysis and relational learning. The key idea here is to factorise the data tensor, with two notable forms of factorisation: PARAFAC [Bro, 1997] and Tucker [Tucker, 1966]. There are a few works in these domains that relate to GPs. InfTucker [Xu et al., 2011] uses the Tucker decomposition directly on the data tensor, and use a non-linear transformation of the parameters $U^{(d)}$ for the regression function. They show that the model converges in distribution to a GP with a product kernel as $r \to \infty$, for a particular choice of the non-linear transformation that matches the kernel.

There are close connections between our model and the Stochastic Relational Model [Yu et al., 2006] in relational learning. Fixing $W = I$ in our model with $D = 2$, our model reduces to [Yu et al., 2006]. The key differences are that we use features to build on the weight-space view of GPs, whereas [Yu et al., 2006] works with GPs in the function-space view. This complicates learning for kernels which cannot be expressed as an inner product of features; the authors resort to Laplace approximation for finding maximum likelihood estimates of parameters. For such kernels we use random feature maps, making learning simple and scalable.

In the domain of matrix factorisation, Lawrence and Urtasun [2009] use a GP Latent Variable Model (GP-LVM) [Lawrence, 2004]. They learn a latent vector for each movie, and pass it through a zero-mean GP with squared exponential (SE) kernel, with one GP per user. For a CF application, they
incorporate side information about movies by taking the product of these kernels with a SE kernel in the movie features.

In general, there has been a wide range of work on CF with side information, mostly building on the framework of matrix factorisation. Suppose user/item side information is given in the form of feature matrices \( F = [\omega(u_1), ..., \omega(u_n)]^T \in \mathbb{R}^{n_1 \times r} \) and \( G = [\omega(v_1), ..., \omega(v_n)]^T \in \mathbb{R}^{n_2 \times r} \). Matrix co-factorization [Singh and Gordon, 2008] (BMCF) attempts to factorise \( F, G \) and \( R \) simultaneously, whereas the Regression-based Latent Factor Model (RLFM) [Agarwal and Chen, 2009] assumes instead that \( U \) and \( V \) are linear in the feature matrices \( F \) and \( G \). Bayesian Matrix Factorization with Side Information (BMFSI) [Porteous and Welling, 2010] gives an additive model in the sense that \( R \) is assumed to be the sum of the standard matrix factorisation prediction \( UV^T \) and linear contributions of \( F \) and \( G \). Hierarchical Bayesian Matrix Factorization with Side Information (HBMFSI) [Park et al., 2013] is an extension of BFMSI with Gaussian-Wishart hyperpriors on the prior mean and variance of \( U \) and \( V \).

4 Experimental Results

Regression on spatial data We use the California house prices data from the 1990 census\footnote{Obtained from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression/cadata.} which consists of average house prices for 20,640 different locations in California. We only use the covariates longitude and latitude, and whiten them along with log-transformed house prices to each have zero mean and unit variance. We chose this data set as spatial data sometimes exhibit separability in the different dimensions. Moreover the data is clustered in urban areas, hence an additive model with each component describing different sections of California may be desirable. Using a random 50:50 train test split, we report the RMSE of the model on the training set and test set after training. We first fit a GP to the data with a squared exponential (SE) kernel on each dimension using the GPML toolbox [Rasmussen and Nickisch, 2010], optimising the hyperparameters by type-II maximum likelihood. Then using these hyperparameters we generate RFF for \( \phi \). See Appendix D for details. We implemented both the full-rank model and TGP with \( n = 25, 50, 100, 200 \) on Stan [Stan Development Team, 2012], which uses the No-U-Turn Sampler (NUTS) [Hoffman and Gelman, 2014] for inference. Note that for both models \( n \) refers to the length of features \( \phi_d(x) \).

For TGP, we use 300 warmup iterations and a further 300 samples on 4 different chains, and use the mean prediction across the samples. For full-rank, we take the same number of samples and chains, but only use 50 warmup draws as we diagnosed that convergence was reached by this point (looking at the Gelman-Rubin statistic [Gelman and Rubin, 1992] and effective sample size). The convergence statistics for TGP are in Appendix E. We can see from Figure 2 that some TGP models give lower test RMSE and higher train RMSE than the GP and the full-rank model. In fact TGP with \( r = 10 \) consistently shows higher predictive performance than full-rank for all values of \( n \), and for \( n \geq 100 \) TGP with \( r = 10 \) outperforms GP. This indicates that TGP is an effective regulariser towards simpler regression functions, namely a linear combination of separable functions. We expect bigger gains for TGP with more warmup iterations, since the convergence diagnostics suggest that TGP hasn’t quite fully mixed by 300 iterations.

![Figure 2: RMSE for GP, full-rank, and TGP for \( r = 2, 5, 10 \) for \( n = 25, 50, 100, 200 \) on the California House Price data.](image)
We further investigate the predictions of TGP by analysing the additive components in the prediction for $r = 2$. We see in Figure 3b that the components are quite different. The upper two components show complementary predictions in the Bay area (North-West) and the central area, whereas the bottom two show complementary predictions in the Los Angeles area (South-East). This confirms the hypothesis that the different additive components will learn different sections of the data. See Appendix F for further plots.

Regression on spatio-temporal data with grid structure

We use the Irish wind data\footnote{Obtained from \url{http://www.inside-r.org/packages/cran/gstat/docs/wind}} giving daily average wind speeds for 12 locations in Ireland between 1961 and 1978 (78,888 observations). We only use the covariates longitude, latitude and time. Note a 2D grid structure arises for the data when we treat the spatial covariates as one dimension and time as another. Again we whiten each covariate and observations, and use 20,000 randomly chosen data points for training and the rest for test. Using an isotropic SE kernel for space, and the sum of a periodic kernel and a SE kernel for time (to model annual periodicity and global trend), we first fit a GP efficiently exploiting the grid structure [Saatçi, 2011]. The optimised hyperparameters are then used to construct Cholesky features. Again we use NUTS for inference on both the full-rank model and TGP, using 4 chains with 100 warmup draws and 100 samples.

We use the MovieLens100K dataset giving data on movies.

All models show good convergence after 100 warmup draws, indicated by the aforementioned convergence diagnostics. Looking at Table 1 we see similar patterns in the results for the wind data as for the house prices data: the GP, which is equivalent to the full-rank model with Cholesky features (confirmed by similar train/test RMSE), shows lower training error than TGP, whereas TGP shows superior predictive performance. These results again suggest that TGP is an effective regulariser towards simpler regression functions compared to GPs. See Appendix G for plots showing predictions of TGP.

Table 1: Train/Test RMSE on Irish wind data.

| Model    | Train RMSE | Test RMSE |
|----------|------------|-----------|
| GP       | 4.8822     | 4.9915    |
| Full-rank| 4.8816     | 4.9898    |
| TGP, $r = 2$ | 4.9120     | 4.9753    |
| TGP, $r = 5$ | 4.8996     | **4.9735**|
| TGP, $r = 10$ | 4.8913    | 4.9754    |

Table 2: Test RMSE on MovieLens100K.

| Model                        | Test RMSE       |
|------------------------------|-----------------|
| BPMF                         | 0.9024 ± 0.0050 |
| TGP, $W = I$ (PMF)           | 0.9395 ± 0.0115 |
| TGP, learn $W$               | 0.9270 ± 0.0097 |
| TGP, $W = I$, side-info      | 0.9014 ± 0.0061 |
| TGP, learn $W$, side-info    | **0.8995 ± 0.0062** |
Collaborative Filtering We use the MovieLens 100K data\footnote{ Obtained from \url{http://grouplens.org/datasets/movielens/100k/} }, which consists of 100,000 ratings in \{1, \ldots, 5\} from 943 users on 1682 movies. User age, gender and occupation are given, as well as the genre of the movies. We represent this side information with binary vectors for $\omega_1(u_i), \omega_2(v_j)$ and use the formulation in (8) in Appendix E. We bin the age into five categories, and there are 20 occupations and 18 genres. Thus $\omega_1(u_i) \in \mathbb{R}^{5+2+20}$ has 3 non-zero entries, one for each feature, and $\omega_2(v_j) \in \mathbb{R}^{18}$ can have multiple non-zero entries since each movie can belong to many genres. We report the mean and standard deviation of the test RMSE on the five 80:20 train test splits that come with the data, as it will offer a sensible means of comparison with other algorithms. $N$ is too large for HMC, hence we use SGD to obtain MAP estimates for the parameters, and compare different configurations: learning $W$/fixing it to be the identity and using/not using side information, along with BPMF initialised by PMF.\footnote{ Code obtained from \url{http://www.cs.toronto.edu/~rsalakhu/BPMF.html} } We used mini-batches of size 100, and set $r = 15$ for all models as it gives best results for PMF and BPMF. We used a grid search for tuning the hyperparameters. See Appendix E for details.

From Table 2 it is evident that TGP makes good use of side information, since the RMSE decreases significantly with side information. Learning $W$ instead of fixing it helps predictive performance, but does not perform as well as BPMF. One reason is that our Gaussian prior on $W$ is not equivalent to the Gaussian-Wishart priors on the mean and variance of $U_i, V_j$ in BPMF. Another reason is that we are resorting to a MAP estimate. If we instead manage to sample from the posterior and average predictions over these samples, we expect enhanced predictions. However, note that using TGP with side information and learning $W$, we are able to get comparable/superior results to BPMF, even with a MAP estimate. We expect further improvements not only with sampling but also by using more sophisticated kernels that make better use of the side information; for example, using different hyperparameter coefficients for the different types of features. In so far as comparison was possible, these numbers are comparable to state-of-the-art algorithms in Section 3.2. A direct comparison was not possible as each use different methods for evaluation.

5 Conclusion and Discussion

We have introduced TGP, a regression model for approximating GPs with product kernels, by using feature maps and a low-rank Tucker decomposition on the parameters in the weight-space view of GPs. We may interpret the low-rank approximation as an effective regulariser towards simpler regression functions, in particular a linear combination of separable functions. This is potentially a reason it outperforms GPs for prediction in spatio-temporal data sets. We also point out that exact Cholesky features can be used with TGP in the case of grid-structured data, and random feature maps can be used for arbitrary kernels. Moreover, we highlight that TGP has runtime at most $O(N)$, making it scalable. Furthermore, we have highlighted the effectiveness of TGP in CF, and that PMF is a special case. Extending PMF, we put the user/item side information to good use; the kernel of the GP we approximate can be designed to encode similarities between different users and items, a particularly neat and natural method for modelling similarity. We confirm experimentally that side information enhances the predictive performance of TGP in collaborative filtering.

Note that TGP can easily be extended to non-Gaussian likelihoods, since all we need for SGD and HMC is the likelihood and priors to be analytic and differentiable in the parameters. It would then be interesting to compare performance on classification tasks against complex inducing points methods.

For very high dimensions where even the $r^D$ entries in $W$ are undesirable, we can use a sparse representation of $W$ with say $Q$ non-zeros. All derivations carry forward, and we obtain time complexity $O(m(nr^D + QD))$ for gradient computations in SGD. Furthermore, it would be desirable to have a sampling algorithm that scales sub-linearly, to benefit from the Bayesian approach to learning when $N$ is large and HMC is infeasible. One example is Stochastic Gradient Langevin Dynamics (SGLD) \cite{Welling2011} among many other Stochastic Gradient MCMC \cite{Ma2015} algorithms. Moreover a more efficient method of tuning hyperparameters than grid search would be ideal. We leave these extensions for future work.
References

D. Agarwal and B. Chen. Regression-based latent factor models. In *ACM SIGKDD*, 2009.

G. Beylkin, J. Garcke, and M. Mohlenkamp. Multivariate regression and machine learning with sums of separable functions. *SIAM Journal on Scientific Computing*, 2009.

D. Bilus and M.P. azzani. Learning collaborative information filters. In *ICML*, 1998.

R. Bro. PARAFAC. Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 1997.

H. Chipman, E. George, and R. McCulloch. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 2010.

P. Drineas and M. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *JMLR*, 2005.

S. Duane, A. Kennedy, B. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics letters B*, 1987.

A. Gelman and D. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 1992.

P. Hall and C. Heyde. *Martingale Limit Theory and its Application*. Academic press, 2014.

M. Hoffman and A. Gelman. The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *The Journal of Machine Learning Research*, 2014.

N. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. *NIPS*, 2004.

N. Lawrence and R. Urtasun. Non-linear matrix factorization with Gaussian processes. In *ICML*, 2009.

M. Lázaro-Gredilla, J. Quinonero-Candela, C. Rasmussen, and A. Figueiras-Vidal. Sparse spectrum Gaussian process regression. *JMLR*, 2010.

Y. Ma, T. Chen, and E. Fox. A complete recipe for stochastic gradient MCMC. In *NIPS*, 2015.

R. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2011.

S. Park, Y. Kim, and S. Choi. Hierarchical bayesian matrix factorization with side information. In *IJCAI*, 2013.

I. Porteous and M. Welling. Bayesian matrix factorization with side information and Dirichlet process mixtures. In *AAAI*, 2010.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

C. Rasmussen and H. Nickisch. Gaussian Processes for Machine Learning (GPML) toolbox, 2010.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML*, 2005.

W. Rudin. *Fourier analysis on groups*. AMS, 1964.

Y. Saatçi. *Scalable Inference for Structured Gaussian Process Models*. PhD thesis, 2011.

R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *NIPS*, 2008a.

R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using Markov Chain Monte Carlo. In *ICML*, 2008b.

M. Seeger, C. Williams, and N. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *AISTATS*, 2003.

A. Singh and G. Gordon. Relational learning via collective matrix factorization. In *ACM SIGKDD*, 2008.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *NIPS*, 2005.

Stan Development Team. Stan: A C++ library for probability and sampling, version 1.0, 2012.

A. Storkey. Truncated covariance matrices and Toeplitz methods in Gaussian processes. In *ICANN*, 1999.

M. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *AISTATS*, 2009.

L. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 1966.

K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *ICML*, 2009.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*, 2011.

C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *NIPS*, 2001.

Z. Xu, F. Yan, and Y. Qi. Infinite Tucker decomposition: Nonparametric bayesian models for multiway data analysis. *arXiv:1108.6296*, 2011.

K. Yu, W. Chu, S. Yu, V. Tresp, and Z. Xu. Stochastic relational models for discriminative link prediction. In *NIPS*, 2006.
Appendix

A Learning Algorithms for TGP

We give detailed derivations of various inference algorithms for the TGP. We have a set of $N$ observations $y_i \in \mathbb{R}$ corresponding to a set of inputs $x_i \in \mathcal{X}$, and we wish to regress $y = (y_i)_{i=1}^N$ on $X = (x_i)_{i=1}^N$. We assume that the data generating mechanism takes the form

$$y = f(X) + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I_N)$$

where $f(X) = (f(x_i))_{i=1}^N \in \mathbb{R}^N$ and also that the regression function takes the following form

$$f(x) = w^\top \otimes_{d=1}^D (U^{(d)})^\top \phi_d(x)$$

where

- $W \in \mathbb{R}^{r \times \cdots \times r}$ is a D-dimensional tensor whose entries are iid $\mathcal{N}(0, \sigma_n^2)$
- $w = vec(W)$ is the vector obtained when flattening tensor $W$, such that $W \times_{d=1}^D v_d = w^\top \otimes_{d=1}^D v_d \forall v_d \in \mathbb{R}^r$
- $(\phi_d(x))_{d=1}^D$ are the features in $\mathbb{R}^n$ extracted from $x$
- $(U^{(d)})_{d=1}^D$ are a set of real $n \times r$ matrices with $U^{(d)} iid \sim \mathcal{N}(0, \sigma_u^2)$

We assume $n > r$, and wish to learn $w$ and the $U^{(d)}$ from the data.

Note from the second point that $\nabla_w (W \times_{d=1}^D v_d) = \otimes_{d=1}^D v_d$. For $D=2$ for example, if $g(U) = s^\top U t$ for some matrix $U$ and vectors $s, t$, then $\nabla_u g(U) = s \otimes t$ where $u = vec(U)$.

First we give the complexity for calculating $f(x)$. Computing $\psi_d(x_i) = (U^{(d)})^\top \phi_d(x_i) \forall d$ requires $O(nrD)$ time, then $w^\top \otimes_{d=1}^D \psi_d(x_i)$ takes $O(rD)$ time. So time for a prediction given $\phi, U, w$ takes $O(nrD + rD)$.

The quantity of interest for MAP and HMC is the log joint distribution $p(y, U, w) = p(y|U, w)p(U)p(w)$. In full this is:

$$\log p(y|U, w) + \log p(U) + \log p(w) = -\frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - f(x_i))^2 - \frac{1}{2\sigma_u^2} \sum_{k=1}^D tr(U^{(k)^\top} U^{(k)}) - \frac{1}{2\sigma_n^2} w^\top w$$

This has the following derivatives:

$$\nabla_w \log p(w) = -w$$

$$\nabla_{U^{(k)}} \log p(U) = -r U^{(k)}$$

$$\nabla_w \log p(y_i|U, w) = \frac{1}{\sigma^2} (y_i - f(x_i)) \otimes_{d=1}^D \psi_d(x_i)$$

$$\nabla_{u^{(k)}} \log p(y_i|U, w) = \frac{1}{\sigma^2} (y_i - f(x_i)) \phi_k(x_i) \otimes (W \times_{d \neq k} v_d)$$

with the following definitions:

- $u^{(k)} = vec(U^{(k)}) \in \mathbb{R}^{nr}$
- $(W \times_{d \neq k} v_d)_l := W \times_{d=1}^D v_d'$ where $v_d' = v_d$ for $d \neq l$ and $v_l' = e_l \in \mathbb{R}^r$, the unit vector with non-zero at the $l$th entry.

The last derivative holds since $f(x) = \phi_k(x)^\top U^{(k)} (W \times_{d \neq k} \psi_d(x_i))$. Computing $W \times_{d \neq k} \psi_d(x_i)$ takes $O(rD)$ for each $k$, hence $O(rD) \forall k$. So we have that using mini-batches $\{x_{i1}, \ldots, x_{im}\}$ for
SGD, we have the following updates for MAP:

\[
\begin{align*}
    w &\leftarrow w + \frac{\epsilon^w}{2} \left( \nabla_w \log p(w_t) + \frac{N}{m} \sum_{i=1}^{m} \nabla_w \log p(y_{ti} | x_{ti}, w, U) \right) \\
u^{(k)} &\leftarrow u^{(k)} + \frac{\epsilon^u}{2} \left( \nabla_{u^{(k)}} \log p(U) + \frac{N}{m} \sum_{i=1}^{m} \nabla_{u^{(k)}} \log p(y_{ti} | x_{ti}, w, U) \right)
\end{align*}
\]

with time complexity \(O(m(nrD + QD))\).

Gathering the parameters into a vector \(\theta = (w, U^{(1)}, \ldots, U^{(k)})\), the HMC algorithm runs as follows:

1. Initialise MC by drawing \(\theta_0 = (w_0, U^{(1)}_0, \ldots, U^{(D)}_0)\) from its prior distribution.
2. For \(t = 0, \ldots, T\):
   (a) Initialise \(p \sim \mathcal{N}(0, I_Q)\), \(V^{(k)} \sim \mathcal{N}(0, I_{n \times r})\) \(\forall k\),
       \(H_t \leftarrow -\log p(w_t) - \sum_{i=1}^{N} \log p(y_i | x_i, \theta_t) + \frac{1}{2} \sum_{k=1}^{D} \text{tr}(V^{(k)T}V^{(k)}) + \frac{1}{2} p^T p\)
       \(\theta_t = (w_t, U^{(1)}_t, \ldots, U^{(D)}_t)\)
   (b) For \(l = 1, \ldots, L\):
       i. \(p \leftarrow p + \frac{\epsilon^w}{2} \left( \nabla_w \log p(w_t) + \sum_{i=1}^{N} \nabla_w \log p(y_i | x_i, \theta_t) \right)\)
       For \(k = 1, \ldots, D\):
          \(V^{(k)} \leftarrow V^{(k)} + \frac{\epsilon^u}{2} \left( \sum_{i=1}^{N} \nabla_{U^{(k)}} \log p(y_i | x_i, \theta) \right)\)
       ii. \(w \leftarrow w + \epsilon^w p\)
          For \(k = 1, \ldots, D\):
             \(u^{(k)} \leftarrow u^{(k)} + \epsilon_t V^{(k)}\)
       iii. same as i.
   (c) \(H^* \leftarrow -\log p(w) - \sum_{i=1}^{N} \log p(y_i | x_i, \theta) + \frac{1}{2} \sum_{k=1}^{D} \text{tr}(V^{(k)T}V^{(k)}) + \frac{1}{2} p^T p\)
       \(u \sim \text{Unif}[0, 1]\)
       If \(u \leq \exp(H_t - H^*)\)
          \(\theta_{t+1} = (w_{t+1}, U^{(1)}_{t+1}, \ldots, U^{(D)}_{t+1}) \leftarrow \theta = (w, U^{(1)}, \ldots, U^{(D)})\)
     else
          \(\theta_{t+1} \leftarrow \theta_t\)

From previous computations, it is easy to see that each update requires \(O(LN(nrD + r^D))\) operations.

**B Elementwise convergence of TGP to \(\mathcal{N}(0, 1)\)**

**Definition B.1. Martingale Difference Sequence** A martingale difference sequence with respect to a filtration \((\mathcal{F}_p)_{p \in \{0, 1, \ldots, r\}}\) is a real-valued sequence of random variables \(X_1, \ldots, X_r\) that satisfies:

1. \(X_p \in \mathcal{F}_p\) measurable
2. \(\mathbb{E}(|X_p|) < \infty\)
3. \(\mathbb{E}(X_p | \mathcal{F}_{p-1}) = 0\) a.s.

for all \(p \in \{1, \ldots, r\}\).

**Theorem 1 (Martingale Central Limit Theorem [Hall and Heyde 2014])**. Let \(X = \{X_1, \ldots, X_r\}\) be a sequence of random variables satisfying the following conditions:

1. \(X\) is a martingale difference sequence with respect to filtration \((\mathcal{F}_p)_{p \in \{0, 1, \ldots, r\}}\)
2. \(\sum_{p=1}^{r} \mathbb{E}(X^2_p | \mathcal{F}_{p-1}) \xrightarrow{P} 1\) as \(r \to \infty\).
3. \(\sum_{p=1}^{r} \mathbb{E}(X^2_p 1(|X_p| > \epsilon) | \mathcal{F}_{p-1}) \xrightarrow{P} 0\) as \(r \to \infty\) \(\forall \epsilon > 0\).
Then the sums \( S_r = \sum_{p=1}^{r} X_p \overset{d}{\to} \mathcal{N}(0,1) \) as \( r \to \infty \).

**Proposition 1.** Let \( n \) by \( r \) matrices \( U^{(d)} \overset{iid}{\sim} \mathcal{N}(0, \frac{1}{r} I) \) for \( d = 1, \ldots, D \), and let \( W \sim \mathcal{N}(0, I) \) where \( W \in \mathbb{R}^{r \times \ldots \times r} \) is a \( D \)-dimensional tensor. Then each element of \( W \times_{d=1}^{D} U^{(d)\top} \) converges in distribution to \( \mathcal{N}(0,1) \) as \( r \to \infty \).

**Proof.** Suppose first that \( D = 2 \). It suffices to show that 
\[ u, v \in \mathbb{R}^{r}, W \sim \mathcal{N}(0, I), W \sim \mathcal{N}(0, I) \]
\[ \Rightarrow u^\top W v \overset{d}{\to} \mathcal{N}(0, 1) \text{ as } r \to \infty \]

We define for each \( r \in \mathbb{N} \):
\[ S_0 = 0 \]
\[ S_p := \sum_{i,j=1}^{p} u_i W_{ij} v_j \]
\[ X_p := S_p - S_{p-1} = u_p W_{pp} v_p + \sum_{i=1}^{p-1} u_i W_{pi} v_i + u_i W_{ip} v_p \]
\[ \mathcal{F}_p := \{ \emptyset, \Omega \} \text{ where } \Omega \text{ is the sample space for the RVs } u, v, W \]
\[ \mathcal{F}_p := \sigma(u_i, v_j, W_{ij})_{i,j=1}^{p} \]

So we have that \( S_p = u^\top W v \), hence it suffices to check conditions 1,2,3 in the Martingale CLT.

We first show 1, that \( X \) is a martingale difference sequence. It is clear that \( X_p \) is \( \mathcal{F}_p \) measurable by definition of \( \mathcal{F}_p \). To show that \( X \) is integrable, we have:
\[
\mathbb{E}(|X_p|) \leq \mathbb{E}(|u_p W_{pp} v_p|) + \sum_{i=1}^{p-1} \mathbb{E}(|u_p W_{pi} v_i|) + \mathbb{E}(|u_i W_{ip} v_p|)
\]
\[
\leq \sqrt{\mathbb{E}(u_p^2 W_{pp}^2 v_p^2)} + \sqrt{\sum_{i=1}^{p-1} \mathbb{E}(u_p^2 W_{pi}^2 v_i^2)} + \sqrt{\mathbb{E}(u_i^2 W_{ip}^2 v_p^2)}
\]
\[
= \frac{1}{r} + (p-1) \left( \frac{1}{r} + \frac{1}{r} \right) < \infty
\]

by the inequality \( \mathbb{E}(|X|)^2 \leq \mathbb{E}(X^2) \) (shown using convexity of \( g : x \to x^2 \) and Jensen’s inequality) and independence of \( u, v, W \). Also we have:
\[
\mathbb{E}(X_p \mid \mathcal{F}_{p-1}) = \mathbb{E}(u_p W_{pp} v_p) + \sum_{i=1}^{p-1} \mathbb{E}(u_p W_{pi}) v_i + u_i \mathbb{E}(W_{ip} v_p)
\]
\[
= 0
\]

since \( u_p, v_p, W_{pi}, W_{ip} \) are independent of \( \mathcal{F}_{p-1} \) and have zero mean. Hence \( X \) forms a martingale difference sequence.

To verify the next two conditions, we first prove a lemma that will help us do so. This is the generalisation of Chebyshev’s inequality to higher moments:

**Lemma 2.** Suppose \( X \) is a random variable with bounded \( n \text{th} \) moment for some \( n \in \mathbb{N} \). Then
\[
\mathbb{P}(|X - \mathbb{E}(X)| > \epsilon) \leq \frac{\mathbb{E}(|X - \mathbb{E}(X)|^n)}{\epsilon^n} \quad \forall \epsilon > 0.
\]

**Proof.** Without loss of generality, assume \( \mathbb{E}(X) = 0 \). Then
\[
\mathbb{P}(|X| > \epsilon) = \mathbb{E}[|X| > \epsilon] = \frac{1}{\epsilon^n} \mathbb{E}[\epsilon^n |X| > \epsilon] \leq \frac{1}{\epsilon^n} \mathbb{E}[|X|^n \mathbb{1}(|X| > \epsilon)] \leq \frac{\mathbb{E}[|X|^n]}{\epsilon^n}
\]

\( \square \)
Note Lemma 2 shows that convergence in \(L^n\) implies convergence in probability. So to show conditions 2 and 3 of the martingale CLT, it suffices to show that the expectations of the quantities on the left hand sides converge to the right hand side as scalars:

2'. \(\sum_{p=1}^{r} E(X_p^2) \to 1\) as \(r \to \infty\).

3'. \(\sum_{p=1}^{r} E(X_p^2 \mathbb{I}(|X_p| > \epsilon)) \to 0\) as \(r \to \infty\) for all \(\epsilon > 0\).

Let us show 2'. In \(E(X_p^2)\), note that all cross terms in (7) cancel since all terms have mean 0. So we have:

\[
E(X_p^2) = E(u_p^2 W_p^2 v_p^2) + \sum_{i=1}^{p-1} E(u_p^2 W_p^2 v_i^2) + E(u_p^2 W_p^2 v_p^2) \\
= \frac{1}{r^2} + (p-1) \left( \frac{1}{r^2} + \frac{1}{r^2} \right) = \frac{2p-1}{r^2}
\]

\[
\Rightarrow \sum_{p=1}^{r} E(X_p^2) = \frac{2}{r^2} \sum_{p=1}^{r} p - r \cdot \frac{1}{r^2} = \frac{2}{r^2} \left( \frac{r+1}{2} \right) - \frac{1}{r} = 1
\]

To show 3', we first note that for a random variable \(X\),

\[
\int_\delta^\infty I(X > t)dt = (X - \delta)I(X > \delta) \text{ for } \delta \in \mathbb{R}
\]

Setting \(X = X_p^2, \delta = \epsilon^2\) and rearranging we have:

\[
X_p^2 I(|X_p| > \epsilon) = X_p^2 I(X_p^2 > \epsilon^2) = \epsilon^2 I(X_p^2 > \epsilon) + \int_{\epsilon}^{\infty} I(X_p^2 > t)dt \\
= \epsilon^2 I(|X_p| > \epsilon) + \int_{\epsilon}^{\infty} 2sI(|X_p| > s)ds \text{ by change of variables } t = s^2
\]

\[
\Rightarrow E[X_p^2 \mathbb{I}(|X_p| > \epsilon)] = \epsilon^2 \mathbb{P}(|X_p| > \epsilon) + \int_{\epsilon}^{\infty} 2s \mathbb{P}(|X_p| > s)ds
\]

Now we would like to use Lemma 2 to upper bound the right hand side. Note we want to use even \(n\) such that \(E[|X|^n] = E(X^n)\), since we know how to compute \(E(X_p^n)\) but not \(E[|X|^n]\). Also note that \(\mathbb{P}(|X_p| > s)\) can be bounded by \(\frac{E(X_p^n)}{s^n}\). So we want \(n > 2\) for the bound on the integral to become finite. Hence we use \(n = 4\), and show that \(E(X_p^4)\) is sufficiently small so that even when we sum over \(p = 1, \ldots, r\), we have that the upper bound tends to 0 as \(r \to \infty\). First we compute \(E(X_p^4)\).

Note from the multinomial theorem:

\[
(x_1 + x_2 + \cdots + x_m)^n = \sum_{k_1+k_2+\cdots+k_m = n} \binom{n}{k_1, k_2, \ldots, k_m} \prod_{1 \leq t \leq m} x_t^{k_t}
\]

where

\[
\binom{n}{k_1, k_2, \ldots, k_m} = \frac{n!}{k_1! k_2! \cdots k_m!}
\]

Applying this to \(X_p^4\) and taking the expectation, we see that the only cross terms that survive are products of even powers of the terms, namely where two of the \(k_i\) are 2 and the rest are 0.
Noting \( \binom{n}{2,2} = 6 \), and that \( \mathbb{E}(X^4) = 3\sigma^4 \) for \( X \sim N(0, \sigma^2) \) we have:

\[
\mathbb{E}[X_p^2] = \mathbb{E}[u_p^4 W_p^4 v_p^4] + \sum_{i=1}^{p-1} u_p^4 W_{pi}^4 v_i^4 + u_1^4 W_{p1}^4 v_1^4 \\
+ 6 \mathbb{E}\left[ \left( \sum_{i=1}^{p-1} u_p^2 W_{pi}^2 v_i^2 \right)^2 \right] \\
+ 6 \mathbb{E}\left[ \left( \sum_{i \neq j} u_p^2 W_{pi}^2 v_i^2 u_p^2 W_{pj}^2 v_j^2 + u_p^2 W_{pi}^2 u_p^2 W_{pj}^2 W_{ipj}^2 \right)^2 \right] \\
+ 6 \mathbb{E}\left[ \sum_{i,j=1}^{p-1} u_p^2 W_{pi}^2 u_p^2 W_{pj}^2 W_{ipj}^2 \right]
\]

\[
= (2p - 1) \cdot 3 \cdot \frac{3}{r^2} + 3 \cdot 3 \cdot \frac{3}{r^2} + 6 \cdot 2(p - 1) \cdot \frac{3}{r^2} \cdot \frac{1}{r^2} + \frac{1}{r^2} \\
+ 6 \cdot 2 \left( \frac{p - 1}{2} \right) \cdot \frac{3}{r^2} \cdot \frac{1}{r^2} + 6(p - 1)^2 \cdot \frac{3}{r^2} \cdot \frac{1}{r^2} \\
= \frac{3}{r^2} (8p^2 + 8p - 7)
\]

So \( \mathbb{P}(|X_p| > \epsilon) \leq \frac{3}{r^2} (8p^2 + 8p - 7) \). Hence

\[
\mathbb{E}[X_p^2 \mathbb{I}(|X_p| > \epsilon)] \leq \frac{3}{r^2} (8p^2 + 8p - 7) + \int_{\epsilon}^{\infty} 2s \cdot \frac{3}{s^2} (8p^2 + 8p - 7) ds \\
= \frac{3}{r^2} (8p^2 + 8p - 7) \left( \frac{1}{\epsilon^2} + \int_{\epsilon}^{\infty} \frac{2}{s} ds \right) \\
= \frac{3C}{r^2} (8p^2 + 8p - 7)
\]

where \( \int_{\epsilon}^{\infty} \frac{2}{s} ds = C \). So

\[
\sum_{p=1}^{r} \mathbb{E}[X_p^2 \mathbb{I}(|X_p| > \epsilon)] \leq \frac{C}{r^2} \sum_{p=1}^{r} 8p^2 + 8p - 7 = O\left( \frac{1}{r} \right) \rightarrow 0 \text{ as } r \rightarrow \infty
\]

since \( \sum_{p=1}^{r} 8p^2 + 8p - 7 = O(r^3) \).

So we have shown conditions 1.2’.3’, hence by martingale CLT we have that

\[
S_r = u^\top W u \xrightarrow{d} \mathcal{N}(0, 1) \text{ as } r \rightarrow \infty
\]

\[\square\]

C Feature Hashing

Suppose we have features \( \phi(x) \in \mathbb{R}^n \). When \( n \) is too large, we may use feature hashing [Weinberger et al. 2009] to reduce the dimensionality of \( \phi \):

**Lemma 3.** Let \( h : \{1, \ldots, n\} \rightarrow \{1, \ldots, m\} \) be a hash function for \( m \ll n \), i.e. \( \mathbb{P}(h(i) = j) = \frac{1}{m} \) for all \( j \in \{1, \ldots, m\} \). Also let \( \xi : \{1, \ldots, n\} \rightarrow \{\pm 1\} \) be a hash function.

Define \( \bar{\phi}(x) \in \mathbb{R}^m \) as follows: \( \bar{\phi}_j(x) = \sum_{i : h(i) = j} \xi(i) \phi_i(x) \)

Then \( \mathbb{E}[\bar{\phi}(x) \bar{\phi}(x')] = \phi(x) \bar{\phi}(x') \), \( \text{Var}[\bar{\phi}(x) \bar{\phi}(x')] = O\left( \frac{1}{m} \right) \).

D Random Fourier Features

**Theorem 4** (Bochner’s Theorem [Rudin 1964]). A stationary kernel \( k(d) \) is positive definite if and only if \( k(d) \) is the Fourier transform of a non-negative measure.
For RFF the kernel can be approximated by the inner product of random features given by samples from its spectral density, in a Monte Carlo approximation, as follows:

\[
k(x - y) = \int_{\mathbb{R}^D} e^{iuv^T(x-y)} d\mathbb{P}(v) \propto \int_{\mathbb{R}^D} p(v) e^{iuv^T(x-y)} dv = \mathbb{E}_{p(v)}[e^{iuv^T(x-y)*}]
\]

\[
= \mathbb{E}_{p(v)}[Re\{e^{iuv^T(x-y)*}\}]
\]

\[
\approx \frac{1}{N} \sum_{k=1}^{N} Re\{e^{iuv_k^T(x-y)*}\}
\]

\[
= \mathbb{E}_\nu[\phi(x)^T \phi(y)]
\]

where \(\phi(x) = \sqrt{\frac{2}{n}} (\cos(v_1^T x + b_1), \ldots, \cos(v_m^T x + b_m))\) with spectral frequencies \(v_k\) iid samples from \(p(v)\) and \(b_k\) iid samples from \(U[0, 2\pi]\).

For a one dimensional squared exponential kernel \(k(x, y) = \sigma_f^2 \exp\left(-\frac{(x-y)^2}{2\sigma_n^2}\right)\), the spectral density is \(N(0, l^{-2})\). So we use features \(\phi(x) = \sigma_f \sqrt{\frac{2}{n}} (\cos(v_1^T x + b_1), \ldots, \cos(v_m^T x + b_m))\) where \(v_k\) iid samples from \(N(0, l^{-2})\) and \(b_k\) iid samples from \(U[0, 2\pi]\).

E Collaborative Filtering

E.1 Connections of TGP with BPMF

\[
\begin{bmatrix}
U^T_i \\
u_i^T \\
1
\end{bmatrix}
\begin{bmatrix}
\nu_i \mu_v \\
\nu_i \lambda_u
\end{bmatrix}
\begin{bmatrix}
\nu_j \mu_v \\
\nu_j \lambda_u
\end{bmatrix}
\begin{bmatrix}
V_j' \\
1
\end{bmatrix}
\]

Figure 4: BayesPMF reparametrised.

Should we decide to learn \(W\) in TGP, interesting parallels arise between our model and BPMF. Observe from the following that learning \(W\) can be a proxy for learning the prior mean and covariance of \(U\) and \(V\), as is done in the BPMF model:

\[
U_i \sim N(\mu_u, \Lambda_u), V_j \sim N(\mu_v, \Lambda_v) \Rightarrow U_i = \mu_u + L_u u_i, V_j = \mu_v + L_v v_j
\]

where \(u_i, v_j \sim N(0, I), \Lambda_u = L_u L_u^T, \Lambda_v = L_v L_v^T\)

\[
\Rightarrow U_i^T V_j = \mu_u^T \mu_v + \mu_u^T L_v v_j + u_i^T L_u \mu_v + u_i^T L_u L_v v_j = U_i^T W V_j'
\]

where \(U_i^T = [u_i^T, 1], W = [L_u L_v, L_u^T \mu_v, L_u^T \mu_v, \mu_u^T L_v, \mu_u^T \mu_v], V_j' = [v_j; 1]\), as displayed in Figure 4. So a full \(W\) with standard iid Gaussian priors on \(U, V\) can capture the effects of modelling \(U, V\) with non-zero means and full covariances for each row of \(U, V\), as in BPMF.

E.2 Using Binary Vectors for Side Information

Note if the side information \(\omega_1(u_i), \omega_2(v_j)\) are binary vectors with non-zeros at indices \(I_i, J_j\) respectively, we have:

\[
f(u_i, v_j) = (a_1 U_i + b_1 \sum_{k \in I_i} U_{n_1+k})^T W (a_2 V_j + b_2 \sum_{k \in J_j} V_{n_2+k})
\]
which can be reparametrised to:

\[ f(u_i, v_j) = a(U_i + b \sum_{k \in \mathcal{I}_i} U_{n_1 + k})^T W(V_j + c \sum_{k \in \mathcal{J}_j} V_{n_2 + k}) \]  

(8)

F California House Prices Data

Figure 5: Heatmap showing the four additive components of predictions of the last sample of TGP for \( r = 2, n = 200 \), using uniform colouring scheme.

Figure 6: Zoom in on LA area of Figure 3b.
Table 3: Mean and standard deviation of Gelman Rubin statistic for HMC on TGP.

| Model          | n = 25   | n = 50   | n = 100  | n = 200  |
|----------------|----------|----------|----------|----------|
| TGP, r = 2     | 2.67 ± 1.34 | 2.55 ± 1.37 | 2.10 ± 0.70 | 1.92 ± 0.67 |
| TGP, r = 5     | 1.06 ± 0.27 | 1.06 ± 0.19 | 1.15 ± 0.11 | 1.11 ± 0.11 |
| TGP, r = 10    | 1.00 ± 0.03 | 1.02 ± 0.04 | 1.01 ± 0.02 | 1.06 ± 0.03 |

Table 4: Mean and standard deviation of Effective Sample Size (out of 1200) for HMC on TGP.

| Model          | n = 25   | n = 50   | n = 100  | n = 200  |
|----------------|----------|----------|----------|----------|
| TGP, r = 2     | 230 ± 459 | 5 ± 17   | 11 ± 81  | 12 ± 74  |
| TGP, r = 5     | 244 ± 118 | 121 ± 70 | 34 ± 64  | 42 ± 79  |
| TGP, r = 10    | 692 ± 152 | 196 ± 93 | 310 ± 111 | 96 ± 165 |
G Irish Wind Data

![Figure 8: The predictions for TGP with $r = 5$ on the 12 locations. The light blue lines are the true observations, the yellow are the mean predictions, and the blue show 2.5% and 97.5% percentiles of predictions for samples.]

H MovieLens 100K

Hyperparameters were tuned on the following values. For PMF and fixed $W$ TGP: $\sigma_u = [0.3, 0.1, 0.03], \sigma^2 = [1.0, 0.1, 0.01, 0.001], \epsilon_u = [10^{-5}, 10^{-6}, 10^{-7}]$ where $\epsilon_u, \epsilon_w$ are the step sizes for SGD on $U/V$ and $W$ respectively. We noticed that for a fixed $W$ the model overfits quickly in less than 30 epochs, whereas when learning $W$ the test RMSE decreases steadily. So we used a different grid of parameters for tuning the models where $W$ is learned: $\sigma_u = [0.3, 0.1], \sigma^2 = [1.0, 0.75], \epsilon_u, \epsilon_w = [10^{-5}, 10^{-6}]$. For models with side information, we tuned on $\alpha = [0.25, 0.5, 0.75], b, c = [0.15, 0.3, 0.45]$. 