Projecting basis functions with tensor networks for Gaussian process regression

Menzen, Clara; Memmel, Eva; Batselier, Kim; Kok, Manon

DOI
10.1016/j.ifacol.2023.10.340

Publication date
2023

Document Version
Final published version

Published in
IFAC-PapersOnLine

Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 56-2 (2023) 7288–7293

Projecting basis functions with tensor networks for Gaussian process regression

Clara Menzen, Eva Memmel, Kim Batselier, Manon Kok

Delft Center for Systems and Control, TU Delft, Netherlands

{c.m.menzen,e.m.memmel,k.batselier,m.kok-1}@tudelft.nl

Abstract: This paper presents a method for approximate Gaussian process (GP) regression with tensor networks (TNs). A parametric approximation of a GP uses a linear combination of basis functions, where the accuracy of the approximation depends on the total number of basis functions M. We develop an approach that allows us to use an exponential amount of basis functions without the corresponding exponential computational complexity. The key idea to enable this is using low-rank TNs. We first find a suitable low-dimensional subspace from the data, described by a low-rank TN. In this low-dimensional subspace, we then infer the weights of our model by solving a Bayesian inference problem. Finally, we project the resulting weights back to the original space to make GP predictions. The benefit of our approach comes from the projection to a smaller subspace: It modifies the shape of the basis functions in a way that it sees fit based on the given data, and it allows for efficient computations in the smaller subspace. In an experiment with an 18-dimensional benchmark data set, we show the applicability of our method to an inverse dynamics problem.

Keywords: Gaussian process regression, tensor networks, reduced-rank approximations.

1. INTRODUCTION

A fundamental task in control is regression and a popular method of choice for regression with uncertainty bounds are Gaussian processes (GPs) (Rasmussen and Williams, 2006). Being flexible function approximators, GPs are capable of using the information in data to learn rich representations and complex structures. In control, GPs have been applied for various problems, including model predictive control e.g. Hewing et al. (2019), nonlinear state estimation e.g. Berntorp (2021) and system identification e.g. Chiuso and Pillonetto (2019). For a tutorial for GPs in learning and control, we refer the reader to Liu et al. (2018). Unfortunately, the appealing features of a GP come at a cost of poor scalability, with a computational complexity that grows cubically with the number of data points N, making GPs prohibitively expensive for large-scale data.

Among many efforts to make GPs feasible for large-scale data, see e.g. Quinonero-Candela and Rasmussen (2005); Liu et al. (2020), the work by Solin and Särkkä (2020) approximates the kernel function as a truncated series of inner products of basis functions. In this way, the GP model becomes parametric where each function value is computed as a linear combination of basis functions. Predictions for unseen inputs can be made with a cost of $\mathcal{O}(NM^2)$, where M is the number of basis functions that defines the quality of the kernel function approximation. Consequently, in cases when M needs to be chosen largely to achieve a good approximation, the complexity that is dominated by M can become large. In this work, we overcome this limitation with promising tools from multilinear algebra: tensor networks (TNs). TNs, as e.g. described in Kolda and Bader (2009), are networks of multidimensional arrays, also called tensors, that can approximate data or models in a compressed format while preserving the essence of the information. To the best of the authors' knowledge, there is little literature that applies TN in the framework of GPs. Existing work uses TNs for scalability and/or model interpretability and expressiveness - see e.g. Izmailov et al. (2018); Kirstein et al. (2022); Konstantinidis et al. (2021). We apply TNs in the framework of Solin and Särkkä (2020) to perform GP regression with a computational complexity of $\mathcal{O}(R^4 M_d^2 N)$, where R is called the rank of a TN, and where

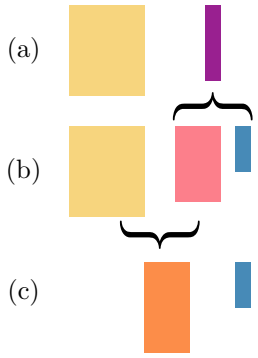

Fig. 1. Illustration of the fundamental idea of our proposed method. (a) GP approximation in terms of a parametric model with basis functions (yellow) and corresponding weights (violet). (b) Projection matrix (red) that projects the basis functions to a smaller subspace where a smaller weight vector (blue) lives. (c) 'New' parametric model with projected basis functions (orange) and a smaller weight vector (blue).

10.1016/j.ifacol.2023.10.340

M_d is the number of basis functions for one dimension, such that M_d is much smaller than the total number of basis functions M.

Our contribution is inspired by Calandra et al. (2016). In their work, a manifold is learned to transform the inputs into a low-dimensional space where GP regression is performed. In this way, functions that e.g. violate the smoothness assumption can still be described well by a GP in the low-dimensional space. We take inspiration from this idea in terms of how we transform the parametric model that approximates the GP (Fig. 1 (a)). As a first step, the weights are split into a tall matrix (red) and a vector that has fewer components than the original weight vector (blue) (Fig. 1 (b)). This is achieved by computing the weight vector as a low-rank TN with the well-known alternating linear scheme (ALS). Then, the tall projection matrix (red) is absorbed into the matrix containing the collection of basis functions (Fig. 1 (c)). We interpret this step as projecting the basis functions into a smaller subspace. The second step is to solve a Bayesian inference problem with the projected model to find a posterior distribution of the smaller weight vector. Finally, this distribution can be projected back to the original space to do approximate GP predictions.

2. BACKGROUND

In this section, we briefly review the necessary background on GP regression and the reduced-rank approximation introduced by Solin and Särkkä (2020), as well as kernel machines, where the model weights are modeled as a TN.

2.1 Gaussian process regression

Given a data set of N observations, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ and $\mathbf{y} = y_1, y_2, \ldots, y_N$, the GP model is given by

$$\begin{aligned} y_n &= f(\mathbf{x}_n) + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, \sigma_y^2) \\ f(\mathbf{x}) &\sim \mathcal{GP}(m(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}')), \end{aligned} \tag{1}$$

where a mean function $m(\cdot)$ and covariance function also called kernel function $\kappa(\cdot, \cdot)$ define the GP, σ_y^2 denotes the variance of the measurement noise $\boldsymbol{\epsilon}$, that is assumed to be zero-mean Gaussian and i.i.d., and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^{1\times D}$ are D-dimensional inputs.

In GP regression, the well-known prediction equations for an unseen input $\mathbf{x}_* \in \mathbb{R}^{1\times D}$ are given by

$$\begin{aligned} f_* \mid \mathbf{y} &\sim \mathcal{N}\left(\mathbb{E}\left(f_*\right), \mathbb{V}\left(f_*\right)\right), \text{ with} \\ \mathbb{E}\left(f_*\right) &= \mathbf{k}_*^\top \left[\mathbf{K} + \sigma_y^2 \mathbf{I}_N\right]^{-1} \mathbf{y} \text{ and} \\ \mathbb{V}\left(f_*\right) &= k_{**} - \mathbf{k}_*^\top \left[\mathbf{K} + \sigma_y^2 \mathbf{I}_N\right]^{-1} \mathbf{k}_*, \end{aligned} \tag{2}$$

where the entries of the kernel matrices are computed with $[\mathbf{K}]_{nn'} = \kappa(\mathbf{x}_n, \mathbf{x}_{n'})$, $[\mathbf{k}_*]_n = \kappa(\mathbf{x}_n, \mathbf{x}_*)$ and $k_{**} = \kappa(\mathbf{x}_*, \mathbf{x}_*)$.

The required matrix inversion in (2) is a GP's computational bottleneck of $\mathcal{O}(N^3)$ when N is large.

2.2 Hilbert space methods for reduced-rank GP regression

Solin and Särkkä (2020) approximate the kernel function by a truncated series of inner products between basis functions. More specifically, given input vectors that lay centered in a domain confined by a D-dimensional hyperbox $[-L_1, L_1] \times [-L_2, L_2] \times \cdots \times [-L_D, L_D]$, the kernel function approximation is given by

$$\kappa(\mathbf{x}, \mathbf{x}') \approx \sum_{m=1}^{M} S([\boldsymbol{\lambda}]_m) \boldsymbol{\phi}_m(\mathbf{x}) \boldsymbol{\phi}_m^\top(\mathbf{x}'). \tag{3}$$

The function $\boldsymbol{\phi}_m(\mathbf{x})$ is the mth eigenfunction of the negative Laplace operator on the confined hyperbox domain, subject to boundary conditions, such as Dirichlet.

The basis functions $\boldsymbol{\phi}_m(\mathbf{x})$ are multivariate basis functions, computed as a product D univariate eigenfunctions, where the (m_d)th entry is computed with

$$\left[\boldsymbol{\phi}\left([\mathbf{x}_n]_d\right)\right]_{m_d} = \frac{1}{\sqrt{L_d}} \sin\left(\pi m_d \frac{[\mathbf{x}_n]_d + L_d}{2L_d}\right). \tag{4}$$

The entry $[\mathbf{x}_n]_d$ is the dth dimension of the nth input and it is $\boldsymbol{\phi}\left([\mathbf{x}_n]_d\right) \in \mathbb{R}^{M_d}$, where M_d is the number of basis functions for the dth dimension. The corresponding eigenvalue is denoted by $[\boldsymbol{\lambda}]_m$.

The eigenvalue corresponding to the mth eigenfunction is computed as a sum of D terms, where each term is computed with

$$[\boldsymbol{\lambda}]_{m_d} = \left(\frac{\pi m_d}{2L_d}\right)^2. \tag{5}$$

The function $S(\cdot)$ denotes the spectral density of the Gaussian kernel, computed dimension-wise with

$$S(\sqrt{[\boldsymbol{\lambda}]_{m_d}}) = \sigma_f^2 \sqrt{2\pi} \ell \exp\left(-\frac{\ell^2}{2}[\boldsymbol{\lambda}]_{m_d}\right), \tag{6}$$

where σ_f^2 and ℓ denote the hyperparameters, i.e., signal variance and length scale, respectively.

Given (3), a reduced-rank approximation of the kernel matrix is given by

$$\mathbf{K} \approx \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^\top = (\boldsymbol{\Phi}\boldsymbol{\Lambda}^{\frac{1}{2}})(\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{\Phi}^\top), \tag{7}$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{N\times M}$ contain the basis functions, $\boldsymbol{\Lambda} \in \mathbb{R}^{M\times M}$ contain the leading M eigenvalues and $\boldsymbol{\Lambda}^{\frac{1}{2}}$ is the Cholesky factor of $\boldsymbol{\Lambda}$.

With the given kernel approximation, the GP can be written as a parametric model given by

$$\mathbf{y} = \boldsymbol{\Phi}\mathbf{w} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_y^2 \mathbf{I}_N), \tag{8}$$

where $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda})$ are the model weights. The prediction equations for this reduced-rank approach are given by

$$\begin{aligned} \mathbb{E}(f_*) &= \boldsymbol{\phi}_* \underbrace{(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \sigma_y^2\boldsymbol{\Lambda}^{-1})^{-1}\boldsymbol{\Phi}^\top\mathbf{y}}_{\mathbb{E}(\mathbf{w}\mid\mathbf{y})} \\ \mathbb{V}(f_*) &= \boldsymbol{\phi}_* \underbrace{\sigma_y^2(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \sigma_y^2\boldsymbol{\Lambda}^{-1})^{-1}}_{\mathbb{V}(\mathbf{w}\mid\mathbf{y})} \boldsymbol{\phi}_*^\top, \end{aligned} \tag{9}$$

where $\mathbf{w} \mid \mathbf{y}$ denotes the posterior distribution of the weights.

In terms of computational complexity, the reduced-rank approach requires $\mathcal{O}(M^3)$ for hyperparameter training and $\mathcal{O}(NM^2)$ for inference. The number of basis functions M needs to be chosen based on multiple criteria: the size of the domain, the required accuracy to approximate the kernel matrix, and the dimensionality of the problem.

2.3 Kernel machines in TN format

In the context of kernel machines, Stoudenmire and Schwab (2016); Batselier et al. (2017); Wesel and Batselier (2021), model the weights of the parametric model as a low-rank TN, such that it is

$$\begin{aligned} \mathbf{y} &= \mathbf{\Phi}\mathbf{w} + \boldsymbol{\epsilon} \\ &\text{s.t. } \mathbf{w} \text{ being a low-rank TN,} \end{aligned} \tag{10}$$

where the model weights are not treated as a random variable, but as deterministic. Furthermore, the matrix $\mathbf{\Phi}$, containing the basis functions, is written as a row-wise Khatri-Rao product given by

$$\mathbf{\Phi} = \begin{bmatrix} \boldsymbol{\phi}([\mathbf{x}_1]_1) \otimes \cdots \otimes \boldsymbol{\phi}([\mathbf{x}_1]_d) \otimes \cdots \otimes \boldsymbol{\phi}([\mathbf{x}_1]_D) \\ \vdots \\ \boldsymbol{\phi}([\mathbf{x}_n]_1) \otimes \cdots \otimes \boldsymbol{\phi}([\mathbf{x}_n]_d) \otimes \cdots \otimes \boldsymbol{\phi}([\mathbf{x}_n]_D) \\ \vdots \\ \boldsymbol{\phi}([\mathbf{x}_N]_1) \otimes \cdots \otimes \boldsymbol{\phi}([\mathbf{x}_N]_d) \otimes \cdots \otimes \boldsymbol{\phi}([\mathbf{x}_N]_D) \end{bmatrix}, \tag{11}$$

such that the nth row is a vector computed as a Kronecker product of D vectors $\boldsymbol{\phi}([\mathbf{x}_n]_1), \ldots, \boldsymbol{\phi}([\mathbf{x}_n]_d), \ldots, \boldsymbol{\phi}([\mathbf{x}_n]_D)$. The dth vector $\boldsymbol{\phi}([\mathbf{x}_n]_d) \in \mathbb{R}^{1\times M_d}$ contains M_d basis functions where the subscript d denotes the dth dimension.

Assuming the same number of basis functions in each dimension, i.e., $M_1 = M_2 = \cdots = M_D$, the storage cost of $\mathbf{\Phi}$ is in principle $\mathcal{O}(NM_d^D)$. However, in practice there is no need to compute $\mathbf{\Phi}$ explicitly, because it can be stored as D matrices of size $N \times M_d$. In this way, the storage cost of $\mathbf{\Phi}$ is linear in D, i.e., $\mathcal{O}(NDM_d)$.

Wesel and Batselier (2021) approximate the Gaussian kernel with the structure in (11), by computing the basis functions as (4) weighted with (5).

To solve (10), Stoudenmire and Schwab (2016); Batselier et al. (2017) model $\mathbf{w}$ as a tensor train (TT) decomposition Oseledets (2011) and Wesel and Batselier (2021) as a CANDECOMP/PARAFAC decomposition (Kolda and Bader, 2009). Since this paper focuses on the TT decomposition, we will consider this case in the following. A weight vector modeled as a TT is given in terms of three-dimensional tensors, called TT-cores denoted by $\boldsymbol{\mathcal{W}}^{(1)}, \boldsymbol{\mathcal{W}}^{(2)}, \ldots, \boldsymbol{\mathcal{W}}^{(D)}$, where $\boldsymbol{\mathcal{W}}^{(d)} \in \mathbb{R}^{R_d M_d R_{d+1}}$ and $R_1, R_2, \ldots, R_{D+1}$ are the ranks of the decomposition. The ranks determine the accuracy of the representation, as well as the complexity of computations with a TT. According to the definition of the TT decomposition, the tensor $\boldsymbol{\mathcal{W}}$ representing $\mathbf{w}$ can be computed element-wise with

$$[\boldsymbol{\mathcal{W}}]_{i_1 i_2 \ldots i_D} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_{D+1}=1}^{R_{D+1}} [\boldsymbol{\mathcal{W}}_1]_{r_1 i_1 r_2} \cdots [\boldsymbol{\mathcal{W}}_D]_{r_D i_D r_{D+1}},$$

where $R_1 = R_{D+1} = 1$.

Because of the multilinear nature of the TT decomposition, a vector represented as a TT can be written as a function that is linear with respect to a specific TT-core. This is achieved by computing a matrix $\mathbf{W}_{\setminus d} \in \mathbb{R}^{M \times R_d M_d R_{d+1}}$ from all TT-cores except the dth, such that $\mathbf{w}$ is modeled as

$$\mathbf{w} = \mathbf{W}_{\setminus d}\mathbf{w}^{(d)}, \tag{12}$$

where $\mathbf{w}^{(d)} \in \mathbb{R}^{R_d M_d R_{d+1}}$ is the vectorized dth TT-core.

Exploiting the multilinearity of the TT decomposition, the regularized least square problem to find the model weights is given by

$$\min_{\mathbf{w}^{(d)}} \|\mathbf{y} - \mathbf{\Phi}\mathbf{W}_{\setminus d}\mathbf{w}^{(d)}\|_2^2 + \lambda \|\mathbf{W}_{\setminus d}\mathbf{w}^{(d)}\|_2^2, \tag{13}$$

where λ denotes the regularization parameter.

Equation (13) is solved by applying the well-known alternating linear scheme (ALS). The ALS is an iterative block coordinate descent method that updates one TT-core at a time while assuming the other TT-cores to be known and fixed. One so-called sweep of the ALS is updating all TT-cores once by solving (13) sequentially. After multiple sweeps, when a convergence criterion is met, a TT representation of the model weights $\mathbf{w}$ is found.

In the ALS, there is a simple way to avoid that the conditioning of the subproblem becomes worse than the overall problem: keeping the TT in site-d-mixed canonical format (Holtz et al., 2012, p.113). In this format, the TT-cores are computed in a way such that it is

$$\mathbf{W}_{\setminus d}^\top \mathbf{W}_{\setminus d} = \mathbf{I}_{R_d M_d R_{d+1}}. \tag{14}$$

3. METHODS

In our method, we compute the mean and covariance of a weight vector with a computational complexity of $\mathcal{O}(R^4 M_d^2 N)$, where R can be assumed to be small and $M_d \ll M$. We combine the model of the kernel machine (10) that models the weight as a low-rank TN, and the probabilistic parametric model (8), that approximated a GP. In this way, we can exploit the TN structure that allows for efficient computation and at the same time take prior knowledge about the weights into account. Subsequently, we compute a posterior distribution for the weights in a TN framework, allowing for both a mean and covariance estimate for the predictions.

In this context, we consider an equivalent way of writing model (8) and (9), given by

$$\mathbf{y} = \mathbf{\Phi}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{w} + \boldsymbol{\epsilon}, \;\; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_y^2 \mathbf{I}_N), \;\; \mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{15}$$

and

$$\begin{aligned} \mathbb{E}(f_*) &= \boldsymbol{\phi}_* \mathbf{\Lambda}^{\frac{1}{2}} \underbrace{(\mathbf{\Phi}^\top \mathbf{\Lambda} \mathbf{\Phi} + \sigma_y^2 \mathbf{I}_M)^{-1} \mathbf{\Lambda}^{\frac{1}{2}} \mathbf{\Phi}^\top \mathbf{y}}_{\mathbb{E}(\mathbf{w}|\mathbf{y})} \\ \mathbb{V}(f_*) &= \boldsymbol{\phi}_* \mathbf{\Lambda}^{\frac{1}{2}} \underbrace{\sigma_y^2 (\mathbf{\Phi}^\top \mathbf{\Lambda} \mathbf{\Phi} + \sigma_y^2 \mathbf{I}_M)^{-1}}_{\mathbb{V}(\mathbf{w}|\mathbf{y})} \mathbf{\Lambda}^{\frac{1}{2}} \boldsymbol{\phi}_*^\top, \end{aligned} \tag{16}$$

Combining (15) with (10), in our method we consider the model

$$\mathbf{y} = \mathbf{\Phi}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{W}_{\setminus d}\mathbf{w}^{(d)} + \boldsymbol{\epsilon}, \;\; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_y^2 \mathbf{I}_N), \tag{17}$$

where $\mathbf{W}_{\setminus d}$ is deterministic and $\mathbf{w}^{(d)}$ has a prior given by

$$\mathbf{w}^{(d)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{R_d M_d R_{d+1}}). \tag{18}$$

The matrix $\mathbf{\Phi}$ is given in the format (11) and the diagonal matrix is given as a Kronecker product over dimensions

$$\mathbf{\Lambda} = \mathbf{\Lambda}^{(1)} \otimes \mathbf{\Lambda}^{(2)} \otimes \cdots \otimes \mathbf{\Lambda}^{(D)}, \tag{19}$$

where each entry of $\mathbf{\Lambda}^{(d)}$ is computed with (6).

In order to approximate a posterior distribution for the weights $\mathbf{w} \mid \mathbf{y}$ and do predictions for unseen input locations, our proposed algorithm, summarized in Alg. 1,

consists in two consecutive steps. First, we compute $\mathbf{W}_{\backslash d}$, a projection matrix that projects the weight vector into a smaller subspace, where we compute the posterior distribution of $\mathbf{w}^{(d)} \mid \mathbf{y}$. Second, $\mathbf{w}^{(d)} \mid \mathbf{y}$ is projected to a posterior distribution $\mathbf{w} \mid \mathbf{y}$, to enable predictions for unseen inputs.

More specifically, the projection matrix $\mathbf{W}_{\backslash d}$ is computed with the regularized ALS (line 1 of Alg. 1). After a TT representation of the weight vector $\mathbf{w}$ with TT-cores $\boldsymbol{\mathcal{W}}^{(1)}, \boldsymbol{\mathcal{W}}^{(2)}, ..., \boldsymbol{\mathcal{W}}^{(D)}$ is found, $\mathbf{W}_{\backslash d}$ is computed from all the TT-cores except the dth, such that it is $\mathbf{w} = \mathbf{W}_{\backslash d}\mathbf{w}^{(d)}$.

Exploiting the TT structure of $\mathbf{w}$ as well as the Khatri-Rao structure of $\boldsymbol{\Phi}$ and the Kronecker structure in $\boldsymbol{\Lambda}$, the matrix-matrix-multiplications $\boldsymbol{\Phi}\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{W}_{\backslash d}$ in (13) are performed without explicitly constructing the matrices. Instead, the multiplication is performed in an efficient way, for each dimension separately.

After the computation of the projection matrix $\mathbf{W}_{\backslash d}$, a Bayesian update to compute $\mathbf{w}^{(d)} \mid \mathbf{y}$ is performed (line 2 of Alg. 1). Considering the prior distribution given in (18) the mean and covariance of the posterior distribution $\mathbf{w}^{(d)} \mid \mathbf{y}$ is given by

$$\begin{aligned}\mathbb{V}(\mathbf{w}^{(d)} \mid \mathbf{y}) &= \sigma_y^2 \left[\mathbf{W}_{\backslash d}^\top \boldsymbol{\Phi}^\top \boldsymbol{\Lambda} \boldsymbol{\Phi} \mathbf{W}_{\backslash d} + \sigma_y^2 \mathbf{I}_{R_d M_d R_{d+1}}\right]^{-1} \\ \mathbb{E}(\mathbf{w}^{(d)} \mid \mathbf{y}) &= \mathbb{V}(\mathbf{w}^{(d)} \mid \mathbf{y}) \sigma_y^{-2} \mathbf{W}_{\backslash d}^\top \boldsymbol{\Lambda}^{\frac{1}{2}} \boldsymbol{\Phi}^\top \mathbf{y}.\end{aligned} \tag{20}$$

The computation of $\mathbf{W}_{\backslash d}^\top \boldsymbol{\Phi}^\top \boldsymbol{\Lambda} \boldsymbol{\Phi} \mathbf{W}_{\backslash d}$ in (20) has a computational complexity of $\mathcal{O}(R^4 M_d^2 N)$, where R denotes the maximum rank of the TT and assuming $M_1 = M_2 = \cdots = M_D$. The complexity depends on R^4, as a consequence the ranks of the TT need to be small to achieve a significant speed-up. Since the choice of the ranks also influences the accuracy of the approximation, a trade-off needs to be made.

After computing the posterior distribution $\mathbf{w}^{(d)} \mid \mathbf{y}$, the projection matrix is used to project it to $\mathbf{w} \mid \mathbf{y}$ (line 3 of Alg. 1) with

$$\begin{aligned}\mathbb{E}(\mathbf{w} \mid \mathbf{y}) &= \mathbf{W}_{\backslash d} \mathbb{E}(\mathbf{w}^{(d)} \mid \mathbf{y}) \\ \mathbb{V}(\mathbf{w} \mid \mathbf{y}) &= \mathbf{W}_{\backslash d} \mathbb{V}(\mathbf{w}^{(d)} \mid \mathbf{y}) \mathbf{W}_{\backslash d}^\top.\end{aligned} \tag{21}$$

Since $\mathbf{W}_{\backslash d}$ is a tall matrix and $\mathbb{V}(\mathbf{w}^{(d)} \mid \mathbf{y})$ is square, it can be seen that the posterior covariance $\mathbb{V}(\mathbf{w} \mid \mathbf{y})$ is approximated as a low-rank matrix, where the rank corresponds to the number of elements in $\mathbf{w}^{(d)}$.

Finally, the obtained mean and covariance for $\mathbf{w} \mid \mathbf{y}$ are used to make predictions for unseen input locations (line 4 of Alg. 1) with

$$\begin{aligned}\mathbb{E}(f_*) &= \boldsymbol{\phi}_* \boldsymbol{\Lambda}^{\frac{1}{2}} \mathbb{E}(\mathbf{w} \mid \mathbf{y}) \\ \mathbb{V}(f_*) &= \boldsymbol{\phi}_* \boldsymbol{\Lambda}^{\frac{1}{2}} \mathbb{V}(\mathbf{w} \mid \mathbf{y}) \boldsymbol{\Lambda}^{\frac{1}{2}} \boldsymbol{\phi}_*^\top.\end{aligned} \tag{22}$$

In practice, $\boldsymbol{\phi}_* \boldsymbol{\Lambda}^{\frac{1}{2}}$ is not computed explicitly. Instead, the product $\boldsymbol{\phi}_* \boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{W}_{\backslash d}$ is computed efficiently exploiting the structure of the matrices, resulting in a vector of size $1 \times R_d M_d R_{d+1}$.

Algorithm 1 Gaussian process regression with projected basis functions

Require: Collections of basis functions $\boldsymbol{\Phi}$ and $\boldsymbol{\phi}_*$, matrices $\boldsymbol{\Lambda}^{(1)}$,...,$\boldsymbol{\Lambda}^{(D)}$, measurements $\mathbf{y}$, noise variance σ_y^2
Ensure: Predictive distribution $f_* \mid \mathbf{y}$
1: Compute $\boldsymbol{\mathcal{W}}^{(1)}, \boldsymbol{\mathcal{W}}^{(2)}, ..., \boldsymbol{\mathcal{W}}^{(D)}$ with ALS, then $\mathbf{W}_{\backslash d}$ from $\boldsymbol{\mathcal{W}}^{(1)}, ..., \boldsymbol{\mathcal{W}}^{(d-1)}, \boldsymbol{\mathcal{W}}^{(d+1)}, ..., \boldsymbol{\mathcal{W}}^{(D)}$.
2: Compute $\mathbf{w}^{(d)} \mid \mathbf{y}$ with Bayesian inference in projected subspace with equation (20).
3: Project $\mathbf{w}^{(d)} \mid \mathbf{y}$ to $\mathbf{w} \mid \mathbf{y}$ with (21).
4: Predict in unseen location using (22).

4. NUMERICAL EXPERIMENTS

All simulations (section 4.1 and 4.2) are performed on a Lenovo computer with processor Intel(R) Core(TM) i7-8650U CPU @ 2.11 GHz and 16GB of RAM. The experiment in section 4.3 is performed on an AMD EPYC 7252 8-Core Processor 3.10 GHz with 256 GB of RAM. The implementation of the method, simulations, and experiments can be found at `https://github.com/clarazen/ProjectedBasisFunctions` commit `a03b47b`.

For all simulations, synthetic data is generated with the model (17). The matrix $\boldsymbol{\Phi}$ is computed as (11) with (4) and an input that is specified for each experiment in the corresponding subsection. The matrix $\mathbf{W}_{\backslash d}$, which is usually computed in the first step of Alg. 1, for simplicity, is initialized randomly, then its columns are orthogonalized. The size of $\mathbf{W}_{\backslash d}$, i.e. $M \times RM_dR$, depends on both d, specified in the subsections of the experiments, as well as on the TT-ranks which are assumed to be all the same $R = R_1 = \cdots = R_D$.

4.1 Basis function analysis

In our first simulation, we investigate, how the dominant basis functions computed in Solin and Särkkä (2020) compare visually to the same basis functions that are projected to a smaller subspace with Alg. 1. In this context, we generate $N = 400$ input locations on a regular 2D grid confined by $[-1, 1] \times [-1, 1]$ using the hyperparameters $\ell^2 = 0.01$ and $\sigma_f^2 = 1.0$ and $R = 5$. Then, we compute $\boldsymbol{\Phi}\boldsymbol{\Lambda}^{\frac{1}{2}} \in \mathbb{R}^{N \times M_1 M_2}$ with $M_d = 40$. Figure 2 shows the leading 32 of the 1600 basis functions in descending order. As can be seen in the subfigures, the dominating basis functions are low-frequent with an increasing frequency the lower the eigenvalue to the corresponding basis function. Figure 3 shows the projected basis functions or columns of the matrix $\boldsymbol{\Phi}\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{W}_{\backslash 2}$. Comparing both figures gives an insight into how Algorithm 1 works: The projection changes the shape of the basis functions in a way that it sees fit based on $\mathbf{W}_{\backslash 2}$ which is generally computed from the given data.

4.2 Prediction accuracy on synthetic data

In our second simulation, we investigate how our method performs in terms of prediction accuracy on data generated from (17), and compare it to the full GP, as well as the reduced-rank approach by Solin and Särkkä (2020) (Hilbert-GP). Given an input with $D = 3$ that is randomly

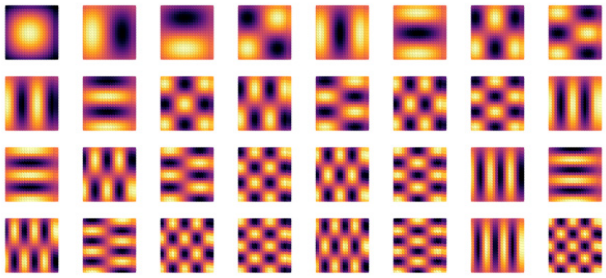

Fig. 2. Leading 32 basis functions computed with Solin and Särkkä (2020), sorted by the descending magnitude of the eigenvalue spectrum.

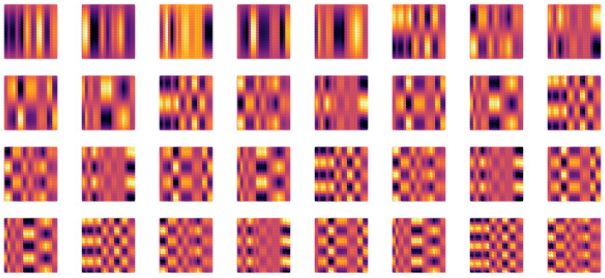

Fig. 3. Subset of $R^2M_2 = 1000$ projected basis functions stored in columns of $\mathbf{\Phi W}_{\backslash d} \in \mathbb{R}^{N\times R^2M_2}$ computed with Alg. 1.

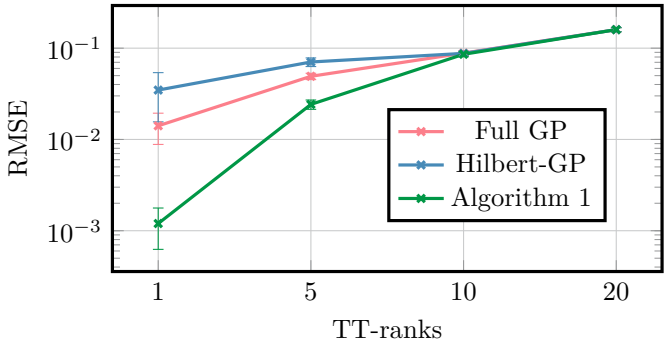

Fig. 4. RMSE on validation data for data generated with model (17) with varying ranks. Mean and standard deviation plotted from 10 different runs.

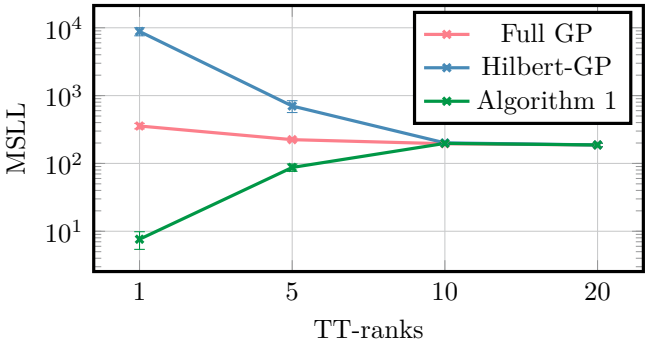

Fig. 5. MSLL for data generated from model (17) with varying ranks. Mean and standard deviation plotted from 10 different parts of the data.

sampled from a box confined by $[-1,1]\times[-1,1]\times[-1,1]$, we sample $N = 4000$ training observations and $N_* = 1000$ validation observations with hyperparameters $\ell^2 = 0.02$ and $\sigma_f^2 = 1.0$. We compute $\mathbf{W}_{\backslash 2} \in \mathbb{R}^{RM_2R}$ with $R = 1, 5, 10, 20$ as described in the introduction of section 4, and choose σ_y^2 such that the signal-to-noise ratio is 10 dB. For the ALS, we choose the same ranks with which we create the data, as we assume them to be known in advance. For the comparison with Hilbert-GP, we choose the dominating R^2M_2 basis functions.

Figures 4 and 5 show the root mean square error (RMSE) and mean standardized log loss (MSLL) on validation data for different TT-ranks ranging from rank-1 to full-rank ($R = 20$). For data sampled from a model where the ranks are chosen to be low, Algorithm 1 performs significantly better than the other two methods. Note, that it is not surprising that Hilbert-GP performs poorly for low ranks since the budget of basis functions to approximate the kernel matrix is very limited. For the limit of full-rank, the result of Algorithm 1 converges to the one of Hilbert-GP. This is because in the full-rank case, the size of $\mathbf{w}^{(2)}$ is the same as $\mathbf{w}$, thus $\mathbf{W}_{\backslash 2}$ is square. This case is of course not useful in practice since it does not offer any computational gain compared to Hilbert-GP.

4.3 Inverse dynamics of a robotic arm

In this experiment, we show that the proposed method works for a real-life data set with a large number of input dimensions, as well as a large number of data points. We use the benchmark industrial robot data set (Weigand et al., 2022), which consists of $N = 39\,988$ training points and 3636 test points. To compute the inverse dynamics of the robotic arm, an 18-dimensional input space (position, velocity, and acceleration for 6 joints) is given together with a 6-dimensional output which are the motor torques for 6 joints. We consider just one output in our experiment. For Hilbert-GP, we use $M = N = 39\,988$ and for our method, we use $R = 5$ and $M_d = 20$ basis functions per dimension, thus $M = 20^{18}$. Table 1 summarizes the required time for training and prediction on the validation data, as well as the RMSE and MSLL for all used methods. In addition, Fig. 6 shows the RMSE for the first 800 measurements of the validation data set. Regarding the RMSE, our method performs better than full GP and Hilbert-GP and requires less time. A possible explanation for this is that the low-rank assumption for the model weights fits the data better than the full GP model. Regarding the MSLL, our method performs significantly worse than the full GP. As discussed in section 3, through the projection, the posterior covariance matrix of the weights is approximated by a low-rank matrix with a rank equal to $R^2M_d = 500$. This approximation seems not to be sufficient in this case.

Table 1. Time, RMSE, and MSLL for val. data.

	Time [s]	RMSE	MSLL
Full GP	1400	1.81	7926.01
Hilbert-GP	388	1.42	17805.6
Our method	57	1.13	11621.18

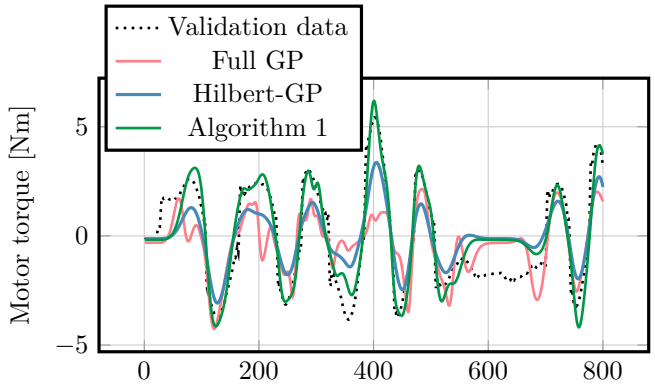

Fig. 6. Predicted torques for one joint.

5. CONCLUSION

In this work, we presented a method for approximate GP regression that uses TNs to project basis functions to a smaller subspace to exploit efficient computations. We computed a projection matrix in TN format, that projects a parametric model to a smaller subspace, where the distribution of a smaller weight vector is computed. Finally, we make predictions by projecting the small weight vector back to the original space. A direction of future work is to explore options to speed up hyperparameter optimization with TNs. In the reduced-rank GP framework by Solin and Särkkä (2020), the approximation leads to inverting a matrix of size $M \times M$ during the hyperparameter optimization. Inverting that matrix in TN format efficiently is not straightforward. Additional future work is to investigate the choice of the ranks of the TN for given data, as well as which of the TT-cores should be chosen for the Bayesian inference. A limitation of this work is that the computation of the projection matrix is done in a deterministic way. This causes $\mathbb{V}(\mathbf{w} \mid \mathbf{y})$ to be underestimated. An idea is to apply the ALS in a Bayesian framework (Menzen et al., 2022) to compute a mean and a covariance for all TT-cores and combine the covariance information with, e.g., the unscented transform.

ACKNOWLEDGEMENTS

This publication is part of the project “Sensor Fusion For Indoor localisation Using The Magnetic Field” with project number 18213 of the research program Veni which is (partly) financed by the Dutch Research Council (NWO).

REFERENCES

Batselier, K., Chen, Z., and Wong, N. (2017). Tensor network alternating linear scheme for MIMO Volterra system identification. *Automatica*, 84, 26–35.

Berntorp, K. (2021). Online bayesian inference and learning of gaussian-process state–space models. *Automatica*, 129, 109613.

Calandra, R., Peters, J., Rasmussen, C.E., and Deisenroth, M.P. (2016). Manifold Gaussian processes for regression. In *International Joint Conference on Neural Networks (IJCNN)*, 3338–3345. IEEE.

Chiuso, A. and Pillonetto, G. (2019). System identification: A machine learning perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, 2, 281–304.

Hewing, L., Kabzan, J., and Zeilinger, M.N. (2019). Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6), 2736–2743.

Holtz, S., Rohwedder, T., and Schneider, R. (2012). The alternating linear scheme for tensor optimization in the tensor train format. *SIAM Journal on Scientific Computing*, 34(2), A683–A713.

Izmailov, P., Novikov, A., and Kropotov, D. (2018). Scalable Gaussian processes with billions of inducing inputs via tensor train decomposition. In *International Conference on Artificial Intelligence and Statistics*, 726–735. PMLR.

Kirstein, M., Sommer, D., and Eigel, M. (2022). Tensor-train kernel learning for gaussian processes. In U. Johansson, H. Boström, K. An Nguyen, Z. Luo, and L. Carlsson (eds.), *Proceedings of the Eleventh Symposium on Conformal and Probabilistic Prediction with Applications*, volume 179 of *Proceedings of Machine Learning Research*, 253–272. PMLR.

Kolda, T.G. and Bader, B.W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3), 455–500.

Konstantinidis, K., Li, S., and Mandic, D.P. (2021). Kernel learning with tensor networks. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2920–2924. IEEE.

Liu, H., Ong, Y.S., Shen, X., and Cai, J. (2020). When Gaussian process meets big data: A review of scalable GPs. *IEEE transactions on neural networks and learning systems*, 31(11), 4405–4423.

Liu, M., Chowdhary, G., Da Silva, B.C., Liu, S.Y., and How, J.P. (2018). Gaussian processes for learning and control: A tutorial with examples. *IEEE Control Systems Magazine*, 38(5), 53–86.

Menzen, C., Kok, M., and Batselier, K. (2022). Alternating linear scheme in a Bayesian framework for low-rank tensor approximation. *SIAM Journal on Scientific Computing*, 44(3), A1116–A1144.

Oseledets, I.V. (2011). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5), 2295–2317.

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

Rasmussen, C.E. and Williams, C.K.I. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

Solin, A. and Särkkä, S. (2020). Hilbert space methods for reduced-rank Gaussian process regression. *Statistics and Computing*, 30, 419–446.

Stoudenmire, E. and Schwab, D.J. (2016). Supervised learning with tensor networks. *Advances in Neural Information Processing Systems*, 29.

Weigand, J., Götz, J., Ulmen, J., and Ruskowski, M. (2022). 6th Edition of the Workshop on Nonlinear System Identification Benchmarks, Leuven, Belgium. URL `http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-67317`.

Wesel, F. and Batselier, K. (2021). Large-scale learning with Fourier features and tensor decompositions. *Advances in Neural Information Processing Systems*, 34.