Tensor-based Kernel Machines with Structured Inducing Points for Large and High-Dimensional Data

Frederiek Wesel
Delft Center for Systems and Control
Delft University of Technology

Kim Batselier
Delft Center for Systems and Control
Delft University of Technology

Abstract

Kernel machines are one of the most studied family of methods in machine learning. In the exact setting, training requires to instantiate the kernel matrix, thereby prohibiting their application to large-sampled data. One popular kernel approximation strategy which allows to tackle large-sampled data consists in interpolating product kernels on a set of grid-structured inducing points. However, since the number of model parameters increases exponentially with the dimensionality of the data, these methods are limited to small-dimensional datasets. In this work we lift this limitation entirely by placing inducing points on a grid and constraining the primal weights to be a low-rank Canonical Polyadic Decomposition. We derive a block coordinate descent algorithm that efficiently exploits grid-structured inducing points. The computational complexity of the algorithm scales linearly both in the number of samples and in the dimensionality of the data for any product kernel. We demonstrate the performance of our algorithm on large-scale and high-dimensional data, achieving state-of-the art results on a laptop computer. Our results show that grid-structured approaches can work in higher-dimensional problems.

1 INTRODUCTION

Kernel machines, such as Support Vectors Machines (SVMs) (Cortes and Vapnik, 1995) and Gaussian Processes (GPs) (Rasmussen and Williams, 2006) are a family of machine learning methods that handle inference of nonlinear functions by lifting the data into a high and possibly infinite-dimensional feature space and performing linear inference therein. Because of their elegant formulation, their connection with reproducing kernel Hilbert spaces and the guarantees which stem from their convex optimization setting, they have become one of the most widely studied machine learning paradigms. Furthermore, kernel machines are known for their connections with neural networks (Lee et al., 2018; Novak et al., 2018; Garriga-Alonso et al., 2018) and for the fact that they can be universal function approximators for a suitable choice of kernel (Hammer and Gersmann, 2003).

The main limitation of kernel machines is that training involves instantiating the kernel matrix which encodes the similarities between all mapped data. This results in a cost of at least $\mathcal{O}(N^2)$, limiting their applicability to small datasets. To obviate this problem, a number of low-rank approaches based on random features (Rahimi and Recht, 2007; Yang et al., 2015) and inducing points (Williams and Seeger, 2001; Smola and Bartlett, 2001; Csató and Opper, 2002; Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2006; Titsias, 2009; Meanti et al., 2020) have been developed. Broadly speaking, these methods seek a rank-$M \ll N$ approximation of the kernel function, which enables faster inference at cost of $\mathcal{O}(NM^2)$. However, this scaling forces the modeler to trade-off accuracy of the kernel approximation with the ability to process large-scale data.

A popular family of approaches that is based on the Nyström approximation of kernel functions is Structured Kernel Interpolation (SKI) (Wilson and Nickisch, 2015; Nickson et al., 2015; Gardner et al., 2018; Stanton et al., 2021; Yadav et al., 2021). SKI-methods do not sacrifice accuracy for fast inference since they interpolate the kernel function globally on a regularly spaced grid in order to exploit the ensuing structure for computational gains. However, since the number of interpolation points are placed on a regular grid, and therefore the number of model parameters increases exponentially with the dimensionality of the data, these approaches are limited to small-dimensional datasets.

Recently, the Canonical Polyadic Decomposition (CPD) (Kolda and Bader, 2009), a tool from multi-linear algebra, has been applied in the context of kernel machines to by-

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206. Copyright 2023 by the author(s).

pass the exponential growth of model parameters affecting Fourier features-based approximations of stationary kernels (Wesel and Batselier, 2021) by constraining the model weights to be a CPD of low rank. This low-rank constraint allows to learn a model with a *linear* number of model parameters in the dimensionality D, but requires knowledge of the spectral representation (Fourier transform) of the kernel of choice which in general can be unknown or non-analytical, requiring then further approximations. In this paper we develop a CPD-based approach to learn from *any* product kernel which allows grid-structured inducing points methods to scale to both large-sampled and high-dimensional data. We exploit the tensor-product structure of grid-structured inducing point by constraining the model weights to be a CPD of low-rank. Under this constraint, we derive a block coordinate descent algorithm that allows for the efficient training of kernel machines. Our algorithm has a computational complexity of $\mathcal{O}(NDM^{\frac{2}{D}}R^2)$ and storage complexity of $\mathcal{O}(NR)$, where R is the rank of the tensor-decomposition which controls the time versus accuracy trade-off. We show through experiments that competitive results in terms of performance can be obtained on a laptop computer for data that is both large in sample size as well as in dimensionality.

2 BACKGROUND

Notation Calligraphic italics indicate a function space or vector space, e.g. $\mathcal{X}$. We indicate as $\boldsymbol{K_{XM}}$ the kernel matrix which considers similarities between rows of $\boldsymbol{X}$ and $\boldsymbol{M}$ such that $[\boldsymbol{K_{XM}}]_{i,j} = k(\boldsymbol{X}_{i,:}, \boldsymbol{M}_{j,:})$. The Kronecker tensor-product, Cartesian product, Hadamard product and Hadamard division are denoted respectively as $\otimes$, $\times$, $\odot$ and $\oslash$. We denote a D-dimensional tensor as $\boldsymbol{w} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$. The vectorization of a tensor $\boldsymbol{w} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ is denoted by $\operatorname{vec}(\boldsymbol{w})_i = w_{i_1 i_2 \cdots i_D}$, where $i = i_1 + \sum_{d=2}^{D}(i_d - 1)\prod_{k=1}^{d-1} I_k$. We denote the Frobenius inner product by $\langle \boldsymbol{a}, \boldsymbol{b} \rangle := \langle \operatorname{vec}(\boldsymbol{a}), \operatorname{vec}(\boldsymbol{b}) \rangle$, where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$.

Supervised Learning In the context of supervised learning, the goal is to estimate a function $f(\cdot) : \mathcal{X} \to \mathcal{Y}$ given only a finite set of i.i.d. input-output pairs $(\boldsymbol{x}_n, y_n)_{n=1}^{N}$ s.t. $\boldsymbol{x}_n \in \mathcal{X}$, $y \in \mathcal{Y}$, $\forall n \in \{1, \ldots, N\}$ generated by some probability measure $P(\boldsymbol{x}, y)$. After defining a measure of loss $\ell(f(\boldsymbol{x}), y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$, this can be accomplished by minimizing the (regularized) empirical:

$$R_{\text{empirical}}(f) = \frac{1}{N}\sum_{n=1}^{N} \ell(f(\boldsymbol{x}_n), y_n) + \lambda \|f\|_{\mathcal{H}}^2, \quad (1)$$

where $\lambda \in \mathbb{R}_+$ is a regularization hyperparameter which enforces a penalty on model complexity.

2.1 Kernel Machines

Kernel machines model f as linear in the mapped data, i.e.

$$f(\boldsymbol{x}) = \langle \boldsymbol{\varphi}(\boldsymbol{x}), \boldsymbol{w} \rangle, \quad (2)$$

and ℓ to be convex. Here $\boldsymbol{\varphi}(\cdot) : \mathcal{X} \to \mathcal{H}$ is the *feature map* which lifts the data in a high (and possibly infinite-dimensional) reproducing kernel Hilbert space $\mathcal{H}$ where linear inference is possible, and $\boldsymbol{w}$ are the model weights. In practice, this explicit mapping can be avoided by considering a kernel function $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that $k(\boldsymbol{x}, \boldsymbol{x'}) = \langle \boldsymbol{\varphi}(\boldsymbol{x}), \boldsymbol{\varphi}(\boldsymbol{x'}) \rangle$. By the representer theorem (Schölkopf et al., 2001) we have in fact that:

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_n, \boldsymbol{x}), \quad (3)$$

which implies that we only need to estimate multipliers $\boldsymbol{\alpha} \in \mathbb{R}^N$. Depending on the choice of loss function different kernel machines arise, for instance hinge loss leads to support vector machines, squared loss to Kernel Ridge Regression (KRR), which is the same estimator as the Gaussian process regression posterior mean. In case of squared loss, Equation (1) can be minimized exactly from the corresponding dual optimization problem:

$$(\boldsymbol{K_{XX}} + \lambda N \mathrm{I})\,\boldsymbol{\alpha} = \boldsymbol{y}. \quad (4)$$

In practice, since the kernel evaluations between all points need to be computed, the computational cost of training in the dual is at least $\mathcal{O}(N^2)$, limiting its usefulness to small-sampled data.

Structured Data One way to enable exact inference on large-scale data is to exploit existing structure in the data. A particular fortunate case arises when the data lies on a Cartesian grid $\boldsymbol{x}^{(1)} \times \boldsymbol{x}^{(2)} \times \cdots \times \boldsymbol{x}^{(D)}$ where each $\boldsymbol{x}^{(d)} \in \mathbb{R}^{N_d}$ such that $N = \prod_{d=1}^{D} N_d$, and when considering product kernels of the form:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \prod_{d=1}^{D} k^{(d)}\left(x_i^{(d)}, x_j^{(d)}\right), \quad (5)$$

where $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{R}^D$. In this case the kernel matrix $\boldsymbol{K_{XX}}$ is the Kronecker product of small matrices $\boldsymbol{K}_{\boldsymbol{x}^{(d)}\boldsymbol{x}^{(d)}} \in \mathbb{R}^{N_d \times N_d}$ (Saatchi, 2011, Equation 5.7):

$$\boldsymbol{K_{XX}} = \boldsymbol{K}_{\boldsymbol{x}^{(1)}\boldsymbol{x}^{(1)}} \otimes \boldsymbol{K}_{\boldsymbol{x}^{(2)}\boldsymbol{x}^{(2)}} \otimes \cdots \otimes \boldsymbol{K}_{\boldsymbol{x}^{(D)}\boldsymbol{x}^{(D)}}. \quad (6)$$

Storing the full kernel matrix $\boldsymbol{K_{XX}} \in \mathbb{R}^{N \times N}$ can then be avoided by storing smaller kernel matrices $\boldsymbol{K}_{\boldsymbol{x}^{(1)}\boldsymbol{x}^{(1)}}, \ldots, \boldsymbol{K}_{\boldsymbol{x}^{(D)}\boldsymbol{x}^{(D)}}$ without ever computing the tensor-products. Exact training can then be accomplished with $\mathcal{O}(DN^{1+\frac{1}{D}})$ operations by exploiting the properties of the Kronecker product (Saatchi, 2011; Gilboa et al., 2015).

Unstructured Data One way to enable faster inference to unstructured data is to consider the Nyström method (Williams and Seeger, 2001; Suykens, 2002), whose key idea is to approximate the spectrum of the full kernel matrix $\boldsymbol{K_{XX}}$ by means of a restricted number of kernel evaluations at a subset of $M \ll N$ inducing points denoted by $\boldsymbol{M}$, traditionally sampled at random (Williams and Seeger, 2001) from the data $\boldsymbol{X}$, defining hence the Nyström approximation (Williams and Seeger, 2001, Equations 8-9):

$$\boldsymbol{K_{XX}} \approx \boldsymbol{K_{XM}} \boldsymbol{L}^{-\mathrm{T}} \boldsymbol{L}^{-1} \boldsymbol{K_{MX}} =: \boldsymbol{K}_{\text{Nyström}}, \tag{7}$$

where $\boldsymbol{L}$ is such that $\boldsymbol{K_{MM}} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$. Embedding Equation (7) in Equation (1) under the assumption of squared loss gives rise to a linear least-squares problem which can be solved from the normal equations:

$$\left(\boldsymbol{L}^{-1}\boldsymbol{K_{MX}}\boldsymbol{K_{XM}}\boldsymbol{L}^{-\mathrm{T}} + \lambda N \mathrm{I}\right)\boldsymbol{w} = \boldsymbol{L}^{-1}\boldsymbol{K_{MX}}\boldsymbol{y}, \tag{8}$$

This formulation enables training at the computational cost of $\mathcal{O}(NM^2 + M^3)$ and storage cost of $\mathcal{O}(M^2)$. As argued previously however, these complexities force to chose between the accuracy of the approximation and the ability to process large-scale data, as $M \ll N$ for any computational gains. When one considers stationary product kernels, one naive approach would be to locate a large number of inducing points $M \gg N$ on a Cartesian grid and to exploit the ensuing Toeplitz (one-dimensional inputs) and Kronecker (higher-dimensional inputs) structures for computational gains, as in Equation (6). Since this alone only alleviates the complexity associated with $\boldsymbol{K_{MM}}$, plenty of research has focused on approximating $\boldsymbol{K_{XM}}$ which accounts for the dominant $\mathcal{O}(NM^2)$ term. One of these methods is Structured Kernel Interpolation (SKI) (Wilson and Nickisch, 2015). In SKI, the cross-covariance matrix $\boldsymbol{K_{XM}}$ is approximated by local interpolation, i.e. $\boldsymbol{K_{XM}} \approx \boldsymbol{P}\boldsymbol{K_{MM}}$, where $\boldsymbol{P}$ is a sparse interpolation matrix with 2^D non-zero elements per row (in case of linear interpolation), giving rise to the SKI kernel

$$\boldsymbol{K_{XX}} \approx \boldsymbol{P}\boldsymbol{K_{MM}}\boldsymbol{P}^{\mathrm{T}} =: \boldsymbol{K}_{\text{SKI}}. \tag{9}$$

When considering a stationary product kernel $\boldsymbol{K_{MM}}$ has a Toeplitz structure (one-dimension) or a Kronecker product structure of Toeplitz matrices (higher-dimensions). SKI takes advantage of these structures by approximately solving $(\boldsymbol{K}_{\text{SKI}} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$ using Krylov subspace methods which rely on matrix-vector products. Since $\boldsymbol{P}$ is sparse and $\boldsymbol{K_{MM}}$ is structured, each iteration of SKI costs only $\mathcal{O}(N + M\log M)$ operations and $\mathcal{O}(NM)$ memory. However, since M scales exponentially in D, SKI is limited to sets of data of small dimensionality $D < 5$ (Wilson and Nickisch, 2015). In order to mitigate this exponential dependency in D, Gardner et al. (2018) approximate the kernel matrix of a stationary product kernel as the Hadamard product of rank-R SKI kernel matrices in order to perform fast matrix-vector products in a divide-and-conquer fashion. Although this approach overcomes the curse of dimensionality, it requires the storage of R copies (typically 30) of the dataset limiting its applicability to data of moderate dimensionality.

Recent extensions and improvement of the SKI framework are the handling of online data (Stanton et al., 2021), its reformulation as a Bayesian linear regression problem (Yadav et al., 2021) and the use of a permutohedral lattice instead of a Cartesian grid (Kapoor et al., 2021). This latter approach reduces the number of neighboring points from 2^D to $D+1$, alleviating the curse of dimensionality by allowing training at $\mathcal{O}(D^2(N+M))$. However, this latter approach is most effective only for $D \leq 20$ (Kapoor et al., 2021) due to the quadratic scaling in D of the computational complexity and the decreasing accuracy of the kernel approximation as D increases.

2.2 Tensor Decompositions

The most common tensor decompositions are the Canonical Polyadic Decomposition (CPD) (Hitchcock, 1927; Kolda and Bader, 2009), the Tucker decomposition (Tucker, 1966) and the tensor train decomposition (Oseledets, 2011). A rank-R CPD decomposes a tensor $\boldsymbol{w} \in \mathbb{R}^{M_1 \times M_2 \times \cdots \times M_D}$ as a sum of R outer products of vectors $\boldsymbol{w}^{(d)} \in \mathbb{R}^{M_d}$ such that:

$$\operatorname{vec}(\boldsymbol{w}) = \sum_{r=1}^{R} \boldsymbol{w}_r^{(1)} \otimes \boldsymbol{w}_r^{(2)} \otimes \cdots \otimes \boldsymbol{w}_r^{(D)}, \tag{10}$$

where the rank R is defined as the smallest R such that Equation (10) holds exactly (Hitchcock, 1927; Kolda and Bader, 2009). Here $\boldsymbol{W}^{(d)} \in \mathbb{R}^{M_d \times R}$ are the *factor matrices* such that $[\boldsymbol{W}^{(d)}]_{:,r} = \boldsymbol{w}_r^{(d)}$. Storing $\boldsymbol{w}$ in decomposed form requires then only to store D factor matrices, requiring $R\sum_{d=1}^{D} M_d$ memory units as opposed to $M = \prod_{d=1}^{D} M_d$. Because of this compression, tensor decompositions have been used to reduce the number of model parameters in deep learning models by tensorizing and decomposing weights (Novikov et al., 2015; Tjandra et al., 2017; Yang et al., 2017; Khodak et al., 2020), or compressing filters which have tensorial structure by definition, e.g. convolutions (Favier and Bouilloc, 2009; Lebedev et al., 2015; Batselier et al., 2017).

Tensor decompositions have also been used to reduce the exponential number of parameters which arise when learning from tensor-product feature maps by constraining the weight tensor to be a low-rank tensor decomposition. So far these models have considered trigonometric (Stoudenmire and Schwab, 2016), polynomial (Novikov et al., 2018; Chen et al., 2018) and Fourier feature maps (Wesel and Batselier, 2021), where the latter are used to induce stationary product kernels. Furthermore, in the context of GPs, tensor decompositions have been used to reduce the complexity of multi-output GPs (Zhe et al., 2019) and in the context of stochastic variational GPs to compress the mean of the variational posterior distribution (Izmailov et al., 2018). However the former method is not designed to handle data which is larger in the number of samples, as it scales with $\mathcal{O}\left(DN^2\right)$, while the latter does not work for $D > 10$, as it becomes unpractical to store the interpolation grid, forcing to train on $D \leq 10$-dimensional embeddings of the data.

In the following section, building on the works of Wilson and Nickisch (2015) and Wesel and Batselier (2021), we derive in the context of classical kernel machines a block-coordinate descent algorithm whose computational complexity at training scales with $\mathcal{O}(NDM^{\frac{2}{D}}R^2)$ requiring $\mathcal{O}(NR)$ memory, allowing to tackle large-scale and large dimensional problems as demonstrated experimentally in Section 4.

3 GRID-STRUCTURED KERNEL MACHINES

In our approach we consider product kernels (Equation (5)) and Nyström inducing points (Equation (7)), which inspired by Wilson and Nickisch (2015) we place on a Cartesian grid $\boldsymbol{m}^{(1)} \times \boldsymbol{m}^{(2)} \times \cdots \times \boldsymbol{m}^{(D)}$, where each $\boldsymbol{m}^{(d)} \in \mathbb{R}^{M_d}$ and $M = \prod_{d=1}^{d} M_d$. This approximation recovers the underlying kernel as $M \to \infty$ (Evans and Nair, 2018, Theorem 1). We have already seen in Equation (6) how $\boldsymbol{K_{MM}}$ can be stored and manipulated efficiently by considering its tensor-product structure by indexing it as a tensor. This allows for the efficient computation of its Cholesky factor $\boldsymbol{L}$ (Saatchi, 2011, Theorem 5.2) from

$$\begin{aligned}&\boldsymbol{K}_{\boldsymbol{m}^{(1)}\boldsymbol{m}^{(1)}} \otimes \boldsymbol{K}_{\boldsymbol{m}^{(2)}\boldsymbol{m}^{(2)}} \otimes \cdots \otimes \boldsymbol{K}_{\boldsymbol{m}^{(D)}\boldsymbol{m}^{(D)}} \\ =&\boldsymbol{L}^{(1)}{\boldsymbol{L}^{(1)}}^{\mathrm{T}} \otimes \boldsymbol{L}^{(2)}{\boldsymbol{L}^{(2)}}^{T} \otimes \cdots \otimes \boldsymbol{L}^{(D)}{\boldsymbol{L}^{(D)}}^{T} \\ =&\underbrace{\left(\boldsymbol{L}^{(1)} \otimes \cdots \otimes \boldsymbol{L}^{(D)}\right)}_{\boldsymbol{L}}\underbrace{\left(\boldsymbol{L}^{(1)} \otimes \cdots \otimes \boldsymbol{L}^{(D)}\right)^{T}}_{\boldsymbol{L}^{\mathrm{T}}},\end{aligned} \tag{11}$$

as it inherits the tensor-product structure. Similarly, each row $\boldsymbol{k_{xM}}$ of $\boldsymbol{K_{XM}}$ has a tensor-product structure:

$$\boldsymbol{k_{xM}} = \boldsymbol{k}_{x_1\boldsymbol{m}^{(1)}} \otimes \boldsymbol{k}_{x_2\boldsymbol{m}^{(2)}} \otimes \cdots \otimes \boldsymbol{k}_{x_D\boldsymbol{m}^{(D)}},$$

derivations can be found in the supplementary material. By the mixed-product property of the tensor-product (Saatchi, 2011, Equation 5.10) we have that:

$$\boldsymbol{k_{xM}}\boldsymbol{L}^{-\mathrm{T}} = \boldsymbol{k}_{x_1\boldsymbol{m}^{(1)}}{\boldsymbol{L}^{(1)}}^{-\mathrm{T}} \otimes \cdots \otimes \boldsymbol{k}_{x_D\boldsymbol{m}^{(D)}}{\boldsymbol{L}^{(D)}}^{-\mathrm{T}}.$$

The computational benefits associated with this tensor-product structure can however not be exploited without further assumptions, as model evaluations $\left\langle \boldsymbol{k_{xM}}\boldsymbol{L}^{-\mathrm{T}}, \boldsymbol{w}\right\rangle$ will require in fact still $\mathcal{O}(M)$ computations and the 'unpacking' of the tensor-product structure. In the next paragraph, we will show how one can leverage fully the tensorial structure of $\boldsymbol{k_{xM}}\boldsymbol{L}^{-\mathrm{T}}$ by assuming that the model weights are a rank-R CPD tensor. This will allow to consider both large and high-dimensional datasets with grid-structured inducing points, by effectively reducing the number of model parameters from M to $RDM^{\frac{1}{D}}$.

We now wish to minimize the empirical risk Equation (1) under a convex loss, with the additional constraint that the weight tensor $\boldsymbol{w}$ has a rank-R CPD structure:

$$\min_{\boldsymbol{w}} \frac{1}{N}\sum_{n=1}^{N} \ell\left(\left\langle \boldsymbol{k}_{\boldsymbol{x}_n\boldsymbol{M}}\boldsymbol{L}^{-\mathrm{T}}, \boldsymbol{w}\right\rangle, y_n\right) + \lambda\left\langle \boldsymbol{w}, \boldsymbol{w}\right\rangle, \tag{12}$$

$$\text{subject to CP-rank}\left(\boldsymbol{w}\right) = R, \tag{13}$$

where if R is chosen to be the true CPD rank of $\boldsymbol{w}$, the solution of Equation (1) associated with Equation (7) is recovered. In this case, $\boldsymbol{w}$ is furthermore also unique under mild conditions (Sidiropoulos and Bro, 2000). As we will see, this constraint enables to fully exploit the rank-1 CPD structure of $\boldsymbol{k_{xM}}\boldsymbol{L}^{-\mathrm{T}}$ by allowing to optimize one CPD factor matrix $\boldsymbol{W}^{(d)}$ of $\boldsymbol{w}$ at a time, enabling to tackle large-sampled and large-dimensional datasets with modest hardware. This is accomplished by exploiting the *multilinearity* of tensor decomposition which allows to express the empirical risk as a *linear* function of the d-th factor matrix $\boldsymbol{W}^{(d)}$. Minimizing the risk successively for each factor matrix yields a well-known block coordinate descent (Carroll and Chang, 1970; Harshman, 1970; Kolda and Bader, 2009) algorithm for which each subproblem is convex and exhibits local linear convergence (Uschmajew, 2012, Theorem 3.3). Similar properties hold when constraining $\boldsymbol{w}$ to be a low-rank tensor train decomposition or Tucker decomposition. However, the number of elements in the Tucker decomposition scales exponentially in D, while the tensor train decomposition models explicitly the correlations between features, yielding for the same rank, different models depending on the ordering of the features. In contrast to other decompositions, our CPD-based approach enables to reduce the costs of storage by clever in-place updates. Furthermore, recent theoretical advances in the domain of tensor decomposition show that the VC dimension and pseudodimension of models of the form of Equation (2), where $\boldsymbol{w}$ is a rank-R tensor decomposition, are *independent* of the choice of decomposition of $\boldsymbol{w}$ and instead upper bounded by the number of parameters (Khavari and Rabusseau, 2021, Theorem 7), further motivating the choice of modeling $\boldsymbol{w}$ as a CPD. Following (Wesel and Batselier, 2021), we begin by showing how the risk can be expressed only as a function of $\boldsymbol{W}^{(d)}$. The model term can in fact be rewritten exactly as:

$$\left\langle \boldsymbol{k_{xM}}\boldsymbol{L}^{-\mathrm{T}}, \boldsymbol{w}\right\rangle = \left\langle \boldsymbol{g}^{(d)}\left(\boldsymbol{x}\right), \operatorname{vec}\left(\boldsymbol{W}^{(d)}\right)\right\rangle. \tag{14}$$

Here

$$\boldsymbol{g}^{(d)}\left(\boldsymbol{x}\right) := \operatorname{vec}\left(\boldsymbol{k}_{\boldsymbol{x}\boldsymbol{m}^{(d)}}\boldsymbol{L}^{-\mathrm{T}} \otimes \left(\bigodot_{p \neq d}\left(\boldsymbol{k}_{\boldsymbol{x}\boldsymbol{m}^{(p)}}{\boldsymbol{L}^{(p)}}^{-\mathrm{T}}\right)\right)\right).$$

The regularization term is

$$\langle \boldsymbol{w}, \boldsymbol{w} \rangle = \left\langle \operatorname{vec}\left(\boldsymbol{W}^{(d)^{\mathrm{T}}} \boldsymbol{W}^{(d)} \right), \operatorname{vec}\left(\boldsymbol{H}^{(d)} \right) \right\rangle . \quad (15)$$

where

$$\boldsymbol{H}^{(d)} := \left(\bigodot_{p \neq d} \left(\boldsymbol{W}^{(p)^{\mathrm{T}}} \boldsymbol{W}^{(p)} \right) \right).$$

The derivations of Equation (14) and Equation (15), which hold for $d = 1, 2 \ldots, D$, can be found in the supplementary material. Substitution of Equation (14) and Equation (15) into Equation (12) leads to a convex optimization problem for vec $\left(\boldsymbol{W}^{(d)}\right)$, consisting in practice to training a kernel machine with $RM^{\frac{1}{D}}$ model parameters:

$$\min_{\operatorname{vec}\left(\boldsymbol{W}^{(d)}\right)} \frac{1}{N} \sum_{n=1}^{N} \ell \left(\left\langle \boldsymbol{g}^{(d)}\left(\boldsymbol{x}_n\right), \operatorname{vec}\left(\boldsymbol{W}^{(d)}\right) \right\rangle, y_n \right) + \lambda \left\langle \operatorname{vec}\left(\boldsymbol{W}^{(d)^{\mathrm{T}}} \boldsymbol{W}^{(d)} \right), \operatorname{vec}\left(\boldsymbol{H}^{(d)} \right) \right\rangle, \quad (16)$$

which in case of squared loss can be solved exactly by means of the normal equations

$$\left(\boldsymbol{G}^{(d)^{\mathrm{T}}} \boldsymbol{G}^{(d)} + \lambda N \boldsymbol{H}^{(d)} \otimes \mathrm{I} \right) \operatorname{vec}\left(\boldsymbol{W}^{(d)} \right) = \boldsymbol{G}^{(d)^{\mathrm{T}}} \boldsymbol{y}, \quad (17)$$

where $\left[\boldsymbol{G}^{(d)}\right]_{i,:} = \boldsymbol{g}^{(d)}\left(\boldsymbol{x}_i\right)$. The computational cost of solving Equation (17) exactly is of $\mathcal{O}(NM^{\frac{2}{D}}R^2 + M^{\frac{3}{D}}R^3)$, where the first term accounts for constructing the relevant squared matrices and the second term accounts for solving the linear system. The storage requirements are $\mathcal{O}(M^{\frac{2}{D}}R^2)$ if one builds up $\boldsymbol{G}^{(d)^{\mathrm{T}}}\boldsymbol{G}^{(d)}$ as a series of N rank-1 updates. Alternating the minimization by iterating across all factor matrices, i.e. for $d = 1, 2, \ldots, D$ yields the a block coordinate algorithm which is well studied in the tensor community (Kolda and Bader, 2009; Uschmajew, 2012). Section 4 shows how the algorithm converges to suitable minima in all our experiments and is numerically stable. The computational complexity of our proposed Algorithm CPD-SIP is then $\mathcal{O}(NDM^{\frac{2}{D}}R^2 + DM^{\frac{3}{D}}R^3)$.

Implementation Details Note that we could have considered in Equation (12) the linearly equivalent feature map $\boldsymbol{k_{xM}}$ which e.g. under squared loss and without rank constraints gives rise to the following regularized linear least-squares problem $(\boldsymbol{K_{XM}}^{\mathrm{T}}\boldsymbol{K_{XM}} + \boldsymbol{K_{MM}})\bar{\boldsymbol{w}} = \boldsymbol{K_{XM}}^{\mathrm{T}}\boldsymbol{y}$, where $\bar{\boldsymbol{w}} = \boldsymbol{L}^{-\mathrm{T}}\boldsymbol{w}$. However this formulation is prone to numerical instability, as the singular values of $\boldsymbol{K_{XM}}$ are not scaled by $\boldsymbol{L}$, as discussed in (Rasmussen and Williams, 2006, Chapter 3.4.3). As a consequence when embedded in Algorithm , $\boldsymbol{G}^{(d)}$ has a higher condition number and spirals out of control after a few iterations.

The naive storage cost of $\mathcal{O}(ND(M^{\frac{2}{D}}R^2))$ can be reduced by a factor D by updating $\boldsymbol{G}^{(d)}$ and $\boldsymbol{H}^{(d)}$ in-place and by locating the inducing points on a dimension-independent grid,

Algorithm CPD-Structured Inducing Points (CPD-SIP).

```
Require: Inputs $X \in \mathbb{R}^{N\times D}$, outputs $y \in \mathbb{R}^N$, kernel
    function $k(\cdot,\cdot) : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$, loss $\ell(\cdot,\cdot) : \mathbb{R}\times\mathbb{R} \to \mathbb{R}_+$,
    number of basis $\hat{M} \in \mathbb{N}_+ : \hat{M} := M^{\frac{1}{D}}$, CP-rank
    $R \in \mathbb{N}_+$, max iterations $S \in \mathbb{N}_+$
Ensure: Factor matrices $W^{(d)} \in \mathbb{R}^{\hat{M}\times R}$, $d = 1, 2, \ldots, D$
 1: $s \leftarrow 0$
 2: $G \leftarrow \mathrm{ones}(\hat{M}, R)$
 3: $H \leftarrow \mathrm{ones}(N, R)$
 4: Compute $K_{mm}$ using Equation (6)
 5: $L \leftarrow \mathrm{chol}(K_{mm})$ from Equation (11)
 6: for $d = D, D-1, \ldots, 1$ do
 7:     $W^{(d)} \leftarrow \mathrm{randn}(\hat{M}, R)$
 8:     $W^{(d)} \leftarrow W^{(d)} / ||W^{(d)}||$
 9:     $G \leftarrow G \odot (K_{Xm} L^{-\mathrm{T}} W^{(d)})$
10:     $H \leftarrow H \odot (W^{(d)\mathrm{T}} W^{(d)})$
11: end for
12: repeat
13:     $s \leftarrow s + 1$
14:     for $d = 1, 2, \ldots, D$ do
15:         $H \leftarrow H \oslash (W^{(d)\mathrm{T}} W^{(d)})$
16:         $G \leftarrow G \oslash (K_{Xm} L^{-\mathrm{T}} W^{(d)})$
17:         Solve Equation (16) for vec $(W^{(d)})$
18:         $W^{(d)} \leftarrow \mathrm{reshape}(\mathrm{vec}(W^{(d)}))$
19:         $H \leftarrow H \odot (W^{(d)\mathrm{T}} W^{(d)})$
20:         $G \leftarrow G \odot (K_{Xm} L^{-\mathrm{T}} W^{(d)})$
21:     end for
22: until Convergence or $s = S$
```

see Algorithm Algorithm , where we denote these in-place updated matrices as $\boldsymbol{G}$ and $\boldsymbol{H}$ respectively. Since these operations are $\mathcal{O}(NR)$, they do not affect the computational complexity. The storage complexity can be further reduced by carrying out the updates for $\boldsymbol{G}$ (e.g. in line 9 of Algorithm) in batches of one or more rows of $\boldsymbol{k_{xm}}\boldsymbol{L}^{-\mathrm{T}}$, which can be computed on-the-fly, bringing it down to $\mathcal{O}(NR)$. Of course, if memory is not an issue, speedup up to a constant factor can be easily obtained by caching $\boldsymbol{k_{Xm}}\boldsymbol{L}^{-\mathrm{T}}$, however this is not a requirement by any means. A summary of the computational and storage complexities of various SKI-related methods is given in Table 1, where we can observe that the computational complexity of Algorithm scales linearly in N and D, while having a storage complexity which is independent on D. All methods except the Simplex-GP, whose complexity scales with $\mathcal{O}(D^2)$, place the inducing points on a Cartesian grid. As a result, either an exponential number of computations or R copies of the whole dataset are required, prohibiting to tackle large-dimensional (in this case $D > 20$ (Kapoor et al., 2021)) data.

Selecting the hyperparameters In Algorithm , the CP-rank R is introduced as additional hyperparameter, which similarly to other SKI-based approaches (Wilson and Nickisch, 2015; Gardner et al., 2018) we advocate to select based on the computational budget at hand: M should be chosen first as to provide an accurate representation of the kernel, possibly to machine precision. This can be ensured for instance by means of cross-validation on a portion of unseen data. R can then be chosen in order to fill in the remainder of the available computational and memory budget.

4 EXPERIMENTS

We implemented Algorithm in MathWorks MATLAB. The implementation and instructions to reproduce the results are available at `https://github.com/fwesel/CPD-SIP`. We scale the inputs in order to lie in the unit hypercube $[0,1]^D$. In case of regression problems, we standardize the responses to have zero mean and unit variance. In case of binary classification, we consider only the sign ± 1 of the responses (Suykens and Vandewalle, 1999). We initialize the factor matrices $\boldsymbol{W}^{(d)}$ with standard normal numbers and normalize them to have unit norm. In all our experiments we set the number of iterations to $S = 20$ (consistent with Wesel and Batselier (2021), as we define iterations as half a sweep). All the experiments were run on the Intel Core i7-10610U $1.8\,\mathrm{GHz}$ CPU of a Dell Inc. Latitude 7410 laptop with $16\,\mathrm{GB}$ of RAM. In what follows we present a series of three numerical experiments. Therein we demonstrate how our algorithm is stable and recovers the underlying KRR baseline with small values of R. We show how it compares with other grid-structured approaches managing to extend their applicability to data large in dimensionality ($D = 384$) or sample-size ($N = 5\,000\,000$).

4.1 Non-Stationary Kernel

The Banana dataset (Hensman et al., 2015) is a two-dimensional binary classification dataset which is often used in the context of low-rank kernel machines to visually demonstrate their characteristics. The dataset comprises $N = 5300$ data points roughly split in two classes. We consider the non-stationary separable polynomial kernel $k(\boldsymbol{x},\boldsymbol{x'}) = \prod_{d=1}^{D}(1 + x_d x'_d)^5$, $\lambda = {}^{1\times10^{-6}}/_{N}$ and consider $M = 2500$ inducing points, 50 per dimension, located on an equidistant Cartesian grid. We then proceed to train a Kernel Ridge Regression classifier of Equation (4) and Algorithm with the same hyperparameter λ. Since the problem is two-dimensional, $\boldsymbol{w}$ is a matrix and has rank $R = M^{\frac{1}{D}} = 50$. In Figure 1 we can observe that for low values of R the classification boundary is similar to the one of the KRR baseline where there is more data. This is because Algorithm seeks to minimize the empirical risk, and when provided with very few parameters it will seek to improve the classification boundary where the data is denser. We notice that already for $R = 6$ the classification boundary is indistinguishable from the one of KRR. As we will see next, the assumption of a small rank is valid also when dealing with higher-dimensional and large-sampled data.

4.2 Comparison with SKI

In order to compare our method with SKI, we consider seven UCI datasets (Dua and Graff, 2017), five of which are considered also by Kapoor et al. (2021). We compare our approach against SKIP (Gardner et al., 2018) and Simplex-GP (Kapoor et al., 2021). We consider the Gaussian kernel and locate $M = 10^D$ inducing points on a equidistant Cartesian grid and model $\boldsymbol{w}$ as a rank-20 CPD. In order to train a model in approximately the same function space, we select our hyperparameters l and λ by means of maximizing the log-likelihood of an exact GP model (Rasmussen and Nickisch, 2010) constructed on a small random uniform subset of 2000 points. We then validate our model by means of 3-fold cross validation and report in Table 2 the Standardized Root Mean Squared Error (RMSE) with one standard deviation. While training we keep track of the quality of our inducing-point approximation $\boldsymbol{k_{xM}}\boldsymbol{L}^{-\mathrm{T}}$ by sampling uniformly at random a subset $\boldsymbol{E}$ of 1000 points and computing the relative error ${}^{||\boldsymbol{K_{EE}}-\boldsymbol{K_{EM}}\boldsymbol{L}^{-\mathrm{T}}\boldsymbol{L}^{-1}\boldsymbol{K_{ME}}||}/_{||\boldsymbol{K_{EE}}||}$, which we report in Table 2. Here we can observe that the quality of the approximation approaches machine precision on many datasets, allowing the modeler to chose R according to the remaining computational budget.

In Table 2 we can observe that notwithstanding the sub-optimal choice of hyperparameters, our model is competitive in term of performance with the other inducing-points based approaches. Notably, although the seven considered datasets range vastly in the number of samples, they do not do so in the dimensionality, as all methods pay a heavy computational or storage-related price when scaling to higher-dimensional data. This is not the case of Algorithm which contrary to SKIP, does not *require* the contemporary storage of D $N \times R$ matrices, which allows us to tackle datasets of large dimensionality such as Slice with $D = 384$. Training our model on the laptop CPU requires then $(11\,274 \pm 189)\,\mathrm{s}$ for the Song dataset and $(4724 \pm 198)\,\mathrm{s}$ for the HouseElectric, compared to a per-epoch $(1075 \pm 176)\,\mathrm{s}$ of Simplex-GP (Kapoor et al., 2021, Table 4) on a Titan RTX GPU with $24\,\mathrm{GB}$ of RAM. Training on the Slice dataset took $(226 \pm 2)\,\mathrm{s}$.

4.3 Large-Scale Classification

In order to demonstrate the favorable complexity of Algorithm when dealing with a larger number of samples, we consider the SUperSYmmetry dataset (SUSY) (Baldi et al., 2014; Dua and Graff, 2017), an binary classification 18-dimensional dataset consisting of $5\,000\,000$ samples, whose first 8 features consist of particle detector measurements, while the following 10 are high-level features engineered from the first 8. We consider $M = 20^D$ inducing points,

Table 1: Computational and storage complexities of various SKI-based approaches when exploiting stationary structure. For SKIP R is typically chosen to be between 20 and 100 (Kapoor et al., 2021).

Method	Complexities	
	Computational	Storage
KRR (Suykens, 2002)	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$
SKI (Wilson and Nickisch, 2015; Yadav et al., 2021)	$\mathcal{O}(N + M \log M)$	$\mathcal{O}(NM)$
SKIP (Gardner et al., 2018)	$\mathcal{O}(NDR + RM^{\frac{1}{D}} \log M + NR^3 \log D + NR^2)$	$\mathcal{O}(DNR)$
Simplex-GP (Kapoor et al., 2021)	$\mathcal{O}(ND + MD^2))$	$\mathcal{O}(MD)$
Algorithm	$\mathcal{O}(D(NM^{\frac{2}{D}}R^2 + M^{\frac{3}{D}}R^3))$	$\mathcal{O}(NR)$

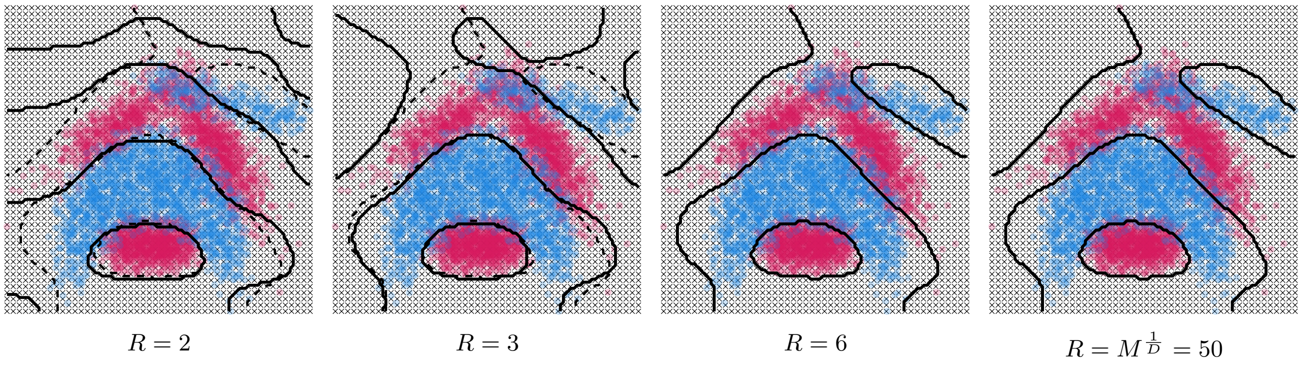

Figure 1: Classification boundary of the two-dimensional Banana dataset for increasing CPD-ranks R for the non-stationary product kernel $k(\boldsymbol{x}, \boldsymbol{x'}) = \prod_{d=1}^{D}(1 + x_d x'_d)^5$. The dashed line is the KRR decision boundary while the full line corresponds to Algorithm . The black crosses are the locations of the inducing points. In the last plot the chosen CP-rank matches the true (matrix) rank of $\boldsymbol{w}$.

and $R = 5, 10, 15, 20$. As is standard on this dataset, training is performed on the first 4 500 000 points and test on the remainder. We train both on only low-level and low-level plus high-level features. We use the Guassian kernel with l as the mean of the standard deviations of the features and $\lambda = {}^{2\times10^{-5}}/{}_{N}$ and report in Table 3 the Area Under the Curve (AUC), misclassification error and training time of our and other methods in literature. In Table 3 we can see that already for $R = 5$ our Algorithm scores similarly to VISH (Dutordoir et al., 2020), whose reliance on numerically unstable spherical harmonics prohibits it however to be deployed on data with $D > 9$. For higher values of R, the performance rivals with DNNs. Others results on the dataset are from (Chen et al., 2017) where the authors obtain a misclassification rate of 20.1 % in 2400 s on a cluster with IBM POWER8 12-core CPUs and 512 GB RAM.

5 CONCLUSION

In this work we build on the idea of placing inducing points on a Cartesian grid in order to exploit the computationally favorable arising tensorial structure. This allows us to obtain a good approximation of the kernel function on the whole domain, without sacrificing accuracy or the ability to tackle large-dimensional problems. In contrast to SKI and inducing points-related literature, we are in fact able to overcome the curse of dimensionality which affects both computational and storage-related complexities of these structured approaches by modeling the weights as a rank-R CPD decomposition. We show by means of numerical experiments how our approach is viable even on modest hardware. Note that all operations in Algorithm can be expressed as a series of matrix-vector products, enabling for efficient (multi-)GPU implementations. Furthermore, since our approach allows to learn from any product kernel, it allows to consider the SKI kernel of Equation (9), which can be a product kernel depending on the the choice of interpolation strategy (Wilson and Nickisch, 2015). This could then allow for cheap caching of the features and further speedup by exploiting to the sparse structure in combination with stationary product kernels. One limitation of our approach is that its computational complexity scales with $\mathcal{O}(ND)$, prohibiting its application to data with a large number of samples *and* dimensionality, e.g. in case of categorical features. Another limitation, which we did not encounter in the experiments, is that the low-rank hypothesis is certainly not always justified, especially when dealing with highly complicated functions. We think that a possible remedy might be to seek for a different kernel space where the low-rank assumption would hold. Furthermore, although our approach is inherently non-probabilistic, our work allows to approximately carry out one of the two GP tasks, namely

Table 2: Predictive Standardized RMSE with one standard deviation on five UCI datasets (Kapoor et al., 2021, Table 2)

Dataset			RMSE			Rel. Approx. Error
	$D\downarrow$	N	SKIP	Simplex GP	Algorithm	Algorithm
Precipitation	3	$628\,474$	1.032 ± 0.001	$\mathbf{0.939 \pm 0.001}$	0.974 ± 0.000	$(0.81 \pm 1.03) \times 10^{-2}$
Protein	9	$45\,730$	0.817 ± 0.012	$\mathbf{0.571 \pm 0.003}$	0.705 ± 0.004	$(1.94 \pm 1.74) \times 10^{-2}$
HouseElectric	11	$2\,049\,280$	NA	$\mathbf{0.079 \pm 0.002}$	0.084 ± 0.002	$(1.63 \pm 8.24) \times 10^{-14}$
Elevators	17	$16\,599$	0.447 ± 0.037	0.510 ± 0.018	$\mathbf{0.382 \pm 0.005}$	$(6.37 \pm 4.67) \times 10^{-15}$
KeggDirected	20	$48\,827$	0.487 ± 0.005	0.095 ± 0.002	$\mathbf{0.089 \pm 0.001}$	$(3.17 \pm 0.81) \times 10^{-13}$
Song	90	$515\,345$	NA	NA	$\mathbf{0.800 \pm 0.003}$	$(2.40 \pm 1.40) \times 10^{-5}$
Slice	386	$53\,500$	NA	NA	$\mathbf{0.094 \pm 0.002}$	$(2.79 \pm 1.65) \times 10^{-12}$

Table 3: Predictive AUC, misclassification rate and training time with one standard deviation on the SUSY dataset. Results for Bayesian decision tree (BDT) and neural networks (NNs) are from (Baldi et al., 2014, Table 2), while the result for VISH is from (Dutordoir et al., 2020, Table 3).

	AUC		1-Accuracy (%)	Time (s)
Technique	Low-level	Complete	Complete	Complete
BDT	0.850 ± 0.003	0.863 ± 0.003	NA	NA
NN	0.867 ± 0.002	0.875 ± 0.001	NA	NA
Dropout NN	0.856 ± 0.001	0.873 ± 0.001	NA	NA
DNN	0.872 ± 0.001	0.876 ± 0.001	NA	NA
Dropout DNN	$\mathbf{0.876 \pm 0.001}$	$\mathbf{0.879 \pm 0.001}$	NA	NA
VISH	0.859 ± 0.001	NA	NA	NA
Algorithm ($R = 5$)	0.862 ± 0.002	0.872 ± 0.002	20.04 ± 0.01	1641 ± 21
Algorithm ($R = 10$)	0.867 ± 0.000	0.874 ± 0.000	19.82 ± 0.01	4650 ± 15
Algorithm ($R = 15$)	0.872 ± 0.000	0.875 ± 0.000	19.74 ± 0.00	6773 ± 52
Algorithm ($R = 20$)	0.872 ± 0.000	0.876 ± 0.000	19.68 ± 0.01	9446 ± 23

data fitting. Interesting further directions would be to investigate the regularizing effects of the low-rank constraint, to incorporate this exact approach in a probabilistic framework allowing for uncertainty quantification and possibly Bayesian model selection.

6 Acknowledgements

We would like to thank the anonymous reviewers for their suggestions and improvements. Frederiek Wesel, and thereby this work, is supported by the Delft University of Technology AI Labs program. The authors declare no competing interests.

References

P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5(1):4308, July 2014. [6, 8]

K. Batselier, Z. Chen, and N. Wong. Tensor Network alternating linear scheme for MIMO Volterra system identification. *Automatica*, 84:26–35, Oct. 2017. [3]

J. D. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3):283–319, Sept. 1970. [4]

J. Chen, H. Avron, and V. Sindhwani. Hierarchically Compositional Kernels for Scalable Nonparametric Learning. *Journal of Machine Learning Research*, 18(66):1–42, 2017. [7]

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. [3]

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sept. 1995. [1]

L. Csató and M. Opper. Sparse On-Line Gaussian Processes. *Neural Computation*, 14(3):641–668, Mar. 2002. [1]

D. Dua and C. Graff. UCI Machine Learning Repository, 2017. [6]

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. [7, 8]

T. Evans and P. Nair. Scalable Gaussian Processes with Grid-Structured Eigenfunctions (GP-GRIEF). In *International Conference on Machine Learning*, pages 1417–1426. PMLR, July 2018. [4]

G. Favier and T. Bouilloc. Parametric complexity reduction of Volterra models using tensor decompositions. In *2009 17th European Signal Processing Conference*, pages 2288–2292, Aug. 2009. [3]

J. Gardner, G. Pleiss, R. Wu, K. Weinberger, and A. Wilson. Product Kernel Interpolation for Scalable Gaussian Processes. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, pages 1407–1416. PMLR, Mar. 2018. [1, 3, 5, 6, 7]

A. Garriga-Alonso, C. E. Rasmussen, and L. Aitchison. Deep Convolutional Networks as shallow Gaussian Processes. In *International Conference on Learning Representations*, Sept. 2018. [1]

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. [2]

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. [1]

R. A. Harshman. Foundations of the PARAFAC procedure : Models and conditions for an. *UCLA Working Papers in Phonetics*, 16:1–84, 1970. [4]

J. Hensman, A. Matthews, and Z. Ghahramani. Scalable Variational Gaussian Process Classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR, Feb. 2015. [6]

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. [3]

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. [4]

S. Kapoor, M. Finzi, K. A. Wang, and A. G. G. Wilson. SKIing on Simplices: Kernel Interpolation on the Permutohedral Lattice for Scalable Gaussian Processes. In *Proceedings of the 38th International Conference on Machine Learning*, pages 5279–5289. PMLR, July 2021. [3, 5, 6, 7, 8]

B. Khavari and G. Rabusseau. Lower and Upper Bounds on the Pseudo-Dimension of Tensor Network Models. In *Advances in Neural Information Processing Systems*, May 2021. [4]

M. Khodak, N. A. Tenenholtz, L. Mackey, and N. Fusi. Initialization and Regularization of Factorized Neural Layers. In *International Conference on Learning Representations*, Sept. 2020. [3]

T. G. Kolda and B. W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 51(3):455–500, Aug. 2009. [1, 3, 4, 5]

V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. In *ICLR (Poster)*, Jan. 2015. [3]

J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep Neural Networks as Gaussian Processes. In *International Conference on Learning Representations*, Feb. 2018. [1]

G. Meanti, L. Carratino, L. Rosasco, and A. Rudi. Kernel methods through the roof: Handling billions of points efficiently. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14410–14422. Curran Associates, Inc., 2020. [1]

T. Nickson, T. Gunter, C. Lloyd, M. A. Osborne, and S. Roberts. Blitzkriging: Kronecker-Structured Stochastic Gaussian Processes. *arXiv:1510.07965 [stat]*, Oct. 2015. [1]

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*, Sept. 2018. [1]

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. [3]

A. Novikov, I. Oseledets, and M. Trofimov. Exponential machines. *Bulletin of the Polish Academy of Sciences: Technical Sciences; 2018; 66; No 6 (Special Section on Deep Learning: Theory and Practice); 789-797*, 2018. [3]

I. V. Oseledets. Tensor-Train Decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, Jan. 2011. [3]

J. Quiñonero-Candela and C. E. Rasmussen. A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of Machine Learning Research*, 6(65):1939–1959, 2005. [1]

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. [1]

C. E. Rasmussen and H. Nickisch. Gaussian Processes for Machine Learning (GPML) Toolbox. *Journal of Machine Learning Research*, 11(100):3011–3015, 2010. [6]

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass, 2006. [1, 5]

Y. Saatchi. *Scalable Inference for Structured Gaussian Process Models*. PhD thesis, University of Cambridge, Cambridge, Nov. 2011. [2, 4, 12]

B. Schölkopf, R. Herbrich, and A. J. Smola. A Generalized Representer Theorem. In D. Helmbold and B. Williamson, editors, *Computational Learning Theory*, Lecture Notes in Computer Science, pages 416–426, Berlin, Heidelberg, 2001. Springer. [2]

N. D. Sidiropoulos and R. Bro. On the uniqueness of multilinear decomposition of N-Way arrays. *Journal of Chemometrics*, 14(3):229–239, 2000. [4]

A. Smola and P. Bartlett. Sparse Greedy Gaussian Process Regression. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2001. [1]

E. Snelson and Z. Ghahramani. Sparse Gaussian Processes using Pseudo-inputs. In *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2006. [1]

S. Stanton, W. Maddox, I. Delbridge, and A. G. Wilson. Kernel Interpolation for Scalable Online Gaussian Processes. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 3133–3141. PMLR, Mar. 2021. [1, 3]

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. [3]

J. Suykens and J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3): 293–300, June 1999. [6]

J. A. K. Suykens, editor. *Least Squares Support Vector Machines*. World Scientific, River Edge, NJ, 2002. [3, 7]

M. Titsias. Variational Learning of Inducing Variables in Sparse Gaussian Processes. In *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, pages 567–574. PMLR, Apr. 2009. [1]

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. [3]

L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, Sept. 1966. [3]

A. Uschmajew. Local Convergence of the Alternating Least Squares Algorithm for Canonical Tensor Approximation. *SIAM Journal on Matrix Analysis and Applications*, 33 (2):639–652, Jan. 2012. [4, 5]

F. Wesel and K. Batselier. Large-Scale Learning with Fourier Features and Tensor Decompositions. In *Advances in Neural Information Processing Systems*, May 2021. [2, 3, 4, 6, 12]

C. Williams and M. Seeger. Using the Nyström Method to Speed Up Kernel Machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001. [1, 3]

A. Wilson and H. Nickisch. Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1775–1784. PMLR, June 2015. [1, 3, 4, 5, 7]

M. Yadav, D. Sheldon, and C. Musco. Faster Kernel Interpolation for Gaussian Processes. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 2971–2979. PMLR, Mar. 2021. [1, 3, 7]

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. [3]

Z. Yang, A. Wilson, A. Smola, and L. Song. A la Carte – Learning Fast Kernels. In *Artificial Intelligence and Statistics*, pages 1098–1106. PMLR, Feb. 2015. [1]

S. Zhe, W. Xing, and R. M. Kirby. Scalable High-Order Gaussian Process Regression. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, pages 2611–2620. PMLR, Apr. 2019. [3]

7 Kronecker Structures

7.1 Kronecker Product Structure of K_{MM} Saatchi (2011)

Let us consider product kernels, where the inducing points $\boldsymbol{m}_i, \boldsymbol{m}_j \in \mathbb{R}^D$ live on a Cartesian grid $\boldsymbol{m}^{(1)} \times \boldsymbol{m}^{(2)} \times \cdots \boldsymbol{m}^{(D)}$ where $\boldsymbol{m}^{(d)} \in \mathbb{R}^{M_d}$. Then:

$$[\boldsymbol{K_{MM}}]_{i,j} = k(\boldsymbol{m}_i, \boldsymbol{m}_j) = k_1(m^{(1)}_{i_1}, m^{(1)}_{j_1}) k_2(m^{(2)}_{i_1}, m^{(2)}_{j_2}) \cdots k_D(m^{(D)}_{i_D}, m^{(D)}_{j_D}),$$

where $i = i_1 + \sum_{d=2}^{D}(i_d - 1)\prod_{k=1}^{d-1} M_k$ and $j = j_1 + \sum_{d=2}^{D}(j_d - 1)\prod_{k=1}^{d-1} M_k$ with $i_d = 1, \ldots, M_d$ and $j_d = 1, \ldots, M_d$. The definition of the Kronecker product implies that

$$\boldsymbol{K_{MM}} = \boldsymbol{K}_{\boldsymbol{m}^{(1)}\boldsymbol{m}^{(1)}} \otimes \boldsymbol{K}_{\boldsymbol{m}^{(2)}\boldsymbol{m}^{(2)}} \otimes \cdots \otimes \boldsymbol{K}_{\boldsymbol{m}^{(D)}\boldsymbol{m}^{(D)}},$$

where $\boldsymbol{K}_{\boldsymbol{m}^{(d)}\boldsymbol{m}^{(d)}} \in \mathbb{R}^{M_d \times M_d}$.

7.2 Row-wise Khatri-Rao Product Structure of K_{XM}

Let us consider product kernels, where the inducing points $\boldsymbol{m}_j \in \mathbb{R}^D$ live on a Cartesian grid $\boldsymbol{m}^{(1)} \times \boldsymbol{m}^{(2)} \times \cdots \boldsymbol{m}^{(D)}$ where $\boldsymbol{m}^{(d)} \in \mathbb{R}^{M_d}$. Then:

$$[\boldsymbol{K_{XM}}]_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{m}_j) = k_1(x_{i_1}, m^{(1)}_{j_1}) k_2(x_{i_2}, m^{(2)}_{j_2}) \cdots k_D(x_{i_D}, m^{(D)}_{j_D}),$$

where $j = j_1 + \sum_{d=2}^{D}(j_d - 1)\prod_{k=1}^{d-1} M_k$, $j_d = 1, \ldots, M_d$ and $i = 1, \ldots, N$. The definition of the Kronecker product implies that

$$[\boldsymbol{K_{XM}}]_{i,:} = \boldsymbol{k}_{\boldsymbol{x}_i \boldsymbol{M}} = \boldsymbol{k}_{x_{i_1}\boldsymbol{m}^{(1)}} \otimes \boldsymbol{k}_{x_{i_2}\boldsymbol{m}^{(2)}} \otimes \cdots \otimes \boldsymbol{k}_{x_{i_D}\boldsymbol{m}^{(D)}}.$$

where $\boldsymbol{k}_{x_{i_d}\boldsymbol{m}^{(d)}} \in \mathbb{R}^{M_d}$.

7.3 Model and Regularization Terms

Following (Wesel and Batselier, 2021) we make use of the multi-linearity property of the CPD and rely on re-ordering the summations. Here $M = \prod_{d=1}^{D} M_d$.

$$\begin{aligned}
\left\langle \boldsymbol{k_{xM}} \boldsymbol{L}^{-\mathrm{T}}, \boldsymbol{w} \right\rangle &= \left\langle \boldsymbol{k}_{x_1\boldsymbol{m}^{(1)}} {\boldsymbol{L}^{(1)}}^{-\mathrm{T}} \otimes \boldsymbol{k}_{x_2\boldsymbol{m}^{(2)}} {\boldsymbol{L}^{(2)}}^{-\mathrm{T}} \otimes \cdots \otimes \boldsymbol{k}_{x_D\boldsymbol{m}^{(D)}} {\boldsymbol{L}^{(D)}}^{-\mathrm{T}}, \sum_{r=1}^{R} \boldsymbol{w}_r^{(1)} \otimes \boldsymbol{w}_r^{(2)} \otimes \cdots \otimes \boldsymbol{w}_r^{(D)} \right\rangle \\
&= \sum_{m_1=1}^{M_1} \cdots \sum_{m_d=1}^{M_d} \cdots \sum_{m_D=1}^{M_D} \sum_{r=1}^{R} \boldsymbol{k}_{x_1\boldsymbol{m}^{(1)}} {\boldsymbol{L}^{(1)}}^{-\mathrm{T}}_{:m_1} w^{(1)}_{m_1 r} \cdots \boldsymbol{k}_{x_d\boldsymbol{m}^{(d)}} {\boldsymbol{L}^{(d)}}^{-\mathrm{T}}_{:m_d} w^{(d)}_{m_d r} \cdots \boldsymbol{k}_{x_D\boldsymbol{m}^{(D)}} {\boldsymbol{L}^{(D)}}^{-\mathrm{T}}_{:m_D} w^{(D)}_{m_D r} \\
&= \sum_{m_d=1}^{M_d} \sum_{r=1}^{R} \left(\boldsymbol{k}_{x_d\boldsymbol{m}^{(d)}} {\boldsymbol{L}^{(D)}}^{-\mathrm{T}}_{:m_d} \sum_{m_1=1}^{M_1} \boldsymbol{k}_{x_1\boldsymbol{m}^{(1)}} {\boldsymbol{L}^{(1)}}^{-\mathrm{T}}_{:m_1} w^{(1)}_{m_1 r} \cdots \sum_{m_D=1}^{M_D} \boldsymbol{k}_{x_D\boldsymbol{m}^{(D)}} {\boldsymbol{L}^{(D)}}^{-\mathrm{T}}_{:m_D} w^{(D)}_{m_D r} \right) w^{(d)}_{m_d r} \\
&= \operatorname{vec}\left(\boldsymbol{k}_{x_d\boldsymbol{m}^{(d)}} {\boldsymbol{L}^{(d)}}^{-\mathrm{T}} \otimes \left(\boldsymbol{k}_{x_1\boldsymbol{m}^{(1)}} {\boldsymbol{L}^{(1)}}^{-\mathrm{T}} \boldsymbol{W}^{(1)} \odot \cdots \odot \boldsymbol{k}_{x_D\boldsymbol{m}^{(D)}} {\boldsymbol{L}^{(D)}}^{-\mathrm{T}} \boldsymbol{W}^{(D)} \right) \right)^{\mathrm{T}} \operatorname{vec}\left(\boldsymbol{W}^{(d)} \right) \\
&= \left\langle \boldsymbol{g}^{(d)}(\boldsymbol{x}), \operatorname{vec}\left(\boldsymbol{W}^{(d)} \right) \right\rangle
\end{aligned}$$

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

$$\begin{aligned}
\langle \boldsymbol{w}, \boldsymbol{w} \rangle &= \left\langle \sum_{r=1}^{R} \boldsymbol{w}_r^{(1)} \otimes \boldsymbol{w}_r^{(2)} \otimes \cdots \otimes \boldsymbol{w}_r^{(D)}, \sum_{r=1}^{R} \boldsymbol{w}_r^{(1)} \otimes \boldsymbol{w}_r^{(2)} \otimes \cdots \otimes \boldsymbol{w}_r^{(D)} \right\rangle \\
&= \sum_{m_1=1}^{M_1} \cdots \sum_{m_d=1}^{M_d} \cdots \sum_{m_D=1}^{M_D} \sum_{r=1}^{R} \sum_{p=1}^{R} w^{(1)}_{m_1 r} w^{(1)}_{m_1,p} \cdots w^{(d)}_{m_d r} w^{(d)}_{m_d,p} \cdots w^{(D)}_{m_D r} w^{(D)}_{p i_D} \\
&= \sum_{r=1}^{R} \sum_{p=1}^{R} \left(\sum_{m_d=1}^{M_d} w^{(d)}_{m_d r} w^{(d)}_{m_d p} \right) \left(\sum_{m_1=1}^{M_1} w^{(1)}_{m_1 r} w^{(1)}_{m_1 p} \cdots \sum_{m_D=1}^{M_D} w^{(D)}_{m_D r} w^{(D)}_{m_D p} \right)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{r=1}^{R} \sum_{p=1}^{R} \left(\boldsymbol{w}_r^{(d)^{\mathsf{T}}} \boldsymbol{w}_p^{(d)} \right) \left(\boldsymbol{w}_r^{(1)^{\mathsf{T}}} \boldsymbol{w}_p^{(1)} \odot \cdots \odot \boldsymbol{w}_r^{(D)^{\mathsf{T}}} \boldsymbol{w}_p^{(D)} \right) \\
&= \operatorname{vec}\left(\boldsymbol{W}^{(d)^{\mathsf{T}}} \boldsymbol{W}^{(d)} \right)^{\mathsf{T}} \operatorname{vec}\left(\boldsymbol{W}^{(1)^{\mathsf{T}}} \boldsymbol{W}^{(1)} \odot \cdots \odot \boldsymbol{W}^{(D)^{\mathsf{T}}} \boldsymbol{W}^{(D)} \right) \\
&= \left\langle \operatorname{vec}\left(\boldsymbol{W}^{(d)^{\mathsf{T}}} \boldsymbol{W}^{(d)} \right), \operatorname{vec}\left(\boldsymbol{H}^{(d)} \right) \right\rangle
\end{aligned}
$$