Tensor Network Based Feature Learning Model

Saiapin, Albert; Batselier, Kim

Publication date
2025

Document Version
Final published version

Published in
Proceedings of Machine Learning Research

Tensor Network Based Feature Learning Model

Albert Saiapin
Delft Center for Systems and Control
Delft University of Technology

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

Abstract

Many approximations were suggested to circumvent the cubic complexity of kernel-based algorithms, allowing their application to large-scale datasets. One strategy is to consider the primal formulation of the learning problem by mapping the data to a higher-dimensional space using tensor-product structured polynomial and Fourier features. The curse of dimensionality due to these tensor-product features was effectively solved by a tensor network reparameterization of the model parameters. However, another important aspect of model training — identifying optimal feature hyperparameters — has not been addressed and is typically handled using the standard cross-validation approach. In this paper, we introduce the Feature Learning (FL) model, which addresses this issue by representing tensor-product features as a learnable Canonical Polyadic Decomposition (CPD). By leveraging this CPD structure, we efficiently learn the hyperparameters associated with different features alongside the model parameters using an Alternating Least Squares (ALS) optimization method. We prove the effectiveness of the FL model through experiments on real data of various dimensionality and scale. The results show that the FL model can be consistently trained 3-5 times faster than and have the prediction quality on par with a standard cross-validated model.

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

1 INTRODUCTION

In the supervised learning setting, there are two main goals: to estimate a function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ given the data $T = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, where $\boldsymbol{x}_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$, and to find the optimal model hyperparameters $\boldsymbol{\theta}$.

As a solution to the first problem, kernel methods, such as Support Vector Machines (SVMs) (Cortes and Vapnik, 1995) and Gaussian Processes (GPs) (Rasmussen and Williams, 2005), were developed and have become a prominent class of machine learning techniques designed to infer non-linear functions. Due to a well-established mathematical framework in terms of Reproducing Kernel Hilbert Spaces (Ghojogh et al., 2021) and theoretical guarantees of convex optimization theory, kernel methods have become widely studied and applied in different problem domains. Moreover, they can be considered universal function approximators when an appropriate kernel is chosen (Hammer and Gersmann, 2002), and recent research highlights some connections with neural networks (Lee et al., 2018; Novak et al., 2019). However, a major limitation of kernel machines is the need to compute the kernel matrix, which captures pairwise similarities between data points in the feature space. This computation has a complexity of at least $\mathcal{O}(N^3)$, making the method impractical for large datasets ($N \approx 10^7 - 10^9$) on low-performance computing systems.

One way to address this problem is to consider the primal formulation of the learning task. In this case, the data is mapped into a high-dimensional feature space, where linear inference is performed:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\phi}_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \boldsymbol{w}. \tag{1}$$

The mapping $\boldsymbol{\phi}_{\boldsymbol{\theta}}(\boldsymbol{x})$ has a significant role as it allows for modeling various non-linear behaviors in the data. This paper focuses on tensor-product features, defined as:

$$\boldsymbol{\phi}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\psi}_{\boldsymbol{\theta}}^{(D)}(x_D) \otimes \cdots \otimes \boldsymbol{\psi}_{\boldsymbol{\theta}}^{(1)}(x_1), \tag{2}$$

where $\boldsymbol{\psi}_{\boldsymbol{\theta}}^{(d)} : \mathbb{C} \to \mathbb{C}^{I_d}$ is a feature map acting on the d-th component of $\boldsymbol{x} \in \mathbb{C}^D$ and $\otimes$ denotes the Kronecker

product (Kolda and Bader, 2009). The same tensor-product structure is related to product kernels (Hensman et al., 2017; Solin and Särkkä, 2019), Fourier features (Wahls et al., 2014) and polynomials (Shawe-Taylor and Cristianini, 2004). At first glance, a disadvantage of the tensor-product structure in Equation (2) is that the input vector $\boldsymbol{x}$ is mapped into an exponentially large feature vector $\boldsymbol{\phi_\theta}(\boldsymbol{x}) \in \mathbb{C}^{I_1 I_2 \ldots I_D}$. As a result, the model is also described by an exponential number of model parameters $\boldsymbol{w}$. This exponential scaling in the number of features limits the use of tensor-product features to low-dimensional data or to the mappings of low degree.

One way to take advantage of the existing tensor-product structure in Equation (2) is by imposing a tensor network (Kolda and Bader, 2009; Cichocki et al., 2016) constraint on the $\boldsymbol{w}$ parameters. For example, using a polyadic rank-R constraint reduces the storage complexity of the model parameters from $\mathcal{O}(I^D)$ down to $\mathcal{O}(DIR)$ and enables the development of efficient learning algorithms with a computational complexity of $\mathcal{O}(DIR)$ per gradient descent iteration, where $I = \max(I_1, \ldots, I_D)$. This idea has been explored for polynomials (Rendle, 2010; Batselier et al., 2017; Blondel et al., 2017), pure-power-1 polynomials (Novikov et al., 2016), pure-power polynomials of higher degree (Chen et al., 2016; Wesel and Batselier, 2024) and Fourier features (Wahls et al., 2014; Stoudenmire and Schwab, 2016; Kargas and Sidiropoulos, 2021; Cheng et al., 2021; Wesel and Batselier, 2021, 2024). In this paper, we focus on Fourier features constructed from Fourier basis functions: $\boldsymbol{\psi}_\theta(x) = \left[e^{-\frac{2\pi j x k}{\theta}}\right]_{k=0}^{I_d-1}$, where k is a basis frequency and θ is a hyperparameter. In addition, $\boldsymbol{w}$ is expressed as a Canonical Polyadic Decomposition (CPD) following (Wesel and Batselier, 2021, 2024).

The second challenge in supervised learning - hyperparameters search - is often hardly addressed (Stoudenmire and Schwab, 2016; Chen et al., 2016; Cheng et al., 2021; Wesel and Batselier, 2021, 2024) and typically relies on application of generic cross-validation techniques (Refaeilzadeh et al., 2009). However, these techniques do not exploit the specific structure of the model.

In this article, we introduce a new model that enables us to avoid the use of cross-validation, which is typically employed to determine the optimal value of the θ hyperparameter. Instead, our model considers a set of P different values of θ simultaneously, using an additional tensor-product structure (2), which is leveraged to derive a computationally efficient training algorithm. Our new Feature Learning (FL) model is described by

$$f(\boldsymbol{x}) = \left[\sum_{p=1}^{P} \lambda_p \, \boldsymbol{\phi}_{\theta_p}(\boldsymbol{x})\right]^\top \boldsymbol{w},$$

which constructs the feature map as a linear combination of P different tensor-product feature maps $\boldsymbol{\phi}_{\theta_p}$. The λ_p parameters can then be interpreted as a measure of the importance of a particular hyperparameter θ_p. The linear combination of features $\boldsymbol{\phi}_{\theta_p}(\boldsymbol{x})$ is a CPD itself. A key benefit of our FL model is that the λ_p parameters can also be efficiently learned from data. Exploiting the CPD structure of both $\boldsymbol{w}$ and the feature map allows us to improve the computational complexity of training the FL model by splitting the main non-linear optimization problem into a series of much smaller linear problems. This reduces the training computational complexity from $\mathcal{O}(I^{2D}[N + I^D])$ to $\mathcal{O}(\mathcal{E}DNIR[P + IR])$ for the large-scale high-dimensional problem, where $\mathcal{E}$ is the number of training epochs, N is the number of training samples, D is the dimensionality of data, I is the maximum dimensionality of a feature map $\boldsymbol{\psi}_\theta$, R is a CPD rank, P is the number of different feature maps.

Furthermore, our numerical experiments demonstrate that the FL model can be effectively applied to datasets that are large both in sample size and dimensionality. Compared to the standard cross-validation technique, the FL model trains 3–5 times faster without any significant drop in prediction performance. In addition to introducing the FL model, we also explore different forms of $\boldsymbol{\lambda}$ regularization and highlight their respective advantages, disadvantages and use cases.

2 BACKGROUND

We denote scalars in italics w, W, vectors in non-capital bold $\boldsymbol{w}$, matrices in capital bold $\boldsymbol{W}$ and tensors, also being high-order arrays, in capital italic bold font $\boldsymbol{\mathcal{W}}$. Sets are denoted with calligraphic capital letters - $\mathcal{Z}$. A range of natural numbers from 1 to N is denoted as $\overline{1, N}$. The i-th entry of a vector $\boldsymbol{w} \in \mathbb{C}^I$ is denoted as w_i and the $i_1 i_2 \ldots i_D$-th entry of a D-order tensor $\boldsymbol{\mathcal{W}} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_D}$ as $w_{i_1 i_2 \ldots i_D}$. The conjugate-transpose of $\boldsymbol{A}$ is denoted as $\boldsymbol{A}^\top$ and $\otimes$, $\odot_R$, $\circledast$, $\oslash$ represent the Kronecker product, row-wise Khatri-Rao product, Hadamard product, element-wise division of matrices correspondingly (Kolda and Bader, 2009; Cichocki et al., 2016). Further we define two useful interconnected operators from tensor linear algebra: vectorization and tensorization.

Definition 2.1 (Vectorization)**.** The vectorization operator $\text{vec}\,(\cdot) : \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_D} \to \mathbb{C}^{I_1 I_2 \ldots I_D}$ such that:

$$\text{vec}\,(\boldsymbol{\mathcal{W}})_i = w_{i_1 i_2 \ldots i_D},$$

where $i = i_1 + \sum_{d=2}^{D} i_d \prod_{j=1}^{d-1} I_j$.

Definition 2.2 (Tensorization)**.** The tensorization operator $\text{ten}(\cdot, I_1, I_2, \ldots, I_D) : \mathbb{C}^{I_1 I_2 \ldots I_D} \to \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_D}$ such that:

$$\text{ten}(\boldsymbol{w}, I_1, I_2, \ldots, I_D)_{i_1 i_2 \ldots i_D} = w_i.$$

2.1 Tensorized Kernel Machines

The main idea of tensor networks (TNs) (Kolda and Bader, 2009; Cichocki, 2014; Cichocki et al., 2016) is to represent a D-th order tensor $\boldsymbol{\mathcal{W}}$ as a multilinear function of small-order tensors - cores. In this work, we use the Canonical Polyadic Decomposition.

Definition 2.3 (Canonical Polyadic Decomposition (Kolda and Bader, 2009))**.** A D-th order tensor $\boldsymbol{\mathcal{W}} \in \mathbb{C}^{I_1 \times I_2 \times \cdots \times I_D}$ has a rank-R CPD if

$$\text{vec}(\boldsymbol{\mathcal{W}}) = \sum_{r=1}^{R} \boldsymbol{w}_r^{(D)} \otimes \cdots \otimes \boldsymbol{w}_r^{(1)}. \tag{3}$$

The cores of this tensor network are the matrices $\boldsymbol{W}^{(d)} \in \mathbb{C}^{I_d \times R}$, where $\boldsymbol{w}_r^{(d)}$ represents the r-th column of the matrix $\boldsymbol{W}^{(d)}$. In other words, the vectorized version of a tensor $\boldsymbol{\mathcal{W}}$ can be represented as a sum of Kronecker product of vectors. Since there are D matrices, the storage complexity of a D-th order tensor is reduced from I^D down to DIR by using the CPD. The main advantage of the CPD, in the context of this paper, is that it has only one hyperparameter, R, whereas the Tensor Train (TT) (Oseledets, 2011) decomposition has $D-1$ rank hyperparameters. Moreover, the CPD does not exhibit the exponential dependence on D that is present in the Tucker Decomposition (Kolda and Bader, 2009).

CPD kernel machines exploit the tensor-product structure of the features (2) by restricting the model parameters $\boldsymbol{w}$ to be a tensor in the CPD format. This reduces the exponential complexity in the data dimensionality D to a linear complexity.

Theorem 2.1 (CPD Kernel Machine (Wesel and Batselier, 2024))**.** Suppose $\text{ten}(\boldsymbol{w}, I_1, I_2, \ldots, I_D)$ is represented by a CPD. The model responses and gradients of

$$f(\boldsymbol{x}) = \left[\boldsymbol{\psi}_\theta^{(D)}(x_D) \otimes \cdots \otimes \boldsymbol{\psi}_\theta^{(1)}(x_1)\right]^\top \boldsymbol{w}$$

can be computed in $\mathcal{O}(DIR)$ instead of $\mathcal{O}(\prod_{d=1}^{D} I_d)$, where $I = \max(I_1, \ldots, I_D)$.

Note that the feature map of the CPD kernel machine is a rank-1 CPD. The approach of tensorizing $\boldsymbol{w}$ and approximating the resulting tensor as a low-rank decomposition has been explored and introduced in several papers on product features and tensor networks (Wahls et al., 2014; Stoudenmire and Schwab, 2016; Batselier et al., 2017; Novikov et al., 2016; Cheng et al., 2021; Wesel and Batselier, 2021, 2024). Training a CPD kernel machine means solving the following non-linear non-convex optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|_2^2 + \frac{\alpha}{2}\|\boldsymbol{w}\|_2^2 \\ &\text{s.t. } \boldsymbol{w} = \sum_{r=1}^{R} \boldsymbol{w}_r^{(D)} \otimes \cdots \otimes \boldsymbol{w}_r^{(1)}, \end{aligned} \tag{4}$$

where $\boldsymbol{y} = [y_n]_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{C}^D$ - n-th row of data matrix $\boldsymbol{X} \in \mathbb{C}^{N \times D}$, $\boldsymbol{\Phi} = \left[\boldsymbol{\phi}_\theta(\boldsymbol{x}_n)^\top\right]_{n=1}^N \in \mathbb{C}^{N \times I_1 I_2 \ldots I_D}$, $\boldsymbol{w}_r^{(d)}$ is the r-th column of the d-th CPD core $\boldsymbol{W}^{(d)}$, and α is a regularization hyperparameter. Note that, in order to compute the model response $\boldsymbol{f} = \boldsymbol{\Phi}\boldsymbol{w}$, we implicitly consider its real part as we generally work with complex features $\boldsymbol{\phi}_\theta$ and model parameters $\boldsymbol{w}$.

Several algorithms have been proposed to solve the optimization problem (4) efficiently by exploiting the CPD structure such as Alternating Least Squares (ALS) (Kolda and Bader, 2009), Riemannian optimization (Novikov et al., 2016), and first-order gradient-based optimization algorithms (Kingma and Ba, 2014).

2.2 Fourier Features

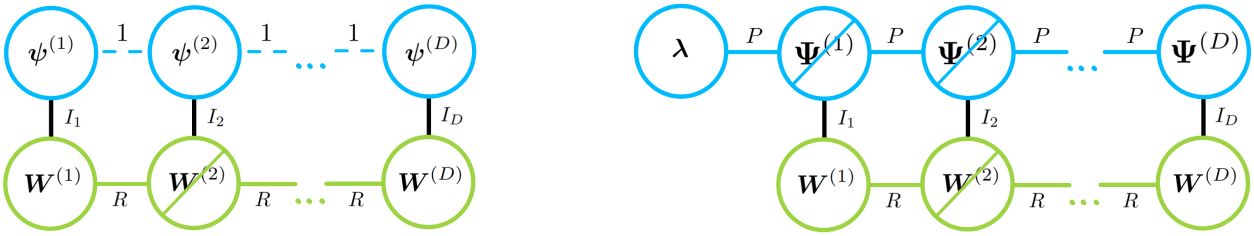
Different possibilities have been explored for constructing the feature mapping $\boldsymbol{\phi}_\theta$, for example polynomial functions (Blondel et al., 2017), pure-power polynomials of higher degree (Chen et al., 2016) and Fourier features (Wahls et al., 2014; Cheng et al., 2021). Since polynomial features can exhibit instabilities during training due to the construction of ill-conditioned Vandermonde matrices, we will focus on Fourier features moving forward. However, it is important to note that other generic functions can be used for high-dimensional data mapping in the context of this work.

Definition 2.4 (Fourier Features (Wesel and Batselier, 2024))**.** For an input sample $\boldsymbol{x} \in \mathbb{C}^D$, the Fourier feature map $\boldsymbol{\phi}_\theta : \mathbb{C}^D \to \mathbb{C}^{I_1 I_2 \ldots I_D}$ with I_d basis frequencies $\frac{-I_d}{2}, \ldots, \frac{I_d}{2} - 1$ per dimension is defined as:

$$\begin{aligned} \boldsymbol{\phi}_\theta(\boldsymbol{x}) &= \bigotimes_{d=1}^{D} c_d \, \boldsymbol{\psi}_\theta^{(d)}(x_d), \\ \boldsymbol{\psi}_\theta^{(d)}(x_d) &= \left[e^{-\frac{2\pi j x_d k}{\theta}}\right]_{k=0}^{I_d - 1}, \end{aligned}$$

where j is the imaginary unit, $c_d = e^{2\pi j x_d \frac{2+I_d}{2\theta}}$ and θ is the periodicity hyperparameter of the function class.

Applications of this mapping can be found in the field of kernel machines, as they can be viewed as



(a) Tensorized CPD Kernel Machine

(b) Feature Learning Model

Figure 1: Tensorized CPD Kernel Machine (Figure 1a) and FL Model (Figure 1b). In the diagrams, each circle without a cross represents a vector or a matrix (defined by the number of outgoing solid lines, one or two respectively); a crossed circle depicts a three-dimensional tensor containing a particular matrix in the diagonal slice; blue color represents parameters related to non-linear features $\boldsymbol{\psi}_{\theta}^{(d)}(x_d)$, $d = \overline{1, D}$; green color represents model parameters $\boldsymbol{w}$ in a CPD format; a solid line denotes a summation along the corresponding index, while a dashed line denotes a Kronecker product (Cichocki et al., 2016); $\boldsymbol{\Psi}^{(d)} = [\boldsymbol{\psi}_{\theta_1}^{(d)}(x_d), \ldots, \boldsymbol{\psi}_{\theta_P}^{(d)}(x_d)] \in \mathbb{C}^{I_d \times P}$. Figure 1a depicts model (1) with a rank-1 CPD feature map, while Figure 1b represents our FL model (5) with a rank-P feature map.

eigenfunctions of a D-dimensional stationary product kernel (Rasmussen and Williams, 2005; Solin and Särkkä, 2019). Generally, the corresponding optimal value of the hyperparamter θ can be identified through cross-validation across different options $\theta_1, \ldots, \theta_P$. To achieve this, one would need to solve the optimization problem in (4) at least P times.

In the next section, we introduce the main contribution of this paper. Our new Feature Learning model is based on the CPD kernel machine concept but, in contrast, learns a linear combination of feature maps from the data. This model allows the use of any non-linear features with a Kronecker product structure and solves an optimization problem analogous to (4) only once.

3 TENSOR NETWORK BASED FEATURE LEARNING MODEL

Previous works (Novikov et al., 2016; Wesel and Batselier, 2021, 2024) introduced similar formulations of the optimization problem (4), with an emphasis on finding optimal model parameters - $\boldsymbol{w}$. However, the search for model hyperparameters, θ, has received little attention. A common approach is to apply cross-validation over a set of hyperparameter options, $\theta_1, \theta_2, \ldots, \theta_P$. While effective for a broad range of problems, this generic technique does not leverage problem-specific characteristics that could enhance model properties, such as reducing computational cost and memory consumption or improving prediction quality. In the following subsections, we propose an alternative approach for optimizing model parameters related to the feature map $\boldsymbol{\phi}_{\theta}$ in the context of tensorized kernel machines.

3.1 Feature Learning Model

Instead of using the model (1) with a feature map $\boldsymbol{\phi}_{\theta}$ depending on hyperparameter θ, we consider the following model formulation:

$$f(\boldsymbol{x}) = \left[\sum_{p=1}^{P} \lambda_p \, \boldsymbol{\phi}_{\theta_p}(\boldsymbol{x}) \right]^{\top} \boldsymbol{w}, \tag{5}$$

with $\boldsymbol{\phi}_{\theta_p}(\boldsymbol{x}) = \boldsymbol{\psi}_{\theta_p}^{(D)}(x_D) \otimes \cdots \otimes \boldsymbol{\psi}_{\theta_p}^{(1)}(x_1)$ and learnable parameters λ_p that represent the scaling of features in the sum. The new feature mapping in the brackets can now be interpreted as a learnable rank-P CPD as it is the sum of P tensor-product features (3). The model reformulation allows us to replace the hyperparameters search problem with a regular optimization problem. In this case, we have to find two distinct sets of parameters: model parameters $\boldsymbol{w}$ and feature parameters $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_P]^{\top}$. Figure 1 demonstrates the difference between the original model (1) and the FL model (5) in a tensor network diagram. For both models the lower green network represents a rank-R CPD of ten $(\boldsymbol{w}, I_1, I_2, \ldots, I_D)$. The main difference is in the upper blue network. In the Figure 1a, the blue network represents the Kronecker product of vectors $\boldsymbol{\psi}_{\theta}^{(1)}, \ldots, \boldsymbol{\psi}_{\theta}^{(D)}$, whereas the blue network in the Figure 1b depicts the rank-P CPD of the feature map.

Under the supervised regression problem framework, both $\boldsymbol{w}$ and $\boldsymbol{\lambda}$ are found by optimizing the following non-convex non-linear optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{\lambda},\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{y} - \sum_{p=1}^{P} \lambda_p \boldsymbol{\Phi}_p \boldsymbol{w}\|_2^2 + \frac{\alpha}{2}\|\boldsymbol{w}\|_2^2 + \beta\,\mathrm{Reg}(\boldsymbol{\lambda}) \\ &\text{s.t. } \boldsymbol{w} = \sum_{r=1}^{R} \boldsymbol{w}_r^{(D)} \otimes \cdots \otimes \boldsymbol{w}_r^{(1)}, \end{aligned} \tag{6}$$

where $\boldsymbol{\Phi}_p = \left[\boldsymbol{\phi}_{\theta_p}(\boldsymbol{x}_n)^\top\right]_{n=1}^{N} \in \mathbb{C}^{N \times I_1 I_2 \ldots I_D}$, $\boldsymbol{w}_r^{(d)}$ is the r-th column of the d-th CPD core $\boldsymbol{W}^{(d)}$. In contrast to the optimization problem (4) there is now an additional regularization term $\mathrm{Reg}(\boldsymbol{\lambda})$ for $\boldsymbol{\lambda}$ together with its own β regularization hyperparameter.

In the next subsections we derive the ALS solution to the problem (6) separating the distinct parameters $\boldsymbol{\lambda}, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(D)}$ such that each optimization subproblem becomes linear. The local convergence of the ALS optimization using CPD has been investigated in (Uschmajew, 2012).

3.2 ALS Update Of Model Parameters $\boldsymbol{W}^{(d)}$

To find the optimal model parameters $\boldsymbol{w}$ represented as a CPD tensor we fix the values of $\boldsymbol{\lambda}$. Secondly, as the CPD is a multi-linear function of its cores (Cichocki et al., 2016) we fix the values of all the cores, but one - $\boldsymbol{W}^{(d)}$. This results in the following Regularized Least Squares problem:

$$\begin{aligned} &\min_{\boldsymbol{v}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}_d \boldsymbol{v}\|_2^2 + \frac{\alpha}{2}\boldsymbol{v}^\top(\boldsymbol{H}_d \otimes \boldsymbol{I})\boldsymbol{v} + \text{const} \\ &\text{s.t. } \boldsymbol{v} = \mathrm{vec}(\boldsymbol{W}^{(d)}), \end{aligned} \tag{7}$$

where:

$$\begin{aligned} \boldsymbol{A}_d &= \sum_{p=1}^{P} \lambda_p \left(\boldsymbol{Z}^{(d,p)} \odot_R \boldsymbol{\Psi}^{(d,p)}\right) \in \mathbb{C}^{N \times I_d R}, \\ \boldsymbol{H}_d &= \circledast_{\substack{j=1 \\ j \neq d}}^{D} \boldsymbol{W}^{(j)\top} \boldsymbol{W}^{(j)} \in \mathbb{C}^{R \times R}, \\ \boldsymbol{Z}^{(d,p)} &= \circledast_{\substack{j=1 \\ j \neq d}}^{D} \boldsymbol{\Psi}^{(j,p)} \boldsymbol{W}^{(j)} \in \mathbb{C}^{N \times R}, \\ \boldsymbol{\Psi}^{(d,p)} &= \left[\boldsymbol{\psi}_{\theta_p}^{(d)}(x_{nd})^\top\right]_{n=1}^{N} \in \mathbb{C}^{N \times I_d}. \end{aligned} \tag{8}$$

In order to update the full vector of model parameters $\boldsymbol{w}$, one sequentially updates the $\boldsymbol{W}^{(d)}$, $d = \overline{1, D}$ cores of the CPD using the following expression:

$$\mathrm{vec}(\boldsymbol{W}^{(d)}) = (\boldsymbol{A}_d^\top \boldsymbol{A}_d + \alpha(\boldsymbol{H}_d \otimes \boldsymbol{I}))^{-1} \boldsymbol{A}_d^\top \boldsymbol{y}. \tag{9}$$

The computational complexity of this update for a single core is $\mathcal{O}\left(N[I_d R P + I_d^2 R^2]\right)$ when $N \gg I_d R$, and the memory complexity is $\mathcal{O}(N I_d R)$. Here, N is the training sample size, I_d is the dimensionality of the mapping $\boldsymbol{\psi}_{\theta_p}^{(d)}$, R is the CPD rank, P is the number of unique feature hyperparameter configurations θ. Once all D core matrices in the CPD representation of $\boldsymbol{w}$ have been updated, the model parameters $\boldsymbol{w}$ are fixed, and the next step is to update the feature parameters $\boldsymbol{\lambda}$.

3.3 ALS Update Of Feature Parameters $\boldsymbol{\lambda}$

The FL model objective (6) can be expressed as a function of $\boldsymbol{\lambda}$ if the vector $\boldsymbol{w}$ vector is assumed to be fixed, resulting in a Regularized Least Squares problem:

$$\min_{\boldsymbol{\lambda}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\lambda}\|_2^2 + \beta\,\mathrm{Reg}(\boldsymbol{\lambda}) + \text{const}, \tag{10}$$

where $\boldsymbol{F} = [\boldsymbol{\Phi}_1 \boldsymbol{w} \ldots \boldsymbol{\Phi}_P \boldsymbol{w}] \in \mathbb{R}^{N \times P}$. The optimal solution depends on the type of regularization and parameter constraints. In this work, we consider 3 types of regularization: L1 (Beck and Teboulle, 2009), L2 (Hoerl and Kennard, 2000), and Fixed Norm (Golub and Van Loan, 2013).

L1 Regularization. The solution of this convex, yet not differentiable problem can be found in iterative way using proximal gradient based algorithm (Beck and Teboulle, 2009):

$$\begin{aligned} &\boldsymbol{p}_s = \boldsymbol{\lambda}_s - t_s \boldsymbol{F}^\top(\boldsymbol{F}\boldsymbol{\lambda}_s - \boldsymbol{y}), \\ &\boldsymbol{\lambda}_{s+1} = \textsc{sign}(\boldsymbol{p}_s) \circledast \textsc{max}(0, \boldsymbol{p}_s - \beta t_s \boldsymbol{1}), \\ &t_s \in \left(0, \frac{1}{\left\|\boldsymbol{F}^\top \boldsymbol{F}\right\|_2}\right), s = \overline{1, S}, \end{aligned} \tag{11}$$

where $\boldsymbol{\lambda}_s$ represents the estimate after s gradient step iterations, t_s is the gradient step size, SIGN and MAX are operators of element-wise sign and max functions respectively, $\boldsymbol{1}$ is a vector of all ones. The computational complexity of this algorithm is $\mathcal{O}(NP^2)$, assuming $N \gg S$.

Due to the nature of the L1 regularization constraint, some coefficients λ_p become exactly zero, leading to sparse models. This sparsity can be utilized to reduce the computational complexity of the $\boldsymbol{W}^{(d)}$ update by avoiding the inclusion of zero contributions when constructing the $\boldsymbol{A}_d$ matrix in Equation (8).

L2 Regularization. Similar to the $\boldsymbol{W}^{(d)}$ update, the solution of the Least Squares problem for $\boldsymbol{\lambda}$ (Equation (10)) with L2 regularization is:

$$\boldsymbol{\lambda} = (\boldsymbol{F}^\top \boldsymbol{F} + \beta \boldsymbol{I})^{-1} \boldsymbol{F}^\top \boldsymbol{y} \tag{12}$$

with a computational complexity of $\mathcal{O}(NP^2)$.

Algorithm 1: FL Model ALS Algorithm

Parameters: $\alpha, \beta, R, \mathcal{E}, [\boldsymbol{\psi}_{\theta_p}^{(d)}]_{d=1,p=1}^{D,P}$
Data: $\boldsymbol{X} \in \mathbb{C}^{N\times D}, \boldsymbol{y} \in \mathbb{R}^N$
1 Initialize $\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(D)}, \boldsymbol{\lambda}$
2 Precompute $\boldsymbol{Z}_p = \circledast_{j=1}^{D} \boldsymbol{\Psi}^{(j,p)} \boldsymbol{W}^{(j)}, p = \overline{1, P}$
3 Precompute $\boldsymbol{H} = \circledast_{j=1}^{D} \boldsymbol{W}^{(j)\top} \boldsymbol{W}^{(j)}$
4 **while** $e \leqslant \mathcal{E}$ **do**
5 **for** $k = \overline{1, D}$ **do**
6 $\boldsymbol{Z}^{(k,p)} = \boldsymbol{Z}_p \oslash [\boldsymbol{\Psi}^{(k,p)} \boldsymbol{W}^{(k)}], p = \overline{1, P}$
7 $\boldsymbol{H}_k = \boldsymbol{H} \oslash [\boldsymbol{W}^{(k)\top} \boldsymbol{W}^{(k)}]$
8 Update $\boldsymbol{A}_k$ from (8) based on $\boldsymbol{Z}^{(k,p)}$
9 Update $\boldsymbol{W}^{(k)}$ based on (9)
10 $\boldsymbol{Z}_p = \boldsymbol{Z}^{(k,p)} \circledast [\boldsymbol{\Psi}^{(k,p)} \boldsymbol{W}^{(k)}], p = \overline{1, P}$
11 $\boldsymbol{H} = \boldsymbol{H}_k \circledast [\boldsymbol{W}^{(k)\top} \boldsymbol{W}^{(k)}]$
12 Update $\boldsymbol{\lambda}$ based on (11) or (12) or (14)
13 $e = e + 1$
Output: $\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(D)}, \boldsymbol{\lambda}$

Fixed Norm Regularization. The introduction of an additional β hyperparameter can be seen as a drawback, as it requires extra tuning. One way to eliminate this hyperparameter from the optimization problem is by considering the following constrained optimization problem:

$$\min_{\|\boldsymbol{\lambda}\|_2^2 \leqslant 1} \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\lambda}\|_2^2. \tag{13}$$

By constraining the norm of $\boldsymbol{\lambda}$, one can effectively regularize the optimization problem. According to (Golub and Van Loan, 2013) the optimal solution of problem (13) is:

$$\boldsymbol{\lambda} = \boldsymbol{V} \left(\boldsymbol{\Sigma}^\top \boldsymbol{\Sigma} + \mu \boldsymbol{I} \right)^{-1} \boldsymbol{\Sigma}^\top \boldsymbol{c}, \tag{14}$$

where $\boldsymbol{F} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ is the Singular Value Decomposition (SVD) of the matrix $\boldsymbol{F}$, $\boldsymbol{c} = \boldsymbol{U}^\top \boldsymbol{y}$, and μ is the solution of the non-linear equation:

$$\sum_{i=1}^{r} \frac{c_i^2 \sigma_i^2}{(\sigma_i^2 + \mu)} = 1.$$

As the SVD is the dominant computational part of this algorithm, its complexity is $\mathcal{O}(NP^2)$.

3.4 FL Model Implementation Details

As we mentioned earlier, the optimization problem (6) is effectively solved using an Alternating Least Squares approach for $\boldsymbol{\lambda}$ and $\boldsymbol{w}$. The parameters are updated over multiple epochs or until convergence, based on a predefined metric or loss function. Implementation details are provided in Algorithm 1.

In line 1 of Algorithm 1, the FL model parameters are randomly initialized. The initial CPD cores $\boldsymbol{W}^{(d)}, d = \overline{1, D}$ are sampled from a standard normal distribution and then normalized column-wise. The initial feature parameters, $\boldsymbol{\lambda}$, are drawn from a uniform distribution from 0 to 1. After the initialization step, computational costs can be reduced by precomputing the matrices from lines 2-3 in Algorithm 1. This precomputation has to be done once, before the training loop starts. The remaining lines in Algorithm 1 are the repeated updates of every core $\boldsymbol{W}^{(d)}, d = \overline{1, D}$ and feature parameters $\boldsymbol{\lambda}$. The overall computational complexity of the FL model training is $\mathcal{O}(\mathcal{E}DNIR[P + IR])$ and peak memory complexity is $\mathcal{O}(NR[P + I])$, computed based on Equations (9), (11), (12), (14) and Algorithm 1.

It is worth mentioning that Fourier features (Definition 2.4) can be *quantized*, meaning further tensorized (Wesel and Batselier, 2024). These quantized features allow for the quantization of the model weights $\boldsymbol{w}$ yielding more expressive models for the same number of model parameters. For example, suppose the dimensionality I_d of $\boldsymbol{\psi}_\theta^{(d)}$ in Equation (2) is $I_d = 2^{K_d}$. Then each $\boldsymbol{\psi}_\theta^{(d)}$ itself can be written as a Kronecker product

$$\boldsymbol{\psi}_\theta^{(d)} = \boldsymbol{\gamma}_\theta^{(K_d)} \otimes \cdots \otimes \boldsymbol{\gamma}_\theta^{(1)},$$

where $\boldsymbol{\gamma}_\theta^{(k)} \in \mathbb{C}^2, k = \overline{1, K_d}$. In this case, CPD kernel machine consists of $\sum_{d=1}^{D} K_d$ cores instead of D, with each core $\boldsymbol{W}^{(d)}$ of size $2 \times R$ instead of $I_d \times R$.

As the difference between the two tensorization methods is negligible in terms of notation and design, our implementation uses the quantized version as it shows better generalization capabilities (Wesel and Batselier, 2024). In the quantized setting the FL model computational complexity is $\mathcal{O}(\mathcal{E}DNKR[P + R])$ and the corresponding peak memory complexity is $\mathcal{O}(NRP)$ with $K = \log_2(I)$, $I = \max(I_1, \ldots, I_D)$. As a result, the quantized algorithm requires less memory for intermediate calculations and less time to update the model parameters.

4 NUMERICAL EXPERIMENTS

In all experiments the input $\boldsymbol{x}$ is scaled to lie in the unit hypercube and the output vector $\boldsymbol{y}$ is standardized. We consider quantized version of Fourier features (Definition 2.4) for all the experiments. The model parameters $\boldsymbol{w}$ are represented as a quantized CPD of rank R. For the cross-validation baseline (CV), we use a CPD kernel machine (Theorem 2.1, Wesel and Batselier (2024)) with quantization and apply 6-fold cross-validation to the θ hyperparameter of the Fourier

Table 1: Impact of regularization on the FL model performance across various datasets. MSE and Time metric values were averaged over 10 restarts. Reg. column denotes different regularization types: L1 represents $\|\boldsymbol{\lambda}\|_1$, L2 denotes $\|\boldsymbol{\lambda}\|_2$, FN enforces the constraint $\|\boldsymbol{\lambda}\|_2 \leqslant 1$. Option (P) introduces an additional constraint to the problem, requiring the solution to be non-negative.

	MSE ↓					Time (sec.) ↓				
Reg.	Airfoil	Concrete	Energy	Wine	Yacht	Airfoil	Concrete	Energy	Wine	Yacht
FN	0.19	0.154	0.003	0.68	0.11	4.9	1.3	0.7	26.9	0.1
FN(P)	0.187	0.14	0.009	0.932	0.366	4.1	1.3	0.7	23.9	0.1
L1	0.184	0.139	0.003	0.692	0.112	2.7	1.4	1.0	24.9	0.1
L1(P)	0.182	0.176	0.003	0.705	0.115	3.4	1.3	1.5	23.7	0.1
L2	0.189	0.15	0.003	0.776	0.1	3.1	1.5	0.7	27.8	0.1
L2(P)	0.188	0.146	0.007	0.672	0.327	3.4	1.4	0.7	26.8	0.1

features, with $\theta \in \{10, 2, 128, 25, 64, 600, 2000, 1024\}$. For the experiments, we select 5 UCI regression datasets (Dua and Graff, 2017). For each dataset, 80% of the data is randomly chosen for training, with the remaining 20% reserved for testing. We set the same number of basis functions, $I_d = I$, uniformly for all $d = \overline{1, D}$ for simplicity. The CPD rank R and hyperparameter I are chosen such that the number of FL model parameters is less than the sample size N, ensuring an underparameterized training regime to prevent the data memorization. Algorithm 1 is performed for 10 full updates (epochs) of $\boldsymbol{w}$ and $\boldsymbol{\lambda}$. The training procedure is repeated 10 times, with the dataset split into training and test sets as described above. Finally, we compute the mean and standard deviation of the mean squared error (MSE) on a test set and check training time (in seconds). All experiments, except for the large-scale one, were conducted on a Dell Inc. Latitude 7440 laptop equipped with a 13th Gen Intel Core i7-1365U CPU and 16 GB of RAM.

The source code for all the functions and classes, written in Python, and data to reproduce all experiments is available on GitHub.[1]

4.1 Regularization Alternatives

The primary goal of the first set of experiments was to examine how different regularization terms, $\text{Reg}(\boldsymbol{\lambda})$, in Equation (6) affect the predictive quality of the regression solution. We consider 3 types of regularization: L1 - $\|\boldsymbol{\lambda}\|_1$, L2 - $\|\boldsymbol{\lambda}\|_2$, Fixed Norm is $\|\boldsymbol{\lambda}\|_2 \leqslant 1$. Apart from that, we also consider an additional non-negative constraint on the final solution - $\lambda_p \geqslant 0$. As a result, there are 6 cases to examine.

Table 1 demonstrates that different regularization functions yield comparable results in terms of prediction quality (test MSE) and training time across all datasets. However, the FL model combined with L1 and L1(P) settings achieved slightly better MSE values and trained faster on average across different datasets. An intuitive explanation for this result is that the computational complexity of the L1 solution is reduced due to sparsity. Specifically, a fraction of the feature parameters $\lambda_p = 0$, which results in fewer computations to update matrices $\boldsymbol{W}^{(d)}$, $d = \overline{1, D}$ in Equation (7).

To summarize the results of Table 1, differently regularized problems yield close solutions in terms of the MSE metric and training time. Therefore one should look for other criteria to distinguish the different regularization methods. For example, L1 regularization generates a sparse solution and the more $\lambda_p = 0$, the faster the ALS method would work for the whole FL model due to reduced computations. However, one might need to do cross-validation with respect to β - regularization hyperparameter. On the other hand, fixed norm regularization allows to avoid extra hyperparameter tuning, but the solution would be dense in terms of $\boldsymbol{\lambda}$ in this case.

4.2 Comparison With Cross-Validation

In this section we compare the FL model training abilities to the corresponding quantized CPD kernel machine (Wesel and Batselier, 2021, 2024) with cross-validation (CV) in the sequential learning setting (parallelization is turned off for fair comparison). In order to do that, we train these algorithms on several small- and large-scale datasets and showcase the test MSE and training time.

4.2.1 Small-Scale Problems

Figure 2 shows how training time and test MSE are affected by the number of features P (i.e., different number of hyperparameter options). It can be observed

[1] https://github.com/AlbMLpy/TN-FL-Model

Table 2: Test MSE results and training time of the FL model and corresponding Cross-Validated model. For all the datasets $P = 8$ and the results were calculated using 10 restarts. For Airline data, $P = 6$ and the results were calculated based on 5 restarts. MSE (FL)/(CV) - the average test MSE value within 1 std; Time (FL)/(CV) - the average training time (in seconds); N - sample size; D - data dimensionality; I - the number of basis functions per dimension; R - the CPD rank.

Data	N	D	I	R	MSE (FL) ↓	MSE (CV) ↓	Time (FL) ↓	Time (CV) ↓
Airfoil	1502	5	4	51	0.184 ± 0.02	0.223 ± 0.02	3.0	23
Energy	768	8	4	15	0.003 ± 0.0	0.003 ± 0.0	0.91	5.5
Yacht	308	6	2	6	0.112 ± 0.02	0.358 ± 0.06	0.149	0.615
Concrete	1030	8	8	10	0.139 ± 0.03	0.118 ± 0.02	1.2	8.8
Wine	6497	11	16	25	0.692 ± 0.07	0.652 ± 0.04	33	152
Airline	5929413	8	64	20	0.804 ± 0.0	0.779 ± 0.0	15159	56590

that the FL model exhibits a less steep learning curve in terms of training time for all datasets. This is due to the training algorithm structure and the choice of L1 regularization that encourages sparser representation of features, thereby reducing the computational load between epochs. The total computational complexity of the FL model: $\mathcal{O}(\mathcal{E}DNIR[P+IR])$ is better than that of CV: $\mathcal{O}(\mathcal{E}DNIR[PIR])$. Speaking of the prediction error, we can conclude that the FL model shows performance similar to CV in most datasets (mean values are close and standard deviation regions intersect). If we consider Yacht dataset the quality of the FL model is superior to CV in that case. A possible explanation for this result is that the sum of features (FL) appears to be more expressive than one particular feature (CV).

In summary, according to Table 2 and Figure 2, the FL model can be trained 3-5 times faster without a significant drop in prediction quality compared to cross-validation.

4.2.2 Large-Scale Problem

In order to show and explore the behavior of the FL model on large scale data, we consider the airline dataset (Hensman et al., 2013), an 8-dimensional dataset which consists of $N = 5929413$ recordings of commercial airplane flight delays that occurred in 2008 in the USA. Specifically for this dataset (Samo and Roberts, 2015) we consider a uniform random draw of $2/3N$ for training and keep the remainder for the evaluation of MSE on the test set and repeat the procedure 5 times. We use Fourier features with local dimension $I_d = 64$. For this experiment, we set $\alpha = 0.01$, $\beta = 0.1$, $R = 20$, and run the ALS optimizer for 10 epochs. The different options for θ are $\{10, 2, 128, 25, 64, 1024\}$. This experiment was conducted on the 2 * AMD EPYC 7252 8-core CPU, 256 GB RAM memory.

As seen from the last row of Table 2, training the FL model took four times less time, with a non-significant difference in prediction quality (0.80 compared to 0.78). The results of this experiment further confirm that the unique structure of the FL model enhances its efficiency.

5 CONCLUSION

We introduced a new Feature Learning (FL) model that uses the CPD structure for features representation and model parameters $\boldsymbol{w}$. We verified experimentally the effectiveness of the FL model on real small and large scale datasets, and demonstrated that it can be trained consistently faster than the cross-validated CPD kernel machine. For future work, we plan to develop a parallel FL model utilizing the summation structure of the model. Another promising direction is to explore a probabilistic formulation of the FL model, enabling uncertainty estimation for classification and regression tasks.

Acknowledgements

We would like to thank the anonymous reviewers for their numerous suggestions and improvements which have greatly improved the quality of this paper. This publication is part of the project Sustainable learning for Artificial Intelligence from noisy large-scale data (with project number VI.Vidi.213.017) which is financed by the Dutch Research Council (NWO).

References

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

Beck, A. and Teboulle, M. (2009). A fast itera-

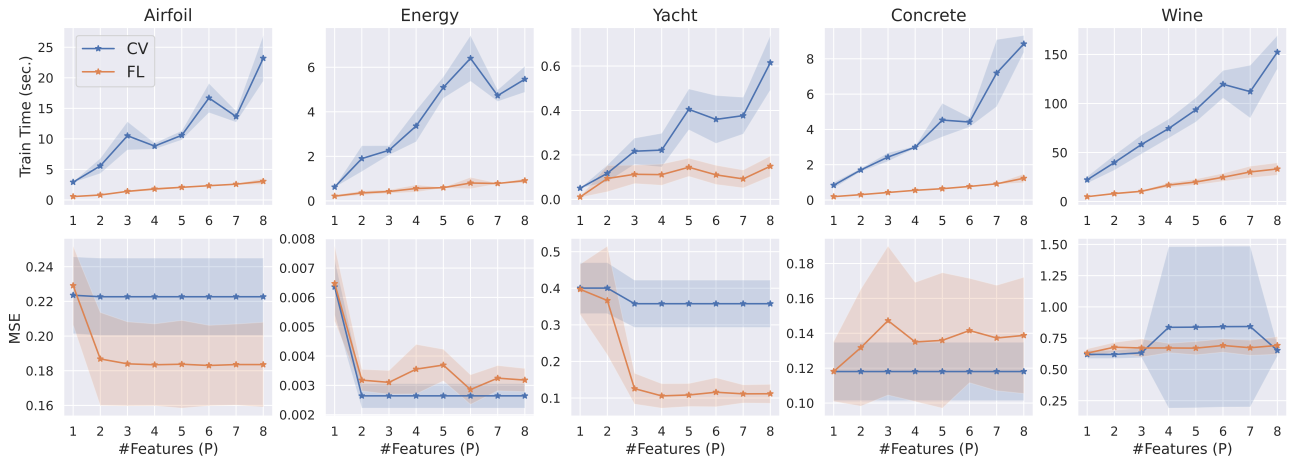


Figure 2: Plots of the training time (first row) and test MSE (second row) of FL and CV models (orange and blue curves respectively) as a function of the number of features P for different real-life datasets (column-wise). Solid lines represent mean metric calculations and shaded regions depict ± 1 standard deviation around the mean across 10 restarts. The proposed FL model requires consistently less time to train compared to the conventional cross-validation. Likewise the prediction error of the FL model is either similar to CV (shaded regions intersect) or significantly lower (Yacht data) that demonstrates the superiority of the FL model.

tive shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202.

Blondel, M., Niculae, V., Otsuka, T., and Ueda, N. (2017). Multi-output polynomial networks and factorization machines. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Chen, Z., Batselier, K., Suykens, J. A. K., and Wong, N. (2016). Parallelized tensor train learning of polynomial classifiers. *IEEE Transactions on Neural Networks and Learning Systems*, 29:4621–4632.

Cheng, S., Wang, L., and Zhang, P. (2021). Supervised learning with projected entangled pair states. *Physical Review B*, 103(12).

Cichocki, A. (2014). Era of big data processing: A new approach via tensor networks and tensor decompositions. *ArXiv*, abs/1403.2048.

Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., and Mandic, D. P. (2016). 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.

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

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Ghojogh, B., Ghodsi, A., Karray, F., and Crowley, M. (2021). Reproducing kernel Hilbert space, Mercer's theorem, eigenfunctions, Nyström method, and use of kernels in machine learning: Tutorial and survey. *ArXiv*, abs/2106.08443.

Golub, G. H. and Van Loan, C. F. (2013). *Matrix Computations - 4th Edition*. Johns Hopkins University Press, Philadelphia, PA.

Hammer, B. and Gersmann, K. (2002). A note on the universal approximation capability of support vector machines. *Neural Processing Letters*, 17.

Hensman, J., Durrande, N., and Solin, A. (2017). Variational Fourier features for Gaussian processes. *The Journal of Machine Learning Research*, 18(1):5537–5588.

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

Hoerl, A. E. and Kennard, R. W. (2000). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 42(1):80–86.

Kargas, N. and Sidiropoulos, N. D. (2021). Supervised learning and canonical decomposition of multivari-

ate functions. *IEEE Transactions on Signal Processing*, 69:1097–1107.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

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

Lee, J., Sohl-dickstein, J., Pennington, J., Novak, R., Schoenholz, S., and Bahri, Y. (2018). Deep neural networks as Gaussian processes. In *International Conference on Learning Representations*.

Novak, R., Xiao, L., Bahri, Y., Lee, J., Yang, G., Abolafia, D. A., Pennington, J., and Sohl-dickstein, J. (2019). Bayesian deep convolutional networks with many channels are Gaussian processes. In *International Conference on Learning Representations*.

Novikov, A., Trofimov, M., and Oseledets, I. (2016). Exponential machines. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, 66.

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

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Refaeilzadeh, P., Tang, L., and Liu, H. (2009). *Cross-Validation*, pages 532–538. Springer US, Boston, MA.

Rendle, S. (2010). Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000.

Samo, Y.-L. K. and Roberts, S. J. (2015). String and membrane Gaussian processes. *The Journal of Machine Learning Research*, 17:131:1–131:87.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Solin, A. and Särkkä, S. (2019). Hilbert space methods for reduced-rank Gaussian process regression. *Statistics and Computing*, 30(2):419–446.

Stoudenmire, E. and Schwab, D. J. (2016). Supervised learning with tensor networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Uschmajew, A. (2012). Local convergence of the alternating least squares algorithm for canonical tensor approximation. *SIAM Journal on Matrix Analysis and Applications*, 33(2):639–652.

Wahls, S., Koivunen, V., Poor, H. V., and Verhaegen, M. (2014). Learning multidimensional Fourier series with tensor trains. In *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 394–398.

Wesel, F. and Batselier, K. (2021). Large-scale learning with Fourier features and tensor decompositions. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 17543–17554. Curran Associates, Inc.

Wesel, F. and Batselier, K. (2024). Quantized Fourier and polynomial features for more expressive tensor network models. In Dasgupta, S., Mandt, S., and Li, Y., editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 1261–1269. PMLR.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes
   (b) Complete proofs of all theoretical results. Yes
   (c) Clear explanations of any assumptions. Yes

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes

(d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes
   (b) The license information of the assets, if applicable. Not Applicable
   (c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable
   (d) Information about consent from data providers/curators. Not Applicable
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable