Large-Scale Learning with Fourier Features and Tensor Decompositions

Anonymous Author(s)
Affiliation
Address
email

Abstract

Random Fourier features provide a way to tackle large-scale machine learning problems with kernel methods. Their slow Monte Carlo convergence rate has motivated the research of *deterministic* Fourier features whose approximation error decreases exponentially with the number of frequencies. However, due to their tensor product structure these methods suffer heavily from the curse of dimensionality, limiting their applicability to two or three-dimensional scenarios. In our approach we overcome said curse of dimensionality by exploiting the tensor product structure of deterministic Fourier features, which enables us to represent the model parameters as a low-rank tensor decomposition. We derive a monotonically converging block coordinate descent algorithm with linear complexity in both the sample size and the dimensionality of the inputs for a regularized squared loss function, allowing to learn a parsimonious model in decomposed form using deterministic Fourier features. We demonstrate by means of numerical experiments how our low-rank tensor approach obtains the same performance of the corresponding nonparametric model, consistently outperforming random Fourier features.

1 Introduction

Kernel methods such as Support Vector Machines and Gaussian Processes are commonly used to tackle problems such as classification, regression and dimensionality reduction. Since they can be universal function approximators [12], kernel methods have received renewed attention in the last few years and have shown equivalent or superior performance to Neural Networks [21, 25, 10]. The main idea behind kernel methods is to lift the data into a higher-dimensional (or even infinite-dimensional) Reproducing Kernel Hilbert Space $\mathcal{H}$ by means of a *feature map* $\boldsymbol{\phi}(\cdot) : \mathcal{X} \to \mathcal{H}$. Considering then the pairwise similarities between the mapped data allows to tackle problems which are highly nonlinear in the original sample space $\mathcal{X}$, but linear in $\mathcal{H}$. This can be done equivalently by considering a *kernel function* $k(\cdot,\cdot) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ such that $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x'}) \rangle = k(\boldsymbol{x}, \boldsymbol{x'})$ and performing thus said mapping implicitly. Although effective at learning nonlinear patterns in the data, kernel methods are known to scale poorly as the number of data points N increases. For example, when considering Kernel Ridge Regression (KRR), Gaussian Process Regression (GPR) [32] or Least Squares Support Vector Machine (LS-SVM) [40, 41] training usually consist in inverting the *Gram matrix* $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, which encodes the pairwise relation between all data. As a consequence, the associated storage complexity is $\mathcal{O}\left(N^2\right)$ and the computational complexity is $\mathcal{O}\left(N^3\right)$, rendering these methods unfeasible for large data. In order to lower the computational load, *data dependent* methods approximate the kernel function by means of M data dependent basis functions. Due to their reduced-rank formulation, the computational complexity is reduced to $\mathcal{O}\left(NM^2\right)$ for large data. However, e.g. for the Nyström method, the convergence rate is only of $\mathcal{O}\left(1/\sqrt{M}\right)$ [7] limiting its effectiveness. As the name suggests, *data independent* methods approximate

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

the kernel function by M data independent basis functions. A good example is the celebrated Random Fourier Features approach by Rahimi and Recht [30], where the authors propose for stationary kernels a low-dimensional *random* mapping $\boldsymbol{z}(\cdot) : \mathbb{R}^D \to \mathbb{R}^M$ such that

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x'}) \rangle \approx \langle \boldsymbol{z}(\boldsymbol{x}), \boldsymbol{z}(\boldsymbol{x'}) \rangle . \tag{1}$$

As in the case of data dependent methods, the reduced-rank formulation allows for a computational complexity of $\mathcal{O}\left(NM^2\right)$. Probabilistic error bounds on the approximation are provided which result in a convergence rate of $\mathcal{O}\left(1/\sqrt{M}\right)$, which is again the Monte Carlo rate.

Improvements in this sense were achieved by considering *deterministic* features resulting from dense quadrature methods [5, 24, 35] and kernel eigenfunctions [15, 38]. These methods are able to achieve *exponential* convergence. For a D-dimensional input space these methods take the tensor product of D vectors, resulting in an exponential increase in the number of basis functions and weights, effectively limiting the applicability of deterministic features to low-dimensional data. In this work we consider deterministic features. In order to take advantage of the tensor product structure which arises when mapping the inputs to $\mathcal{H}$, we represent the weights as a low-rank tensor. This allows us to *learn* the inter-modal relations in the tensor product of (low-dimensional) Hilbert spaces, avoiding the exponential computational and storage complexities in D. In this way we are able to obtain a *linear* computational complexity in both the sample size N *and* input dimension D during training, without having to resort to using sparse grids or additive kernels.

The main contribution of this work is in lifting the curse of dimensionality affecting deterministic Fourier Features by modeling the weights as a low-rank tensor. This enables efficiently solving large-scale and high-dimensional kernel ridge regression problems with an exponential convergence of the kernel function approximation. We test the resulting algorithm under regularized squared loss for regression and classification.

2 Related Work

Fourier Features (FF) are a collection of data independent methods that leverage Bochner's theorem [34] from harmonic analysis to approximate stationary kernels by numerical integration of their spectral density $p(\cdot)$:

$$k(\boldsymbol{x}, \boldsymbol{x'}) \overset{\text{Stationarity}}{:=} k(\boldsymbol{x} - \boldsymbol{x'}) \overset{\text{Bochner}}{=} \int p(\boldsymbol{\omega}) \exp\left(\langle i\boldsymbol{\omega}, (\boldsymbol{x} - \boldsymbol{x'}) \rangle\right) d\boldsymbol{\omega} \overset{\text{FF}}{\approx} \langle z(\boldsymbol{x}), z(\boldsymbol{x'}) \rangle . \tag{2}$$

Rahimi and Recht [30] proposed to approximate the integral by Monte Carlo integration i.e. by drawing M random frequencies $\boldsymbol{\omega} \sim p(\cdot)$. In their work they show how the method converges uniformly at the Monte Carlo rate. See [22] for an overview of random FF. In order to achieve faster convergence and a lower sample complexity, a multitude of approaches that rely on deterministic numerical quadrature of the Fourier integral were developed. These methods generally consider kernels whose spectral density factors over D in the frequency domain, which enables in turn to factor the Fourier integral. The resulting deterministic Fourier features are then the tensor product of D one-dimensional deterministic features.

For example, in [5] the authors give an analysis of the sample complexity of features resulting from dense *Gaussian Quadrature* (GQ). In [24] the authors present a similar quadrature approach for kernels whose spectral density factors over the dimensions of the inputs. They provide an explicit construction for their *Quadrature Fourier Features* relying on a dense Cartesian grid, and note that their method, as well as GQ, can achieve exponentially fast convergence in the total number of frequencies M [24, Theorem 1]. To avoid the curse of dimensionality, they make use of additive modeling of kernels. *Variational Fourier Features* (VFF) [15] are derived probabilistically in a one-dimensional setting for Matérn kernels by projecting a Gaussian Process onto a set of Fourier basis. An extension to multiple dimensions when considering product kernels is then achieved by taking the tensor product of the one-dimensional features, incurring however again in exponentially rising computational costs in D. In what they call *Hilbert-GP* [38] the authors diagonalize stationary kernels in terms of the eigenvalues and eigenfunctions of the Laplace operator with Dirichlet boundary conditions. Due to the multiplicative structure of the eigenfunctions of the Laplace operator, the complexity of the basis function increases exponentially in D when one considers product kernels. Like with other deterministic approximations, exponential convergence can be achieved [38, Theorem 8].

Tensor decompositions have been used extensively in machine learning in order to benefit from structure in data [37], in particular to obtain an efficient representation of the model parameters. In the context of Gaussian Process Regression, the tensor product structure which arises when the inputs are located on a Cartesian grid have been exploited to speedup inference [11]. In their variational inference framework, [18] proposed to parameterize the posterior mean of a Gaussian Process by means of a tensor decomposition in order to exploit the tensor product structure which arises when interpolating the kernel function. In the context of neural networks, [26] proposed to represent the weights in deep neural networks as a tensor decomposition to speed up training. A similar approach was carried out in case of recurrent neural network [46] [42].

Tensor decompositions have also been used to learn from simple features. Motivated by spin vectors in quantum mechanics, [39] considered trigonometric feature maps which they argue induce uninformative kernels. They then proceed to tackle multi-class image classification by modeling the model weights as a tensor decomposition. On a similar note, [27] and [1] leverage the tensor product structure of polynomial mappings to induce a polynomial kernel and consider model parameters in decomposed form. While these existing approaches are able to overcome the curse of dimensionality affecting polynomial feature maps by modeling the parameter tensor as a tensor network, they consider simple and empirical feature maps which induce uninformative kernels. In the following section we show how deterministic Fourier features can be used for supervised learning in large and high-dimensional scenarios.

3 Learning with deterministic Fourier features

3.1 Notation

All tensor operations and mappings can be formulated over both the complex as the real field. For the remainder of this article real-valued mappings will be considered. Vectors and matrices are denoted by boldface lowercase and uppercase letters, respectively. Higher-order tensors are denoted by boldface calligraphic letters. Tensor entries are always written as lightface lowercase letters with a subscript index notation. For example, element i_1, i_2, i_3 of the third-order tensor $\boldsymbol{\mathcal{A}}$ is written as $a_{i_1 i_2 i_3}$. The vectorization $\operatorname{vec}(\boldsymbol{\mathcal{A}})$ of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ is the vector such that

$$\operatorname{vec}(\boldsymbol{\mathcal{A}})_{[i_1 i_2 \cdots i_D]} = a_{i_1 i_2 \cdots i_D},$$

where the relationship between the linear index $[i_1 i_2 \cdots i_D]$ and $i_1, i_2, \ldots, i_D$ is given by

$$[i_1 i_2 \cdots i_D] = i_1 + \sum_{d=2}^{D} (i_d - 1) \prod_{k=1}^{d-1} I_k.$$

The Frobenius inner product between tensors $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}$ is defined as

$$\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle_{\mathrm{F}} = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_D} a_{i_1 i_2 \cdots i_D} b_{i_1 i_2 \cdots i_D} = \operatorname{vec}(\boldsymbol{\mathcal{A}})^T \operatorname{vec}(\boldsymbol{\mathcal{B}}).$$

Depending on the context the symbol $\otimes$ either denotes the tensor outer product or the matrix Kronecker product. The Khatri-Rao product $\boldsymbol{A} \circledast \boldsymbol{B}$ of the matrices $\boldsymbol{A} \in \mathbb{R}^{N \times R}$ and $\boldsymbol{B} \in \mathbb{R}^{M \times R}$ is the $NM \times R$ matrix that is obtained by taking the column-wise Kronecker product.

3.2 The model

In this article we assume kernelized models of the form

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle. \tag{3}$$

Learning such a kernelized model consists in finding a hyperplane $\boldsymbol{w} \in \mathcal{H}$ that minimizes a symmetric loss function $L(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$ with the mapped data $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathcal{H}$ in a high and possibly infinite-dimensional space $\mathcal{H}$ such that:

$$\min_{\boldsymbol{w}} \quad \sum_{n=1}^{N} L(y_n, f(\boldsymbol{x}_n)) + \lambda \langle \boldsymbol{w}, \boldsymbol{w} \rangle_{\mathrm{p}}. \tag{4}$$

Here $\lambda \langle \boldsymbol{w}, \boldsymbol{w} \rangle_{\mathrm{p}}$ is a p-th norm regularization term, with $\lambda \geq 0$ being the regularization constant. A variety of machine learning algorithms arise when considering different combinations of loss functions and regularization terms. For example, considering a squared loss and $p = 2$ leads to Kernel Ridge Regression [41], while considering a hinge loss and $p = 1$ leads to SVM [4], and so on. Furthermore, applying the *kernel trick* when considering $\boldsymbol{\phi}(\cdot)$ recovers the nonparametric formulation which, depending on the choice of kernel, enables to perform inference using infinitely many basis functions with a computational complexity of $\mathcal{O}(N^3)$. Considering a low-dimensional finite mapping $\boldsymbol{z}(\cdot) : \mathcal{X} \to \mathbb{R}^M$ such as Random Fourier Features or the Nyström method leads to computational savings and a computational complexity of $\mathcal{O}\left(NM^2\right)$. However, as already discussed, these low-dimensional random mappings converge at the slow Monte Carlo rate, motivating our approach.

In order to approximate the kernel function with fast *exponential* convergence, we consider deterministic, finite-dimensional features which are the tensor product of D vectors. For a given D-dimensional input point $\boldsymbol{x}$ we define the deterministic feature mapping $\boldsymbol{\mathcal{Z}}(\cdot) : \mathbb{R}^D \to \mathbb{R}^{\hat{M}_1} \otimes \mathbb{R}^{\hat{M}_2} \otimes \cdots \otimes \mathbb{R}^{\hat{M}_D}$ as

$$\boldsymbol{\mathcal{Z}}(\boldsymbol{x}) = \boldsymbol{z}^{(1)}(x_1) \otimes \boldsymbol{z}^{(2)}(x_2) \cdots \otimes \boldsymbol{z}^{(D)}(x_D), \tag{5}$$

where $\boldsymbol{z}^{(d)}$ is a deterministic mapping applied to the d-th dimension. The dimension of the feature space is denoted

$$M = \prod_{d=1}^{D} \hat{M}_d.$$

Without loss of generality it is from here on assumed that $\hat{M}_1 = \cdots = \hat{M}_D = \hat{M}$ such that $M = \hat{M}^D$. The mapping $\boldsymbol{\mathcal{Z}}(\cdot)$ encompasses for instance the mappings derived by quadrature [5, 24, 35] and by projection [15, 38]. Note that these mappings cover many popular kernels such as the Gaussian and Matérn kernels. To give a concrete example, in the framework of A. Solin and S. Särkkä [38, Equation 60], for input data centered in a hyperbox $\mathcal{X} = [-U_1, U_1] \otimes \cdots \otimes [-U_D, U_D]$, the Gaussian kernel is approximated by means of D tensor products of $\hat{M}$ weighted sinusoidal basis functions with frequencies lying on a harmonic scale such that:

$$\boldsymbol{z}^{(d)}(x_d)_{i_d} = \frac{1}{\sqrt{U_d}}\, p\left(\frac{\pi i_d}{2U_d}\right) \sin\left(\frac{\pi i_d (x_d + U_d)}{2U_d}\right). \tag{6}$$

Here $p(\cdot)$ is the spectral density of the Gaussian kernel with one-dimensional inputs, which is known in closed-form [32, page 83]. This deterministic mapping then approximates the Gaussian kernel function [38, Equation 59] such that

$$k(\boldsymbol{x}, \boldsymbol{x'}) \approx \langle \mathrm{vec}(\boldsymbol{\mathcal{Z}}(\boldsymbol{x})), \mathrm{vec}(\boldsymbol{\mathcal{Z}}(\boldsymbol{x'})) \rangle = \langle \boldsymbol{\mathcal{Z}}(\boldsymbol{x}), \boldsymbol{\mathcal{Z}}(\boldsymbol{x'}) \rangle_{\mathrm{F}}$$

and converges exponentially fast in M [38, Theorem 8]. We therefore use $\boldsymbol{\mathcal{Z}}(\cdot)$ instead of $\boldsymbol{\phi}(\cdot)$ in (3) to obtain

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \mathrm{vec}(\boldsymbol{\mathcal{Z}}(\boldsymbol{x})) \rangle = \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{Z}}(\boldsymbol{x}) \rangle_{\mathrm{F}}, \tag{7}$$

where the weight vector $\boldsymbol{w}$ has been reshaped into a D-dimensional tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{\hat{M}_1 \times \hat{M}_2 \times \cdots \times \hat{M}_D}$. Learning the exponential number of model parameters $\boldsymbol{\mathcal{W}}$ in (7) under a hinge loss leads to Support Vector Machines, while considering a squared loss leads to Kernel Ridge Regression:

$$\min_{\boldsymbol{\mathcal{W}}} \sum_{n=1}^{N} \left(y_n - \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{Z}}(\boldsymbol{x}_n) \rangle_{\mathrm{F}}\right)^2 + \lambda \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{W}} \rangle_{\mathrm{F}}. \tag{8}$$

Since the number of elements M in $\boldsymbol{\mathcal{W}}$ and $\boldsymbol{\mathcal{Z}}$ grows exponentially in D, this *primal* approach is advantageous compared to the nonparametric *dual* approach only if $\hat{M}^D \ll N$, limiting it to low-dimensional inputs. In order to lift this curse of dimensionality, we propose to represent and to learn $\boldsymbol{\mathcal{W}}$ directly as a low-rank tensor decomposition. A low-rank tensor decomposition allows us to exploit redundancies in $\boldsymbol{\mathcal{W}}$ in order to obtain a parsimonious model with a storage complexity that scales linearly in both $\hat{M}$ and D. The low-rank structure will, as explicitly shown in the experiments, act as a form of regularization by limiting the total number of degrees of freedom of the model.

3.3 Low-rank tensor decompositions

Tensor decompositions (also called tensor networks) can be seen as generalizations of the Singular Value Decomposition (SVD) of a matrix to tensors [19]. Three common tensor decompositions are the Tucker decomposition [6, 44], the Tensor Train decomposition [28] and the Canonical Polyadic decomposition (CPD) [16, 20], where each of them encompasses different properties of the matrix SVD. In this subsection we briefly discuss these three decompositions and list both their advantages and disadvantages for modeling $\boldsymbol{\mathcal{W}}$ in (8). For a detailed exposition on tensor decompositions we refer the reader to [2] and the references therein. An important property of tensor decompositions is that they can always be written linearly in their components, which implies that applying a block coordinate descent algorithm to solve (8) results in a linear least squares problem.

A rank-R CPD of $\boldsymbol{\mathcal{W}}$ consists of D factor matrices $\boldsymbol{W}^{(d)} \in \mathbb{R}^{\hat{M} \times R}$ and a vector $\boldsymbol{s} \in \mathbb{R}^R$ such that

$$\boldsymbol{w} = \left(\boldsymbol{W}^{(1)} \circledast \boldsymbol{W}^{(2)} \circledast \cdots \circledast \boldsymbol{W}^{(D)}\right) \boldsymbol{s} = \sum_{r=1}^{R} \boldsymbol{w}_r^{(1)} \otimes \boldsymbol{w}_r^{(2)} \otimes \cdots \otimes \boldsymbol{w}_r^{(D)}.$$

Note that the entries of the $\boldsymbol{s}$ vector were absorbed in one of the factor matrices when writing the CPD as a sum of R terms. The CPD has been shown, in contrast to matrix factorizations, to be unique under mild conditions [36]. The storage complexity comes mainly from the D factor matrices and is therefore $\mathcal{O}(R\hat{M}D)$. The Tucker decomposition generalizes the CPD in two ways. First, the Khatri-Rao product is replaced with a Kronecker product. The $\boldsymbol{s}$ vector has to grow in length accordingly from R to R^D. Second, the R-dimension of each of the factor matrices is allowed to vary, resulting in a multi-linear rank $(R_1, R_2, \ldots, R_D)$:

$$\boldsymbol{w} = \left(\boldsymbol{W}^{(1)} \otimes \boldsymbol{W}^{(2)} \otimes \cdots \otimes \boldsymbol{W}^{(D)}\right) \boldsymbol{s}.$$

The Tucker decomposition is inherently non-unique and its storage complexity $\mathcal{O}(R^D)$ is dominated by the $\boldsymbol{s}$ vector, which limits its use to low-dimensional input data. For this reason we do not consider the Tucker decomposition any further in this article.

The Tensor Train (TT) decomposition consists of D third-order tensors $\boldsymbol{\mathcal{W}}^{(d)} \in \mathbb{R}^{R_d \times \hat{M} \times R_{d+1}}$ such that

$$w_{i_1 i_2 \cdots i_D} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{D+1}=1}^{R_{D+1}} w_{r_1 i_1 r_2}^{(1)} w_{r_2 i_2 r_3}^{(2)} \cdots w_{r_D i_D r_{D+1}}^{(D)}. \tag{9}$$

The auxiliary dimensions $R_1, R_2, \ldots, R_{D+1}$ are called the TT-ranks. In order to ensure that the right-hand side of (9) is a scalar, the boundary condition $R_1 = R_{D+1} = 1$ is enforced. The TT decomposition is, just like the Tucker decomposition, non-unique and its storage complexity $R^2\hat{M}D$ is due to the D tensor components $\boldsymbol{\mathcal{W}}^{(d)}$. Considering their storage complexity, both the CPD and TT decomposition are viable candidates to replace $\boldsymbol{\mathcal{W}}$ in (8). The CP-rank R and TT-ranks $R_2, \ldots, R_D$ are additional hyperparameters, which favors the CPD in practice. For the TT, one could choose $R_2 = R_3 = \cdots = R_D$ to reduce the number of additional hyperparameters but this constraint turns out in practice to lead to suboptimal results. For these reasons we limit the discussion of the learning algorithm to the CPD case.

3.4 Tensor learning with deterministic Fourier features

We now wish to minimize the standard regularized squared loss function as in (8) with the additional constraint that the weight tensor has a rank-R CPD structure:

$$\min_{\boldsymbol{\mathcal{W}}} \quad \sum_{n=1}^{N} \left(y_n - \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{Z}}(\boldsymbol{x}_n) \rangle_{\mathrm{F}}\right)^2 + \lambda \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{W}} \rangle_{\mathrm{F}} \tag{10}$$

$$\text{subject to} \quad \text{CP-rank}(\boldsymbol{\mathcal{W}}) = R. \tag{11}$$

Note that if R equals the true CP-rank of the underlying weight tensor then the exact solution of (8) would be obtained. In practice a low-rank solution for $\boldsymbol{\mathcal{W}}$ achieves a sufficiently complex decision boundary that is practically indistinguishable from the full-rank solution, as is demonstrated in the

experiments. Imposing the rank-R reduces the number of unknowns from $\hat{M}^D$ to $R\hat{M}D$ and allows the application of a block coordinate descent algorithm (also known as alternating linear scheme), which is shown to converge monotonically [3, 17, 33]. Each of the factor matrices $\boldsymbol{W}^{(d)}$ is optimized in an iterative fashion while keeping the others fixed. Such a factor matrix update is obtained by solving a linear least squares problem with $\hat{M}D$ unknowns. In what follows we derive the linear problem for the dth factor matrix $\boldsymbol{W}^{(d)}$. The data-fitting term $\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{Z}}(\boldsymbol{x}) \rangle_{\mathrm{F}}$ can be rewritten linearly in terms of the unknown factor matrix as

$$\begin{aligned} \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{Z}}(\boldsymbol{x}) \rangle_{\mathrm{F}} &= \operatorname{vec}\left(\boldsymbol{W}^{(d)}\right)^{\mathrm{T}} \operatorname{vec}\left(\boldsymbol{z}^{(d)} \otimes \left(\boldsymbol{W}^{(1)}\boldsymbol{z}^{(1)} \odot \cdots \odot \boldsymbol{W}^{(D)}\boldsymbol{z}^{(D)}\right)\right) \\ &= \left\langle \operatorname{vec}\left(\boldsymbol{W}^{(d)}\right), \operatorname{vec}\left(\boldsymbol{G}^{d}\right) \right\rangle, \end{aligned} \tag{12}$$

where $\odot$ denotes the Hadamard (element-wise) matrix product and

$$\boldsymbol{G}^{(d)} := \boldsymbol{z}^{(d)} \otimes \left(\boldsymbol{W}^{(1)}\boldsymbol{z}^{(1)} \odot \cdots \odot \boldsymbol{W}^{(D)}\boldsymbol{z}^{(D)}\right).$$

Similarly, for the regularization term:

$$\begin{aligned} \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{W}} \rangle_{\mathrm{F}} &= \operatorname{vec}\left(\boldsymbol{W}^{(d)^{\mathrm{T}}}\boldsymbol{W}^{(d)}\right)^{\mathrm{T}} \operatorname{vec}\left(\boldsymbol{W}^{(1)^{\mathrm{T}}}\boldsymbol{W}^{(1)} \odot \cdots \odot \boldsymbol{W}^{(D)^{\mathrm{T}}}\boldsymbol{W}^{(D)}\right) \\ &= \left\langle \operatorname{vec}\left(\boldsymbol{W}^{(d)^{\mathrm{T}}}\boldsymbol{W}^{(d)}\right), \operatorname{vec}\left(\boldsymbol{H}^{(d)}\right) \right\rangle. \end{aligned} \tag{13}$$

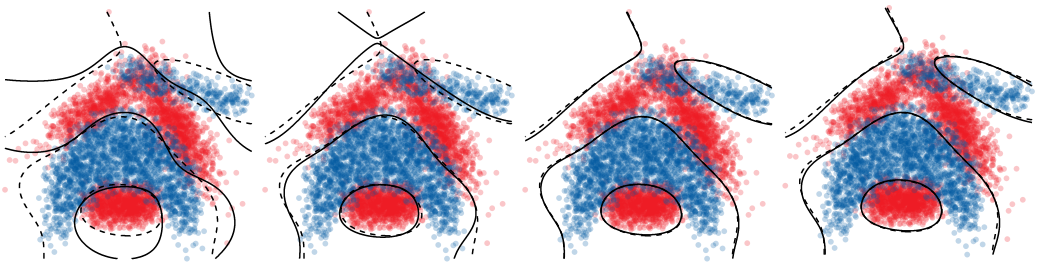
Here $\boldsymbol{H}^{(d)} := \left(\boldsymbol{W}^{(1)^{\mathrm{T}}}\boldsymbol{W}^{(1)} \odot \cdots \odot \boldsymbol{W}^{(D)^{\mathrm{T}}}\boldsymbol{W}^{(D)}\right)$. Derivations of (12) and (13) can be found in Appendix A. Substitution of the expressions (12) and (13) into (10) and (11) leads to a parametric ridge regression problem for $\operatorname{vec}\left(\boldsymbol{W}^{(d)}\right)$:

$$\min_{\operatorname{vec}\left(\boldsymbol{W}^{(d)}\right)} \sum_{n=1}^{N} \left(y_n - \left\langle \operatorname{vec}\left(\boldsymbol{W}^{(d)}\right), \operatorname{vec}\left(\boldsymbol{G}^{d}\right) \right\rangle\right)^2 + \lambda \left\langle \operatorname{vec}\left(\boldsymbol{W}^{(d)^{\mathrm{T}}}\boldsymbol{W}^{(d)}\right), \operatorname{vec}\left(\boldsymbol{H}^{(d)}\right) \right\rangle \tag{14}$$

Its solution can be efficiently computed exactly by applying the matrix inversion lemma and then solving a small system of linear equations with $\hat{M}R$ unknowns, requiring $N\left(\hat{M}R\right)^2 + \left(\hat{M}R\right)^3$ operations. Since we are solving a non-convex optimization problem that consists of a series of convex and exactly solvable sub-problems, our algorithm is monotonically decreasing, and, although it is not guaranteed to converge to the global optimum, it is the standard choice when dealing with tensors [3, 19]. The total computational complexity of the algorithm when $N \gg \hat{M}R$ is then $\mathcal{O}(ND\hat{M}^2R^2)$, rendering it suitable for problems which are large in both N and D, provided that R and $\hat{M}$ are small. The choice of $\hat{M}$ and the additional hyperparameter R which we introduce, are directly linked with the available computational budget. One should in fact choose $\hat{M}$ so that the model has access to a sufficiently complex set of basis functions to learn from. In practice we notice that at most $\hat{M} = 40$ basis functions per dimension suffice for most problems. R can then be fixed accordingly in order to match the computational budget at hand. As we will show in the next section, learning is possible with small values of $\hat{M}$ and R.

4 Numerical experiments

We have implemented our Tensor-Kernel Ridge Regression (T-KRR) method in Mathworks MATLAB 2021a (Update 1) [23] and tested it on several regression and classification problems. Our implementation can be freely downloaded from `https://github.com/NeurIPS2021Submission/Code` and allows reproduction of all experiments in this section. In our implementation we avoid constructing $\boldsymbol{G}^{(d)}$ and $\boldsymbol{H}^{(d)}$ from scratch at every iteration by updating their components. We further speedup our implementation by considering only the diagonal of the regularization term $\boldsymbol{H}^{(d)}$. All experiments were run on an Intel(R) Core i7-10610U CPU running at 1.80GHz of a Dell Inc. Latitude 7410 laptop with 16GB of RAM. In all our experiments the Gaussian kernel $k\left(\boldsymbol{x}, \boldsymbol{x'}\right) = \exp\left(-||\boldsymbol{x}-\boldsymbol{x'}||_2^2/2l^2\right)$ was used which is approximated by means of the *Hilbert-GP* [38] mapping of (6). In all experiments inputs were scaled to lie in a D-dimensional unit hypercube. When dealing with regression, the responses were standardized around the mean, while when considering binary classification inference



(a) $R = 2$ (b) $R = 4$ (c) $R = 6$ (d) $R = \hat{M} = 12$

Figure 1: Classification boundary of the two-dimensional banana dataset for increasing CP-ranks. In the last plot the chosen CP-rank matches the true matrix rank. The dashed line is the Hilbert-GP, while the full line is T-KRR.

was done by looking at the sign of the model response (LS-SVM [40]). One sweep of our T-KRR algorithm is defined as updating factor matrices in the order $1 \rightarrow D$ and then back from $D \rightarrow 1$. All initial factor matrices were initialized with standard normal numbers and normalized by dividing all entries with their Frobenius norm. For all experiments the number of sweeps of T-KRR algorithm are set to 10. In the following three experiments it is shown how our proposed T-KRR algorithm is stable and recovers the full KRR estimate in case of low number of frequencies $\hat{M}$ and low rank R, consistently outperforming RFF. Finally, our model exhibits very competitive performance on large-scale problems when compared with other kernel methods.

4.1 Banana classification

The banana dataset is a two-dimensional classification problem [14] consisting of $N = 5300$ datapoints. Since the data is two-dimensional, we consider $\hat{M} = 12$ frequencies per dimension, which enables us to compare T-KRR with its full-rank counterpart. Furthermore since for tensors of order two (i.e. matrices) the CP-rank is the matrix rank, we can deduce that our approach should recover the underlying *Hilbert-GP* method with $R = \hat{M}$. In this example we fix the kernel hyperparameters of our method and of the Hilbert-GP to $l = 1/2$ and $\lambda = 10^{-5}$ for visualization purposes. In Figure 1 we plot the decision boundary of T-KRR (full line) and of the associated Hilbert-GP (dashed line). We can see that already when $R = 6$ the learned decision boundary is almost identical to the one of Hilbert-GP, meaning that in this example it is possible to obtain a low-rank representation of the model weights. From a computational point of view, we solve a series of linear systems with $\hat{M}R$ unknowns instead of $\hat{M}^D$. However due to the low-dimensionality of the dataset, the computational savings per iteration are very modest, i.e. for $R = 6$ and $\hat{M} = 12$, we solve per iteration a 72×72 linear system as opposed to the one-time solve of a 144×144 linear system in case of Hilbert-GP. As we show in Subsection 4.2, the linear computational complexity in D of our algorithm results in ever-increasing performance benefits as the dimensionality of the problem becomes larger.

4.2 Model performance with baseline

We consider five UCI [8] datasets in order to compare the performance of our model with RFF and the baseline, which we choose to be GPR. For each dataset, we consider 90% of the data for training and the remaining 10% for testing. In particular, in all regression datasets, we first obtain an estimate of the lengthscale of the Gaussian kernel l and regularization term λ by log-marginal likelihood optimization over the whole training set using the GPLM toolbox [31] (FreeBSD License). We subsequently train KRR, RFF and our method with the estimated hyperparameters. In case of classification, we choose the lengthscale l to be the sample mean of the sample standard deviation of our data (which is the default choice in the Matlab Machine Learning Toolbox and scikit-learn [29]) and $\lambda = 10^{-5}$. We repeat this procedure ten times over different random splits and report the sample mean and sample standard deviation of the predictive mean-squared error (MSE) and the misclassification rate for regression and classification respectively. In order to test the approximation capabilities of our approach, we set $\hat{M}$ and R such that $\hat{M}R < N$ in order to benefit from computational

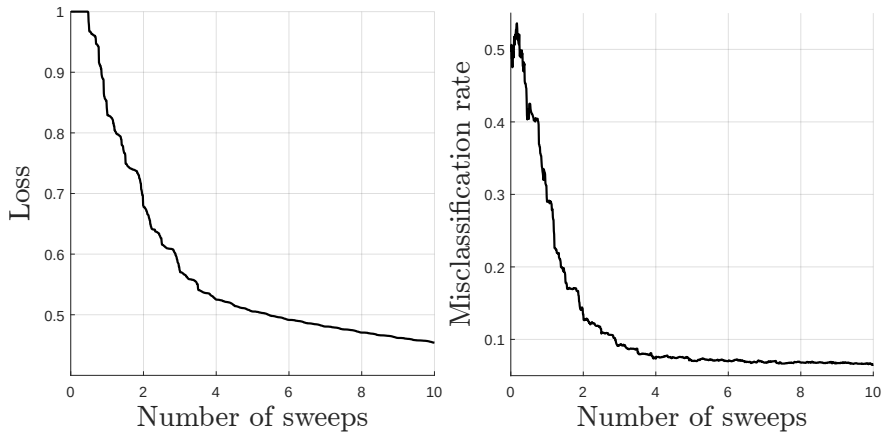


Figure 2: Monotonically decreasing normalized residual (loss) and misclassification rate as a function of the training iteration on the Spambase dataset.

Table 1: Predictive MSE (regression) and misclassification rate (classification) with one standard deviation for RFF, T-KRR and KRR on different UCI datasets.

Dataset	N	D	$\hat{M}$	R	RFF	T-KRR	KRR
Yacht	308	6	10	25	0.0021 ± 0.0018	0.0009 ± 0.0006	$\mathbf{0.0007} \pm 0.0004$
Energy [43]	768	9	20	10	0.0275 ± 0.0075	0.0200 ± 0.0066	0.0200 ± 0.0087
Airfoil	1503	5	20	10	0.2180 ± 0.0336	0.1679 ± 0.0258	$\mathbf{0.1587} \pm 0.0232$
Spambase	4601	57	40	10	0.3620 ± 0.0188	0.0935 ± 0.0095	$\mathbf{0.0909} \pm 0.0115$
Adult	45222	96	40	10	0.3976 ± 0.0056	$\mathbf{0.1596} \pm 0.0046$	N/A

gains. Furthermore, we note that due to the curse of dimensionality it is not possible to compare our approach with RFF while considering the same total number of basis functions $M = \hat{M}^D$. Hence we select $M_{\text{RFF}} = \hat{M}R$ such that the computational complexity of both methods is similar. Table 1 shows the MSE for different UCI datasets. We see that our approach comes close to the performance of full KRR and outperforms RFF on all datasets when considering the same number of model parameters. This is because our approach considers implicitly $\hat{M}^D$ frequencies to learn a low-rank parsimonious model which is fully described by $\hat{M}RD$ parameters, as opposed to RFF where the number of frequencies is the same as the number of model weights. Similarly to what is reported in [45, Figure 2] we observe lower performance of RFF on the Adult dataset than reported in [30]. Figure 2 plots the monotonically decreasing loss function as well as the corresponding misclassification rate while training with T-KRR on the Spambase dataset. After four sweeps the misclassification rate has converged. Since T-KRR is able to obtain similar performance as the KRR baseline on a range of small datasets, we proceed to tackle a large-scale regression problem.

4.3 Large-scale experiment

In order to highlight the ability of our method to deal with large data we consider the Airline dataset. The Airline dataset [13, 15] (Apache License 2.0) is a large-scale regression problem originally considered in [13] which is often used to compare state-of-the-art Gaussian Process Regression approximations. This is because of its large size and its non-stationary features. The goal is in fact to predict the delay of a flight given eight features, which are the age of the airplane, route distance, airtime, departure time, arrival time, day of the week, day of the month, and month. We follow the same exact preprocessing steps as in the experiments in [15], [38] and [9], which consider subsets of data of size $N = 10000, 100000, 1000000, 5929413$, each chosen uniformly at random. Training the model is then accomplished with $2/3N$ datapoints, with the remaining portion reserved for testing. The entire procedure is then repeated 10 times with random initialization in order to obtain a sample mean and sample standard deviation estimate of the MSE. Following exactly the approach in Hilbert-GP [38], we consider $\hat{M} = 40$ basis functions per dimension, with the crucial

Table 2: Predictive MSE with one standard deviation for T-KRR. The prefix -A indicates that the model in question considers an additive kernel.

N	10000	100000	1000000	5929413
T-KRR($R = 5$)	0.91 ± 0.10	0.82 ± 0.03	0.80 ± 0.02	0.80 ± 0.008
T-KRR($R = 10$)	0.89 ± 0.05	0.80 ± 0.05	0.79 ± 0.02	0.785 ± 0.009
T-KRR($R = 15$)	0.90 ± 0.07	0.80 ± 0.04	0.78 ± 0.02	0.773 ± 0.007
T-KRR($R = 20$)	0.97 ± 0.15	$\mathbf{0.78} \pm 0.04$	$\mathbf{0.77} \pm 0.01$	$\mathbf{0.763} \pm 0.007$
A-Hilbert-GP[Table 1] [38]	0.97 ± 0.14	0.80 ± 0.06	0.83 ± 0.02	0.827 ± 0.005
A-VFF[Table 1] [38]	0.89 ± 0.15	0.82 ± 0.05	0.83 ± 0.01	0.827 ± 0.004
SVIGP[Table 1] [38]	0.89 ± 0.16	0.79 ± 0.05	0.79 ± 0.01	0.791 ± 0.005
VISH[Table 2] [9]	0.90 ± 0.16	0.81 ± 0.05	0.83 ± 0.03	0.834 ± 0.055
GPR [Table 1] [38]	0.89 ± 0.16	N/A	N/A	N/A
A-GPR [Table 1] [38]	0.89 ± 0.16	N/A	N/A	N/A

difference that T-KRR approximates a standard *product* Gaussian kernel, as opposed to an *additive* model. As was the case in the previous section for classification, we select the lengthscale of the kernel as the mean of the standard deviations of the eight features and choose $\lambda_N = {}^{100}/_{N}$ for the different splits, where the dependency on N is to account for the fact that we minimize the squared loss instead of the mean squared loss. We then train four distinct models for $R = 5, 10, 15, 20$. Table 2 compares T-KRR with the best performing models found in the literature. The T-KRR method is able to recover the baseline GPR performance already with $R = 5$ and remarkably outperform all other approaches although we choose the hyperparameters naively and consider equal lengthscales l for all dimensions. Interestingly, a higher choice of R does result in better performance in all cases except for $N = 10000$, where model performance very much depends on the random split (note that we consider different splits for each experiment). The SVIGP [13] achieves comparable results to T-KRR, indicating that a performance gain is possible from product kernels. The difficulty with SVIGP is however that the kernel function is interpolated *locally* at M nodes, requiring an increasing amount of interpolation nodes to cover the whole domain to allow for good generalization. In contrast, T-KRR considers an exponentially large amount of basis functions in the frequency domain, and learns an efficient representation of the model weights. In light of the performance of the additive Hilbert-GP and additive VFF models, we expect similar performance when considering VFF feature maps. When considering the whole dataset, training our model with $R = 5$ takes 1565 ± 1 seconds on a laptop, while for $R = 20$ it takes 7141 ± 245 seconds. Reported training times of SVIGP indicate 18360 ± 360 seconds [38] on a cluster. Since the computational complexity of our algorithm is dominated by matrix-matrix multiplications, we expect significant speedups when relying on GPU computations.

5 Conclusion

In this work a framework to perform approximate Kernel Ridge Regression with tensors by leveraging the tensor product structure of deterministic Fourier features was introduced. Concretely, a monotonically decreasing learning algorithm with linear complexity in both sample size and dimensionality was derived. This algorithm leverages the efficient format of the Canonical Polyadic Decomposition in combination with exponentially converging deterministic Fourier features. Numerical experiments show how the performance of the baseline Kernel Ridge Regression is recovered with a very limited number of parameters. Very much like Kernel Ridge Regression itself, the biggest limitation of the current approach is that it is non-probabilistic, which motivates further work in that direction. We also plan to test our approach further on other large-scale problems.

References

[1] Z. Chen, K. Batselier, J. A. K. Suykens, and N. Wong. Parallelized Tensor Train Learning of Polynomial Classifiers. *IEEE Transactions on Neural Networks and Learning Systems*, 29(10): 4621–4632, Oct. 2018. ISSN 2162-2388. doi: 10.1109/TNNLS.2017.2771264.

[2] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic. Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions. *Foundations and Trends® in Machine Learning*, 9(4-5):249–429, 2016. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000059.

[3] P. Comon, X. Luciani, and A. L. F. de Almeida. Tensor decompositions, alternating least squares and other tales. *Journal of Chemometrics*, 23(7-8):393–405, July 2009. ISSN 08869383, 1099128X. doi: 10.1002/cem.1236.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sept. 1995. ISSN 1573-0565. doi: 10.1007/BF00994018.

[5] T. Dao, C. D. Sa, and C. Ré. Gaussian quadrature for kernel features. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6109–6119, Red Hook, NY, USA, Dec. 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.

[6] L. De Lathauwer, B. De Moor, and J. Vandewalle. A Multilinear Singular Value Decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, Jan. 2000. ISSN 0895-4798. doi: 10.1137/S0895479896305696.

[7] P. Drineas and M. W. Mahoney. On the Nystrom Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. *Journal of Machine Learning Research*, 6(72):2153–2175, 2005. ISSN 1533-7928.

[8] D. Dua and C. Graff. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml, 2017.

[9] V. Dutordoir, N. Durrande, and J. Hensman. Sparse Gaussian Processes with Spherical Harmonic Features. In *International Conference on Machine Learning*, pages 2793–2802. PMLR, Nov. 2020.

[10] A. Garriga-Alonso, C. E. Rasmussen, and L. Aitchison. Deep Convolutional Networks as shallow Gaussian Processes. In *International Conference on Learning Representations*, 2018.

[11] E. Gilboa, Y. Saatçi, and J. P. Cunningham. Scaling Multidimensional Inference for Structured Gaussian Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2): 424–436, Feb. 2015. ISSN 1939-3539. doi: 10.1109/TPAMI.2013.192.

[12] B. Hammer and K. Gersmann. A Note on the Universal Approximation Capability of Support Vector Machines. *Neural Processing Letters*, 17(1):43–53, Feb. 2003. ISSN 1573-773X. doi: 10.1023/A:1022936519097.

[13] J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for Big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, UAI'13, pages 282–290, Arlington, Virginia, USA, Aug. 2013. AUAI Press.

[14] J. Hensman, A. Matthews, and Z. Ghahramani. Scalable Variational Gaussian Process Classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR, Feb. 2015.

[15] J. Hensman, N. Durrande, and A. Solin. Variational Fourier features for Gaussian processes. *The Journal of Machine Learning Research*, 18(1):5537–5588, Jan. 2017. ISSN 1532-4435.

[16] F. L. Hitchcock. The Expression of a Tensor or a Polyadic as a Sum of Products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927. ISSN 1467-9590. doi: 10.1002/sapm192761164.

[17] S. Holtz, T. Rohwedder, and R. Schneider. The alternating linear scheme for tensor optimization in the tensor train format. *SIAM J. Sci. Comput.*, 34(2):A683–A713, 2012.

[18] P. Izmailov, A. Novikov, and D. Kropotov. Scalable Gaussian Processes with Billions of Inducing Inputs via Tensor Train Decomposition. In *International Conference on Artificial Intelligence and Statistics*, pages 726–735. PMLR, Mar. 2018.

[19] T. G. Kolda and B. W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 51(3): 455–500, Aug. 2009. ISSN 0036-1445, 1095-7200. doi: 10.1137/07070111X.

[20] J. B. Kruskal. Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra and its Applications*, 18(2): 95–138, Jan. 1977. ISSN 0024-3795. doi: 10.1016/0024-3795(77)90069-6.

[21] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep Neural Networks as Gaussian Processes. *arXiv:1711.00165 [cs, stat]*, Mar. 2018.

[22] F. Liu, X. Huang, Y. Chen, and J. A. K. Suykens. Random Features for Kernel Approximation: A Survey on Algorithms, Theory, and Beyond. *arXiv:2004.11154 [cs, stat]*, July 2020.

[23] Matlab. *9.10.0.1649659 (R2021a) Update 1*. The MathWorks Inc., Natick, Massachussetts, 2021.

[24] M. Mutný and A. Krause. Efficient high dimensional Bayesian optimization with additivity and quadrature fourier features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 9019–9030, Red Hook, NY, USA, Dec. 2018. Curran Associates Inc.

[25] R. Novak, L. Xiao, Y. Bahri, J. Lee, G. Yang, J. Hron, D. A. Abolafia, J. Pennington, and J. Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2018.

[26] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[27] A. Novikov, M. Trofimov, and I. Oseledets. Exponential Machines. *arXiv:1605.03795 [cs, stat]*, Dec. 2017.

[28] I. V. Oseledets. Tensor-Train Decomposition. *SIAM Journal on Scientific Computing*, 33(5): 2295–2317, Jan. 2011. ISSN 1064-8275, 1095-7197. doi: 10.1137/090752286.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. ISSN 1533-7928.

[30] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pages 1177–1184, Red Hook, NY, USA, Dec. 2007. Curran Associates Inc. ISBN 978-1-60560-352-0.

[31] C. E. Rasmussen and H. Nickisch. Gaussian Processes for Machine Learning (GPML) Toolbox. *Journal of Machine Learning Research*, 11(100):3011–3015, 2010. ISSN 1533-7928.

[32] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass, 2006. ISBN 978-0-262-18253-9.

[33] T. Rohwedder and A. Uschmajew. On local convergence of alternating schemes for optimization of convex problems in the tensor train format. *SIAM Journal on Numerical Analysis*, 51(2): 1134–1162, 2013.

[34] W. Rudin. *Fourier Analysis on Groups.* Number 12 in Interscience Tracts in Pure and Applied Mathematics. Interscience Publishers, New York, 1962. ISBN 0-471-52364-X 978-0-471-52364-2.

[35] P. F. Shustin and H. Avron. Gauss-Legendre Features for Gaussian Process Regression. *arXiv:2101.01137 [cs, math]*, Jan. 2021.

[36] N. D. Sidiropoulos and R. Bro. On the uniqueness of multilinear decomposition of N-way arrays. *Journal of Chemometrics*, 14(3):229–239, 2000. ISSN 1099-128X. doi: 10.1002/1099-128X(200005/06)14:3<229::AID-CEM587>3.0.CO;2-N.

[37] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. Tensor Decomposition for Signal Processing and Machine Learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, July 2017. ISSN 1941-0476. doi: 10.1109/TSP.2017.2690524.

[38] A. Solin and S. Särkkä. Hilbert space methods for reduced-rank Gaussian process regression. *Statistics and Computing*, 30(2):419–446, Mar. 2020. ISSN 1573-1375. doi: 10.1007/s11222-019-09886-w.

[39] E. M. Stoudenmire and D. J. Schwab. Supervised learning with tensor networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 4806–4814, Red Hook, NY, USA, Dec. 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9.

[40] J. Suykens and J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300, June 1999. ISSN 1573-773X. doi: 10.1023/A:1018628609742.

[41] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, Nov. 2002. ISBN 978-981-238-151-4 978-981-277-665-5. doi: 10.1142/5089.

[42] A. Tjandra, S. Sakti, and S. Nakamura. Compressing recurrent neural network with tensor train. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 4451–4458, May 2017. doi: 10.1109/IJCNN.2017.7966420.

[43] A. Tsanas and A. Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567, 2012.

[44] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3): 279–311, Sept. 1966. ISSN 1860-0980. doi: 10.1007/BF02289464.

[45] T. Yang, Y.-f. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[46] Y. Yang, D. Krompass, and V. Tresp. Tensor-Train Recurrent Neural Networks for Video Classification. In *International Conference on Machine Learning*, pages 3891–3900. PMLR, July 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Yes, we describe how to efficiently learn from deterministic Fourier features by leveraging their tensor product structure.
   (b) Did you describe the limitations of your work? [Yes] Yes, we discussed the limitations in Subsection 3.4 and in the Conclusion 5
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our article describes a general-purpose methodology and it is therefore difficult to discuss any potential negative societial impact due to its indirectness.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Yes, all results are easily reproducible by means of the code, which as mentioned, can be accessed at `https://github.com/NeurIPS2021Submission/Code`.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Yes, all details are discussed in Section 4. Also, the uploaded code is reproducible by means fixed random seeds.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Yes, see Table 1 and Table 2.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Yes, see Section 4.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] Yes, see [23], [8], [13, 15] in the text. We cited the creators of UCI datasets whenever they explicitly required it.

(b) Did you mention the license of the assets? [Yes] Yes, whenever first introduced and necessary (so in case of Matlab, GPML Toolbox, and the Airline dataset. In case of UCI, neither UCI, nor the datasets we used are licensed. We still cited them if they required it).

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

Derivation of the data-fitting term of (12). We make use of the multi-linearity property of the CPD and rely on re-ordering the summations:

$$
\begin{aligned}
\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{Z}}(\boldsymbol{x}) \rangle_{\mathrm{F}} &= \left\langle \sum_{r=1}^{R} \boldsymbol{w}_r^{(1)} \otimes \boldsymbol{w}_r^{(2)} \otimes \cdots \otimes \boldsymbol{w}_r^{(D)}, \boldsymbol{z}^{(1)} \otimes \boldsymbol{z}^{(2)} \cdots \otimes \boldsymbol{z}^{(D)} \right\rangle_{\mathrm{F}} \\
&= \sum_{i_1=1}^{\hat{M}} \cdots \sum_{i_d=1}^{\hat{M}} \cdots \sum_{i_D=1}^{\hat{M}} \sum_{r=1}^{R} w_{i_1 r}^{(1)} z_{i_1}^{(1)} \cdots w_{i_d r}^{(d)} z_{i_d}^{(d)} \cdots w_{i_D r}^{(D)} z_{i_D}^{(D)} \\
&= \sum_{i_d=1}^{\hat{M}} \sum_{r=1}^{R} w_{i_d r}^{(d)} \left(z_{i_d}^{(d)} \sum_{i_1=1}^{\hat{M}} w_{i_1 r}^{(1)} z_{i_1}^{(1)} \cdots \sum_{i_D=1}^{\hat{M}} w_{i_D r}^{(D)} z_{i_D}^{(D)} \right) \\
&= \mathrm{vec}\left(\boldsymbol{W}^{(d)} \right)^{\mathrm{T}} \mathrm{vec}\left(\boldsymbol{z}^{(d)} \otimes \left(\boldsymbol{W}^{(1)} \boldsymbol{z}^{(1)} \odot \cdots \odot \boldsymbol{W}^{(D)} \boldsymbol{z}^{(D)} \right) \right) \\
&= \left\langle \mathrm{vec}\left(\boldsymbol{W}^{(d)} \right), \mathrm{vec}\left(\boldsymbol{G}^{d} \right) \right\rangle
\end{aligned}
$$

The derivation of the regularization term of (13) follows a similar reasoning as for the data-fitting term:

$$
\begin{aligned}
\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{W}} \rangle_{\mathrm{F}} &= \left\langle \sum_{r=1}^{R} \boldsymbol{w}_r^{(1)} \otimes \boldsymbol{w}_r^{(2)} \otimes \cdots \otimes \boldsymbol{w}_r^{(D)}, \sum_{r=1}^{R} \boldsymbol{w}_r^{(1)} \otimes \boldsymbol{w}_r^{(2)} \otimes \cdots \otimes \boldsymbol{w}_r^{(D)} \right\rangle_{\mathrm{F}} \\
&= \sum_{i_1=1}^{\hat{M}} \cdots \sum_{i_d=1}^{\hat{M}} \cdots \sum_{i_D=1}^{\hat{M}} \sum_{r=1}^{R} \sum_{p=1}^{R} w_{i_1 r}^{(1)} \, w_{i_1,p}^{(1)} \cdots w_{i_d r}^{(d)} \, w_{i_d,p}^{(d)} \cdots w_{i_D r}^{(D)} \, w_{p i_D}^{(D)} \\
&= \sum_{r=1}^{R} \sum_{p=1}^{R} \left(\sum_{i_d=1}^{\hat{M}} w_{i_d r}^{(d)} \, w_{i_d p}^{(d)} \right) \left(\sum_{i_1=1}^{\hat{M}} w_{i_1 r}^{(1)} \, w_{i_1 p}^{(1)} \cdots \sum_{i_D=1}^{\hat{M}} w_{i_D r}^{(D)} \, w_{i_D p}^{(D)} \right) \\
&= \sum_{r=1}^{R} \sum_{p=1}^{R} \left(\boldsymbol{w}_r^{(d)^{\mathrm{T}}} \boldsymbol{w}_p^{(d)} \right) \left(\boldsymbol{w}_r^{(1)^{\mathrm{T}}} \boldsymbol{w}_p^{(1)} \odot \cdots \odot \boldsymbol{w}_r^{(D)^{\mathrm{T}}} \boldsymbol{w}_p^{(D)} \right) \\
&= \mathrm{vec} \left(\boldsymbol{W}^{(d)^{\mathrm{T}}} \boldsymbol{W}^{(d)} \right)^{\mathrm{T}} \mathrm{vec} \left(\boldsymbol{W}^{(1)^{\mathrm{T}}} \boldsymbol{W}^{(1)} \odot \cdots \odot \boldsymbol{W}^{(D)^{\mathrm{T}}} \boldsymbol{W}^{(D)} \right) \\
&= \left\langle \mathrm{vec} \left(\boldsymbol{W}^{(d)^{\mathrm{T}}} \boldsymbol{W}^{(d)} \right), \mathrm{vec} \left(\boldsymbol{H}^{(d)} \right) \right\rangle
\end{aligned}
$$