Proceedings of Machine Learning Research 179:1–20, 2022 Conformal and Probabilistic Prediction and Applications

Tensor-Train Kernel Learning for Gaussian Processes

Max Kirstein MAX.KIRSTEIN@DE.BOSCH.COM
Bosch Center for Artificial Intelligence, Hildesheim, Germany
David Sommer SOMMER@WIAS-BERLIN.DE
Martin Eigel EIGEL@WIAS-BERLIN.DE
Weierstrass Institute for Applied Analysis and Stochastics, Berlin, Germany

Editor: Ulf Johansson, Henrik Boström, Khuong An Nguyen, Zhiyuan Luo and Lars Carlsson

Abstract

We propose a new kernel learning approach based on efficient low-rank tensor compression for Gaussian process (GP) regression. The central idea is to compose a low-rank function represented in a hierarchical tensor format with a GP covariance function. Compared to similar deep neural network architectures, this approach facilitates to learn significantly more expressive features at lower computational costs as illustrated in the examples. Additionally, over-fitting is avoided with this compositional model by taking advantage of its inherent regularisation properties. Estimates of the generalisation error are compared to five baseline models on three synthetic and six real-world data sets. The experimental results show that the incorporated tensor network enables a highly accurate GP regression with a comparatively low number of trainable parameters. The observed performance is clearly superior (usually by an order of magnitude in mean squared error) to all examined standard models, in particular to deep neural networks with more than 1000 times as many parameters.
Keywords: Tensor networks, Tensor-Train decomposition, Gaussian processes.

1. Introduction

This work introduces a kernel learning approach based on tensor network models for approximating high-dimensional functions. Specifically, we investigate the capabilities of tree tensor networks (Stoudenmire and Schwab, 2016; Oseledets, 2011) to learn expressive features for Gaussian process (GP) regression (Rasmussen and Williams, 2005) in order to enable uncertainty aware predictions. To achieve this, the tensor network parameters are treated as kernel hyper-parameters, which can be optimised through the marginal likelihood of the GP.

We also highlight tensor kernel learning as an approach for quantifying predictive uncertainty for tensor network models. Thereby, their applicability can be extended to high stakes domains like medical applications. For these domains, integration of prior knowledge into the modelling process and interpretability through rigorous mathematical foundations are important properties, which neural networks (arguably) do not provide (yet) although they exhibit similar (but theoretically larger) representational power.

A first approach for using tensor networks for kernel learning was proposed in (Konstantinidis et al., 2021). We add to that work in two important ways: first, we make use of a different type of low rank structure, namely Tensor-Trains (TTs) instead of the Canonical-Polyadic (CP) decomposition. Contrary to the latter, the TT format permits an efficient pre-training procedure so that the tensor network kernel starts with near optimal weights before end-to-end training. Moreover, the TT format allows pre-training and gradient-based optimisation to be easily regularised w.r.t. the norm of an underlying function space. This prevents over-fitting even in complex architectures and speeds up convergence. As a second addition, we perform thorough experiments for all compared architectures in this work and are able to show that our TT kernel method beats all baselines on the considered real world tasks usually by at least one order of magnitude. These baselines include the architecture proposed in (Konstantinidis et al., 2021) as well as a deep neural network with about a hundred times as many parameters as our method.

The rest of the work is structured as follows. In section 2 we briefly review the related literature. Section 3 then introduces the concept of low-rank feature extractors using TTs, which will be combined with GPs in section 4 to define the Tensor-Train kernel learning method. In section 5 we detail the experimental setup on several synthetic and real world data sets and report the results. Finally, section 6 provides a summary of the work and some concluding remarks.

2. Related Work

Tensor network models for supervised learning have become increasingly popular in the machine learning community for the last five years. Some notable contributions are (Stoudenmire and Schwab, 2016; Novikov et al., 2017; Efthymiou et al., 2019; Cohen and Shashua, 2016; Glasser et al., 2020; Blagoveschensky and Phan, 2020; Grelier et al., 2019; Ali and Nouy, 2020a,b, 2021). It should be noted that the physics and numerical mathematics communities have driven research in this direction for a much longer time, resulting, e.g. in standard textbooks like (Hackbusch, 2012), a range of training methods (Holtz et al., 2011; White, 1992; Grasedyck and Krämer, 2019) or application of tensor networks to (uncertainty quantification of) partial differential equations (Bachmayr et al., 2016; Eigel et al., 2019; Dolgov et al., 2015; Eigel et al., 2017, 2018), optimal transport (Eigel et al., 2020) and control (Oster et al., 2020).

Uncertainty quantification for tensor networks and tensor regression can e.g. be found in (Kirstein et al., 2020; Hu et al., 2020; Hawkins and Zhang, 2019). In contrast to our work however, the authors treat the whole model as Bayesian rather than only using a Bayesian layer at the end of a multi-layer model architecture.

Another interesting work is (Izmailov et al., 2018), where tensor networks are used to compress a tensor-variate mean function of a sparse Gaussian process, in order to radically increase the possible number of inducing points.

In what follows, we treat kernel learning as the composition of standard kernels (i.e. a Matérn or squared-exponential kernel) with functions from a different model class (i.e. neural or tensor networks). This approach is mainly developed in (Fu Jie Huang and LeCun, 2006; Salakhutdinov and Hinton, 2007) and further extended, e.g. in (Wilson et al., 2016a,b; Damianou and Lawrence, 2013; Zhuang et al., 2011). The authors compose different kernel functions or use deep neural networks in conjunction with kernels to learn rich high level features, which improve performance compared to classical support vector machines and Gaussian processes.

In (Ober et al., 2021) the question of over-fitting in kernel learning with deep neural networks is posed. The authors show empirically that an optimisation of the marginal likelihood does not provide enough regularisation for highly complex models with lots of parameters like deep neural networks. This is in contrast to the usual assumption that the marginal likelihood objective - as part of the Bayesian approach - should deliver an inherent complexity penalty sufficient to prevent over-fitting. At this point we conjecture that tensor networks are less prone to this kind of behaviour due to the inherent regularisation properties of a low-rank structure and the introduction of prior knowledge through the choice of basis functions for the tensor-product space.

Interestingly, (van Amersfoort et al., 2021) implicitly addresses the same problem as (Ober et al., 2021). As a remedy for over-fitting in their deep neural network model, the authors introduce further regularisation, which increases smoothness of the generated function approximation. This supports our claim that tensor networks can act as a stand-in for deep neural networks, especially when introduction of prior information is needed, since regularisation is much more intuitive and a direct feature of the model.

3. Low-Rank Tensor Networks

3.1. Basis Representations in High-Dimensional Spaces

In order to construct a function approximator, we start with a set of basis functions $\{P_{\alpha_i} : \mathbb{R} \to \mathbb{R}\}_{i=1}^{d}$, with $\alpha_i = 1, \ldots, J_i$ and $J_i \in \mathbb{N}$. Under the assumption of a tensor product function space, we can represent the approximator as

$$f_W(x) = \sum_{\alpha_1=1}^{J_1} \cdots \sum_{\alpha_d=1}^{J_d} W_{\alpha_1,\ldots,\alpha_d} \prod_{i=1}^{d} P_{\alpha_i}(x_i), \tag{1}$$

for $x = (x_1, \ldots, x_d) \in \mathcal{X}$, where W is a coefficient tensor of order d, i.e. $W \in \mathbb{R}^{J_1 \times \ldots \times J_d}$. While this means large expressive power, it poses a significant computational burden due to the exponential complexity growth in the dimensionality d ("curse of dimensionality"), which we alleviate by the introduction of a low-rank tensor format as described subsequently.

3.2. Tensor Decomposition

The exponential full tensor complexity (regarding storage and algorithms) fortunately can be reduced significantly by exploiting an often encountered low-rank structure of f_W (see, e.g. (Cichocki et al., 2016)). For our experiments we choose the so-called Tensor-Train format (Oseledets, 2011), which is a popular linearised case of more general tree tensor formats (Grelier et al., 2019). Recalling expansion (1), a low-rank TT compression of the coefficient tensor W reads

$$W_{\alpha_1,\ldots,\alpha_d} \approx \sum_{k_0=1}^{r_0} \cdots \sum_{k_d=1}^{r_d} \prod_{i=1}^{d} V^{(i)}_{k_{i-1},\alpha_i,k_i}, \tag{2}$$

with $r_0 = r_d = 1$. The components $V^{(i)} \in \mathbb{R}^{r_i \times J_i \times r_{i+1}}$, $i = 1, \ldots, d-1$, are tensors of order three. The remaining components $V^{(1)}, V^{(d)}$ are of order two. The upper summation bounds r_i, $i = 1, \ldots, d-1$, are called TT ranks.

The TT format exhibits a storage complexity of $\mathcal{O}(\max(J_1, \dots, J_d) d \max(r_1, \dots, r_{d-1})^2)$, which scales only linearly in the dimension d, hence avoiding the curse of dimensionality.

Imposing a low-rank structure on the coefficient tensor results in an embedded manifold of functions with low-rank tensors of rank at most $\mathbf{r} = (r_1, \dots, r_{d-1})$. In practice this can be interpreted as neglecting spurious features by controlling the rank parameters $r_1, \dots, r_{d-1}$ in the sense of an implicit feature selection. Analytical bounds, which quantify this behaviour rigorously can be found in (Oseledets, 2011; Oseledets and Tyrtyshnikov, 2010).

Utilising a TT format for the coefficient tensor W yields

$$f_W(x) = \sum_{\alpha_1,\dots,\alpha_d} \sum_{k_0=1}^{r_0} \dots \sum_{k_d=1}^{r_d} \prod_{i=1}^{d} V^{(i)}_{k_{i-1},\alpha_i,k_i} P_{\alpha_i}(x_i). \tag{3}$$

Ultimately, this model allows to perform practical computations in a very high dimensional space while staying trainable.

4. Tensor-Train Kernel Learning

As an application and extension of tensor network regression, Gaussian processes provide an immediate and principled way to equip an otherwise deterministic model with probabilistic features (see section A for a brief summary). Then again, tensor networks can also be beneficial to GP regression. Due to the inherent regularisation by the choice of basis functions and by imposing a low-rank structure, the problem of over-fitting as, e.g. encountered in standard deep kernel learning can be significantly reduced. Additionally, the low-rank structure performs implicit feature selection on the data, thus providing more expressive features for GP regression. Therefore, combining a low-rank tensor network model and GP should improve upon regular GP regression especially in the domain of high dimensional problems, where kernel evaluation is costly. Thus, the combination of a tensor network compression together with a GP model provides principled and flexible modelling options.

4.1. Tensor-Train Kernel Model

The general idea of our proposed method is to first propagate data through a TT function, then subsequently perform a GP regression on the resulting outputs (latents). Thus, the TT function serves as a feature extractor for the GP.

Concretely, we estimate an $\mathbb{R}$-valued random variable Y which depends on a d-dimensional $\mathcal{X}$-valued random vector X. Subsequently, we assume $Y = \Phi(X)$ and our goal is to approximate Φ by a GP g with TT feature extractor using samples of the pair (X, Y). Therefore, we start by defining an i.i.d. data set $D_N \coloneqq \{x^{(n)}, y^{(n)}\}$ for $n = 1, \dots, N$.

To this end, let $f_W \colon \mathcal{X} \to \mathcal{Z}$ denote a TT function with coefficient tensor W, mapping to the latent space $\mathcal{Z}$. This space need not be one-dimensional: multivariate latents can be achieved by adding a latent output dimension $d_{\mathcal{Z}}$ to the first component $V^{(1)}$ of the tensor train, transforming it to shape $r_1 \times J_1 \times r_2 \times d_{\mathcal{Z}}$. Now, assume the composite kernel

$$k(x, x'|\boldsymbol{\theta}) \coloneqq \hat{k}(f_W(x), f_W(x')|\theta),$$

where $\hat{k}\colon \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}$ is a base kernel with hyper-parameters θ, e.g. a squared-exponential or linear kernel, and $\boldsymbol{\theta} \coloneqq \{W, \theta\}$ are the joint hyper-parameters. Now, we perform regression with the composite kernel, yielding the GP prior

$$g(x) \sim \mathcal{GP}(m(x), k(x, x'|\boldsymbol{\theta})). \tag{4}$$

With $\mathbf{k}_i = k(x, x^{(i)}|\boldsymbol{\theta})$, $K_{i,j} = k(x^{(i)}, x^{(j)}|\boldsymbol{\theta})$, $\mathbf{y}_i = y^{(i)} - m(x^{(i)})$ this leads to the representation

$$m^{\Phi}(x) = m(x) + \mathbf{k}^{\intercal} K^{-1} \mathbf{y} \tag{5}$$

for the posterior GP's mean function m^{Φ} (for more details we refer again to section A).

With the described approach, we have defined a composition of a tensor network function and a GP, where the parameters of the TT are now treated as kernel hyper-parameters. Thus, we are able to jointly train our tensor network model and the GP using gradient-based optimisation. However, optimising the resulting composite hyper-parameters of the kernel globally by standard gradient descent neglects the multi-linear structure of the Tensor-Train. We thus prefer to pre-train the TT, using a regularised version of the Alternating Linear Scheme (ALS) (Holtz et al., 2011) (see section B).

4.2. Learning Tensor-Train Kernel Models

After pre-training the TT model with ALS, we resort to stochastic variational inference (VI) for training of a sparse GP with composite kernel $k(x, x'|\boldsymbol{\theta})$. For a thorough treatment of VI for GPs we further suggest (Leibfried et al., 2021).

The idea is to introduce a set of inducing point locations $\{x_1, x_2, \dots, x_M\}$, which are used as pseudo-training data in order to ensure a sparse GP. For VI we have the following optimisation problem

$$\begin{aligned} \max_{\boldsymbol{\theta},\gamma} \sum_{n=1}^{N} \mathbb{E}_{q_{\boldsymbol{\theta},\gamma}} \log p(y_n|g(x_n)) \\ - \mathrm{KL}(q_{\boldsymbol{\theta},\gamma}(u) \| p_{\boldsymbol{\theta},\gamma}(u)), \end{aligned} \tag{6}$$

where $q_{\boldsymbol{\theta},\gamma}$ is the variational density (depended on kernel hyper-parameters $\boldsymbol{\theta}$ and variational parameters γ) used to approximate the GP posterior with inducing variables $u = g(x)$, i.e. a point-wise evaluation of the GP at the inducing points. In VI related literature the stated objective is usually called evidence lower bound, ELBO for short.

Furthermore, we need the likelihood function $p(y_n|g(x_n))$ and prior density $p_{\boldsymbol{\theta},\gamma}(u)$ which corresponds to equation 4. Due to the summation over data points, the objective is amenable to stochastic mini-batching. In our case of regression, the expectation can be evaluated in closed-form, assuming $q_{\boldsymbol{\theta},\gamma}$ is a Gaussian. Otherwise, we rely on Monte-Carlo integration.

In our experiments, we introduce the inducing points not on the data space $\mathcal{X}$, but on the (potentially lower dimensional) latent space $\mathcal{Z}$ in order to reduce complexity. This leads to an overall asymptotic cost of
$\mathcal{O}(N(\max(J_1, \dots, J_d) d \max(r_1, \dots, r_{d-1})^2 + M^2))$ for $M \ll N$ operations.

5. Numerical Experiments

Tensor-Train kernel learning (TTKL) and five baselines[1], namely TT and sparse GP regression, a deep neural network (DNN), deep kernel learning (DKL) and the Canonical-Polyadic model from (Konstantinidis et al., 2021) (CPKL) are compared on three synthetic and six real-world data sets[2]. Moreover, we execute ablation experiments for TTKL.

As methodological tool for hyper-parameter optimisation of all models, we use a random search (Bergstra and Bengio, 2012) together with advanced early-stopping (Li et al., 2020). Additionally, we repeat model evaluation six times with randomly selected seeds and report the resulting mean and standard deviation.

Further information on the experiments, baselines, data sets and hyper-parameter optimisation is given in section C.

5.1. Tensor-Train Kernel Learning Set-up

The TT feature extractor uses orthonormal polynomials in a Sobolev space (H^1) of order one up to a fixed degree J as basis functions P_{α_i} which is also used to impose an H^1-regularisation during end-to-end training (see section B for details). Additionally, we constrain the tensor-train rank uniformly to a fixed value r. For pre-training, we use regularised ALS (Holtz et al., 2011), where we solve the underlying linear systems by LU factorisation. This yields improved initial values for subsequent stochastic gradient based optimisation of the tensor kernel model. For the base kernel $\hat{k}$, we use the standard radial basis function (RBF) kernel k_{RBF} (see section A). Subsequently, we use sparse variational inference (VI) (Leibfried et al., 2021) with a mean-field Gaussian density. Additionally, we assign individual initial learning rates for each TT component, the RBF kernel hyper-parameters and VI related parameters by specifying so-called parameter groups for PyTorch's optimiser class. As optimisation method we choose ADAM (Kingma and Ba, 2015) with default hyper-parameters, except for the initial learning rates. In order to adjust the TT's latent dimension from pre-training to end-to-end training, we simply duplicate the TT's first component to match.

In the ablation experiments we compare the individual additions we made to the TTKL model to one with all additions removed (Vanilla). That is, we examine the influence of pre-training (Pre-Train), H^1 regularisation (H^1) and TT component specific optimisation (Opt).

5.2. Results

In Table 3 we present estimates of the generalisation error of TTKL and the various baselines. Additionally, we give a parameter count for each trained model on the specific data set. We also provide the probabilistic models' log-likelihoods in Table 1, which by including uncertainty information, constitute a more "complete" metric. However, in our discussion, we will focus on the mean squared error, because it is available for probabilistic and deterministic models alike. Note that the results for DKL are different than those of (Wilson et al., 2016a), as we used a slightly different, richer architecture for the DNN (see C.3.2) and a standard

1. All models were implemented with the PyTorch (Paszke et al., 2019) and GPyTorch (Gardner et al., 2018) frameworks. Distributed model selection and evaluation were facilitated by means of Ray (Moritz et al., 2018) and Tune (Liaw et al., 2018).
2. Available at `https://drive.google.com/open?id=0BxWe_IuTnMFcYXhxdUNwRHBKTlU`

kernel (squared exponential), which we also used for all other model classes. The change in DNN architecture is due to non-reproducible results of the original paper. For the sake of consistency, we used sparse GP regression on all data sets, irrespective of the size.

The TTKL model outperforms all baselines in terms of mean squared error on five of the six real data sets, on four even by an order of magnitude. On the synthetic data sets, it surpasses the GP, DNN, DKL and CPKL models, while only TT regression achieves better results. We attribute the TT regression's strong performance to the structure of the synthetic data sets, which are readily approximated in a high dimensional polynomial space (Ali and Nouy, 2021). Moreover we can see that having a TT feature extractor improves upon a pure GP's generalisation error. Additionally, we argue that better performance of TTKL compared to DKL corroborates our earlier conjecture about better regularisation properties and therefore a reduced risk of over-fitting in TTKL. Another point we stress is the immensely reduced number of trainable parameters of the TTKL model when compared to the neural network feature extractor of DKL. Furthermore, especially in the high dimensional setting, the use of a TT feature extractor for GP regression even reduces the amount of parameters when compared to ordinary sparse GP regression. Moreover, construction of the Euclidean distance-based kernel matrix is sped-up, due to the kernel's evaluation on the lower dimensional latent space. Furthermore, we notice that CPKL fails in the high dimensional tasks. We attribute this to the exploding / vanishing gradient problem when training tensor network models purely by gradient descent. This behaviour underlines the importance of efficient and robust training methods for tensor based models such as ALS. With regard to the log-likelihood, TTKL outperforms the probabilistic baseline methods on all but one data set. This shows TTKL's ability to explain the data much better than the compared models.

In Table 2 we give results of the ablation experiments carried out on the synthetic data sets. Specifically, we remove all extensions, i.e. pre-training, H^1 regularisation and component specific optimisation, in order to receive a vanilla TTKL model. Subsequently, we only add one extension a time in order to investigate its effect on TTKL's performance. From these results we conclude that training methods specific to the structure of low-rank functions used in pre-training provide the most benefit. Next in line for consistently increased performance is H^1 based regularisation. Component specific gradient optimisation loses its benefit with increasing data dimension. Only on the low dimensional problem is the vanilla model able to achieve near comparable performance to extended models. We conclude that the interactions of extensions contribute the most to the performance of TTKL, especially in regard to increased feature dimension.

6. Conclusion

In this work we presented a new method for GP kernel learning in the form of low-rank functions parameterised by decomposed weight tensors. We derived the TTKL model and gave a short introduction into the learning method. Furthermore, we compared TTKL to five baseline models - namely TT and GP regression, a DNN, DKL and CPKL - on three synthetic and six real-world data sets. All models were subjected to extensive hyper-parameter optimisation and an average over six performance evaluations on an independent test set was reported together with the respective standard deviation. TTKL showed superior

Table 1: Log-likelihood (higher is better) and one standard deviation for all probabilistic models on three synthetic and six real-world data sets.

Data set	N	d	Test LL			
			TTKL (ours)	CPKL	DKL	GP
HighDimSin (Synthetic)	100 000	30	$\mathbf{2.46}$ $\pm \mathbf{9.51 \times 10^{-1}}$	-1.94 $\pm 6.90 \times 10^{-4}$	2.43×10^{-2} $\pm 7.84 \times 10^{-2}$	-3.43 $\pm 7.15 \times 10^{-2}$
Friedman (Synthetic)	100 000	5	$\mathbf{3.06}$ $\pm \mathbf{3.05 \times 10^{-2}}$	2.50 $\pm 1.28 \times 10^{-1}$	1.33×10^{-1} $\pm 6.66 \times 10^{-2}$	1.53 $\pm 9.32 \times 10^{-2}$
Grid (Synthetic)	65 536	2	$\mathbf{2.54}$ $\pm \mathbf{8.21 \times 10^{-2}}$	1.88 $\pm 5.52 \times 10^{-1}$	1.34 $\pm 9.02 \times 10^{-2}$	$\pm$
Kegg (Real)	48 827	22	$\mathbf{6.40 \times 10^{-1}}$ $\pm \mathbf{5.11 \times 10^{-1}}$	-1.74 $\pm 2.72 \times 10^{-5}$	3.56×10^{-1} $\pm 1.11 \times 10^{-1}$	3.41×10^{-1} $\pm 1.38 \times 10^{-2}$
Skillcraft (Real)	3338	19	-1.53 $\pm 4.39 \times 10^{-1}$	-7.20×10^{-1} $\pm 2.58 \times 10^{-1}$	$\mathbf{-2.95 \times 10^{-1}}$ $\pm \mathbf{3.14 \times 10^{-2}}$	-3.05×10^{-1} $\pm 7.30 \times 10^{-3}$
Elevators (Real)	16 599	18	$\mathbf{8.17 \times 10^{-1}}$ $\pm \mathbf{3.51 \times 10^{-2}}$	-8.04×10^{-2} $\pm 8.30 \times 10^{-2}$	7.16×10^{-2} $\pm 1.18 \times 10^{-2}$	7.61×10^{-2} $\pm 1.80 \times 10^{-3}$
Housing (Real)	506	13	$\mathbf{-9.46 \times 10^{-1}}$ $\pm \mathbf{2.78 \times 10^{-1}}$	-3.98 $\pm 8.89 \times 10^{-2}$	-3.54 $\pm 1.77 \times 10^{-1}$	-5.50 $\pm 3.04 \times 10^{-1}$
Protein (Real)	45 730	9	$\mathbf{1.11 \times 10^{-1}}$ $\pm \mathbf{4.17 \times 10^{-2}}$	-1.17 $\pm 7.00 \times 10^{-5}$	-7.49×10^{-1} $\pm 1.94 \times 10^{-1}$	-1.18 $\pm 1.94 \times 10^{-2}$
Kin40K (Real)	40 000	8	$\mathbf{2.15}$ $\pm \mathbf{1.34 \times 10^{-1}}$	3.11×10^{-1} $\pm 9.27 \times 10^{-1}$	7.92×10^{-1} $\pm 2.89 \times 10^{-2}$	-2.78×10^{-2} $\pm 1.72 \times 10^{-3}$

performance when compared to the other baselines on real-world data sets. Only on synthetic data could better performance be achieved by the TT regression. Moreover, we conducted ablation experiments. These clearly suggested that best performance of TTKL can only be achieved with the interaction of all extensions proposed for our method.

More importantly, we gave evidence of the benefit of low-rank functions for kernel learning on regression problems when compared to deep neural networks. Additionally, we introduced means to extend ordinary low-rank tensor regression with uncertainty estimates for its prediction. We hence showed that TTKL is a viable alternative to DKL and a promising research area.

Table 2: Mean squared error and one standard deviation for ablation experiments (vanilla, extended and full TTKL model) on three synthetic data sets.

Data set	N	d	Test MSE				
			Vanilla	H^1	Pre-Train	Opt	Full
HighDimSin	100 000	30	2.84 $\pm 1.04 \times 10^{-2}$	1.62×10^{-2} $\pm 4.75 \times 10^{-2}$	4.48×10^{-5} $\pm 1.92 \times 10^{-5}$	2.89 $\pm 7.92 \times 10^{-2}$	2.54×10^{-4} $\pm 3.75 \times 10^{-4}$
Friedman	100 000	5	1.92 ± 2.84	3.76×10^{-2} $\pm 8.59 \times 10^{-3}$	1.40×10^{-2} $\pm 1.71 \times 10^{-5}$	1.54×10^{-2} $\pm 1.91 \times 10^{-2}$	9.00×10^{-6} $\pm 9.42 \times 10^{-7}$
Grid	65 536	2	8.39×10^{-5} $\pm 2.77 \times 10^{-5}$	4.58×10^{-5} $\pm 2.91 \times 10^{-5}$	5.15×10^{-6} $\pm 1.60 \times 10^{-6}$	1.15×10^{-4} $\pm 5.26 \times 10^{-5}$	4.99×10^{-6} $\pm 8.29 \times 10^{-7}$

Table 3: Mean squared error, one standard deviation and number of trainable parameters for all models on three synthetic and six real-world data sets.

Data set	N	d	Test MSE Num Params					
			TTKL (ours)	CPKL	DKL	DNN	TT	GP
HighDimSin (Synthetic)	100 000	30	2.54×10^{-4} $\pm\ 3.75 \times 10^{-4}$ 26 609	2.83 $\pm\ 2.77 \times 10^{-4}$ 40 767	2.77×10^{-2} $\pm\ 5.97 \times 10^{-3}$ 58 007 943	3.34×10^{-2} $\pm\ 3.47 \times 10^{-3}$ 42 575 956	$\mathbf{2.69 \times 10^{-14}}$ $\pm\ \mathbf{1.87 \times 10^{-15}}$ 252 600	2.11×10^{-2} $\pm\ 2.94 \times 10^{-4}$ 18 512
Friedman (Synthetic)	100 000	5	9.00×10^{-6} $\pm\ 9.42 \times 10^{-7}$ 4308	1.20×10^{-4} $\pm\ 3.12 \times 10^{-5}$ 3746	1.14×10^{-2} $\pm\ 2.74 \times 10^{-3}$ 50 820 695	1.23×10^{-2} $\pm\ 1.88 \times 10^{-3}$ 2 052 701	$\mathbf{2.34 \times 10^{-16}}$ $\pm\ \mathbf{1.25 \times 10^{-16}}$ 1020	1.73×10^{-3} $\pm\ 2.85 \times 10^{-4}$ 2572
Grid (Synthetic)	65 536	2	4.99×10^{-6} $\pm\ 8.29 \times 10^{-7}$ 2182	1.12×10^{-3} $\pm\ 1.48 \times 10^{-3}$ 1394	1.06×10^{-3} $\pm\ 3.36 \times 10^{-4}$ 2 695 770	1.29×10^{-3} $\pm\ 1.97 \times 10^{-4}$ 2 614 358	$\mathbf{9.02 \times 10^{-8}}$ $\pm\ \mathbf{7.66 \times 10^{-12}}$ 52	2.65×10^{-6} $\pm\ 4.61 \times 10^{-7}$ 1030
Kegg (Real)	48 827	22	$\mathbf{1.93 \times 10^{-3}}$ $\pm\ \mathbf{3.85 \times 10^{-4}}$ 1469	1.89 $\pm\ 2.66 \times 10^{-5}$ 25 258	2.22×10^{-2} $\pm\ 1.60 \times 10^{-3}$ 2 999 434	1.76×10^{-2} $\pm\ 8.90 \times 10^{-4}$ 57 336 262	3.23×10^{-2} $\pm\ 2.78 \times 10^{-3}$ 19 448	2.39×10^{-2} $\pm\ 2.52 \times 10^{-4}$ 17 844
Skillcraft (Real)	3338	19	$\mathbf{3.61 \times 10^{-2}}$ $\pm\ \mathbf{1.21 \times 10^{-2}}$ 14 457	1.51×10^{-1} $\pm\ 1.12 \times 10^{-5}$ 18 318	8.18×10^{-2} $\pm\ 1.52 \times 10^{-3}$ 9 530 027	1.14×10^{-1} $\pm\ 5.76 \times 10^{-2}$ 57 292 606	3.18×10^{-1} $\pm\ 5.44 \times 10^{-2}$ 12 350	9.78×10^{-2} $\pm\ 1.92 \times 10^{-3}$ 6709
Elevators (Real)	16 599	18	1.14×10^{-2} $\pm\ 1.64 \times 10^{-3}$ 2366	6.31×10^{-2} $\pm\ 2.18 \times 10^{-6}$ 7400	4.91×10^{-2} $\pm\ 3.82 \times 10^{-4}$ 12 513 644	4.86×10^{-2} $\pm\ 3.23 \times 10^{-4}$ 14 920 122	$\mathbf{9.07 \times 10^{-3}}$ $\pm\ \mathbf{1.74 \times 10^{-4}}$ 52 200	4.99×10^{-2} $\pm\ 2.15 \times 10^{-4}$ 7346
Housing (Real)	506	13	$\mathbf{1.62 \times 10^{-1}}$ $\pm\ \mathbf{6.75 \times 10^{-2}}$ 2116	8.17×10^{1} $\pm\ 4.76 \times 10^{-1}$ 21 023	1.88×10^{1} $\pm$ 1.22 89 226 791	2.82×10^{1} $\pm$ 1.16 2 982 619	1.42×10^{1} $\pm\ 1.54 \times 10^{-1}$ 3770	3.83×10^{1} $\pm$ 2.62 1835
Protein (Real)	45 730	9	$\mathbf{5.18 \times 10^{-2}}$ $\pm\ \mathbf{7.96 \times 10^{-3}}$ 4754	6.04×10^{-1} $\pm\ 1.04 \times 10^{-5}$ 18 974	4.00×10^{-1} $\pm\ 2.04 \times 10^{-1}$ 7 557 785	2.06×10^{-1} $\pm\ 1.02 \times 10^{-2}$ 26 561 968	6.70×10^{-1} $\pm\ 7.70 \times 10^{-2}$ 1170	6.00×10^{-1} $\pm\ 1.08 \times 10^{-3}$ 6185
Kin40K (Real)	40 000	8	$\mathbf{3.49 \times 10^{-4}}$ $\pm\ \mathbf{1.84 \times 10^{-4}}$ 9916	1.82×10^{-2} $\pm\ 1.78 \times 10^{-2}$ 11 090	9.74×10^{-3} $\pm\ 5.09 \times 10^{-4}$ 9 466 377	1.13×10^{-2} $\pm\ 3.55 \times 10^{-4}$ 2 631 074	1.96×10^{-3} $\pm\ 6.43 \times 10^{-5}$ 3600	4.00×10^{-2} $\pm\ 4.62 \times 10^{-4}$ 7738

References

Mazen Ali and Anthony Nouy. Approximation with tensor networks. Part I: Approximation spaces. *arXiv e-print*, page arXiv:2007.00118, 2020a.

Mazen Ali and Anthony Nouy. Approximation with tensor networks. Part II: Approximation rates for smoothness classes. *arXiv e-print*, page arXiv:2007.00128, 2020b.

Mazen Ali and Anthony Nouy. Approximation with tensor networks. Part III: Multivariate approximation. *arXiv e-print*, page arXiv:2101.11932, 2021.

Markus Bachmayr, Reinhold Schneider, and Andre Uschmajew. Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations. *Foundations of Computational Mathematics*, 16:1423–1472, 2016.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, (13):281–305, 2012.

Philip Blagoveschensky and Anh-Huy Phan. Deep convolutional tensor network. In *First Workshop on Quantum Tensor Networks in Machine Learning, 34th Conference on Neural InformationProcessing Systems (NeurIPS 2020)*, 2020. URL https://tensorworkshop.github.io/NeurIPS2020/accepted_papers/dctn_article.pdf.

Andrzej Cichocki, Namgil Lee, Ivan V. Oseledets, Anh Huy Phan, Qibin Zhao, and Danilo P. Mandic. Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges PART 1. *CoRR*, abs/1609.00893, 2016. URL http://arxiv.org/abs/1609.00893.

Nadav Cohen and Amnon Shashua. Convolutional rectifier networks as generalized tensor decompositions. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 955–963. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/cohenb16.html.

Andreas C. Damianou and Neil D. Lawrence. Deep Gaussian processes. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*, volume 31 of *JMLR Workshop and Conference Proceedings*, pages 207–215. JMLR.org, 2013. URL http://proceedings.mlr.press/v31/damianou13a.html.

Sergey Dolgov, Boris N. Khoromskij, Alexander Litvinenko, and Hermann G. Matthies. Polynomial chaos expansion of random coefficients and the solution of stochastic partial differential equations in the tensor train format. *SIAM/ASA Journal on Uncertainty Quantification*, 3:1109–1135, 2015. doi: https://doi.org/10.1137/140972536.

Stavros Efthymiou, Jack Hidary, and Stefan Leichenauer. Tensornetwork for machine learning. *CoRR*, abs/1906.06329, 2019. URL http://arxiv.org/abs/1906.06329.

Martin Eigel, Max Pfeffer, and Reinhold Schneider. Adaptive stochastic galerkin FEM with hierarchical tensor representations. *Numerische Mathematik*, 136:765–803, 2017. doi: https://doi.org/10.1007/s00211-016-0850-x.

Martin Eigel, Manuel Marschall, and Reinhold Schneider. Sampling-free Bayesian inversion with adaptive hierarchical tensor representations. *Inverse Problems*, 34(3):035010, 2018. doi: 10.1088/1361-6420/aaa998.

Martin Eigel, Reinhold Schneider, Philipp Trunschke, and Sebastian Wolf. Variational Monte Carlo - bridging concepts of machine learning and high-dimensional partial differential equations. *Advances in Computational Mathematics*, 45:2503–2532, 2019. doi: https://doi.org/10.1007/s10444-019-09723-8.

Martin Eigel, Robert Gruhlke, and Manuel Marschall. Low-rank tensor reconstruction of concentrated densities with application to Bayesian inversion, 2020.

Jerome H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19 (1):1–67, 1991. URL http://www.jstor.org/stable/2241837.

Fu Jie Huang and Y. LeCun. Large-scale learning with SVM and convolutional for generic object categorization. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 284–291, 2006. doi: 10.1109/CVPR.2006.164.

Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, 2018.

I. Glasser, N. Pancotti, and J. I. Cirac. From probabilistic graphical models to generalized tensor networks for supervised learning. *IEEE Access*, 8:68169–68182, 2020. doi: 10.1109/ACCESS.2020.2986279.

Lars Grasedyck and Sebastian Krämer. Stable ALS approximation in the TT-format for rank-adaptive tensor completion. *Numerische Mathematik*, 143:855—-904, 2019. doi: https://doi.org/10.1007/s00211-019-01072-4.

Erwan Grelier, Anthony Nouy, and Mathilde Chevreuil. Learning with tree-based tensor formats, 2019.

Wolfgang Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*, volume 42. 01 2012. ISBN 978-3-642-28026-9. doi: 10.1007/978-3-642-28027-6.

Cole Hawkins and Zheng Zhang. Bayesian tensorized neural networks with automatic rank selection, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 630–645, Cham, 2016. Springer International Publishing.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. The alternating linear scheme for tensor optimization in the Tensor Train format. *SIAM Journal on Scientific Computing*, 01 2011. doi: 10.1137/100818893.

Robert Hu, Geoff K. Nicholls, and Dino Sejdinovic. Large scale tensor regression using kernels and variational inference. *arXiv e-prints*, page arXiv:2002.04704, 2020.

Pavel Izmailov, Alexander Novikov, and Dmitry Kropotov. Scalable Gaussian processes with billions of inducing inputs via Tensor Train decomposition. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 726–735. PMLR, 09–11 Apr 2018. URL https://proceedings.mlr.press/v84/izmailov18a.html.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Max Kirstein, Martin Eigel, and Dimitrios Bariamis. Bayesian tensor regression of random fields. In *First Workshop on Quantum Tensor Networks in Machine Learning, 34th Conference on Neural InformationProcessing Systems (NeurIPS 2020)*, 2020. URL https://tensorworkshop.github.io/NeurIPS2020/accepted_papers/btt_qtnml_2020_final.pdf.

Kriton Konstantinidis, Shengxi Li, and Danilo P. Mandic. Kernel learning with tensor networks. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2920–2924, 2021. doi: 10.1109/ICASSP39728.2021.9413826.

Felix Leibfried, Vincent Dutordoir, ST John, and Nicolas Durrande. A tutorial on sparse Gaussian processes and variational inference, 2021.

Liam Li, Kevin G. Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020. URL https://proceedings.mlsys.org/book/303.pdf.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training, 2018.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica.

Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577, Carlsbad, CA, October 2018. USENIX Association. ISBN 978-1-939133-08-3. URL https://www.usenix.org/conference/osdi18/presentation/moritz.

Alexander Novikov, Mikhail Trofimov, and Ivan V. Oseledets. Exponential machines. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rkm1sE4tg.

Sebastian W. Ober, Carl E. Rasmussen, and Mark van der Wilk. The promises and pitfalls of deep kernel learning. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence (UAI)*, Proceedings of Machine Learning Research. PMLR, 2021. URL https://proceedings.mlr.press/v124/kugelgen20a.html.

I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5): 2295–2317, 2011. doi: 10.1137/090752286. URL https://doi.org/10.1137/090752286.

Ivan Oseledets and Eugene Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, 2010. ISSN 0024-3795. doi: https://doi.org/10.1016/j.laa.2009.07.024. URL https://www.sciencedirect.com/science/article/pii/S0024379509003747.

Mathias Oster, Leon Sallandt, and Reinhold Schneider. Approximating the stationary Hamilton-Jacobi-Bellman equation by hierarchical tensor products, 2020.

Mathias Oster, Leon Sallandt, and Reinhold Schneider. Approximating optimal feedback controllers of finite horizon control problems using hierarchical tensor formats. *SIAM Journal on Scientific Computing*, 44(3):B746–B770, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

Ruslan Salakhutdinov and Geoffrey E. Hinton. Using deep belief nets to learn covariance kernels for Gaussian processes. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1249–1256. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper/2007/hash/4b6538a44a1dfdc2b83477cd76dee98e-Abstract.html.

Edwin M. Stoudenmire and David J. Schwab. Supervised learning with tensor networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4799–4807. Curran Associates, Inc., 2016.

Joost van Amersfoort, Lewis Smith, Andrew Jesson, Oscar Key, and Yarin Gal. On feature collapse and deep kernel learning for single forward pass uncertainty, 2021.

Steven R. White. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.*, 69:2863–2866, Nov 1992. doi: 10.1103/PhysRevLett.69.2863. URL https://link.aps.org/doi/10.1103/PhysRevLett.69.2863.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, volume 51 of *JMLR Workshop and Conference Proceedings*, pages 370–378. JMLR.org, 2016a. URL http://proceedings.mlr.press/v51/wilson16.html.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Stochastic variational deep kernel learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 2594–2602, Red Hook, NY, USA, 2016b. Curran Associates Inc. ISBN 9781510838819.

Jinfeng Zhuang, Ivor W. Tsang, and Steven C. H. Hoi. Two-layer multiple kernel learning. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 909–917. JMLR.org, 2011. URL http://proceedings.mlr.press/v15/zhuang11a/zhuang11a.pdf.

Appendix A. Gaussian Processes

We briefly recall the fundamentals of Gaussian process regression, following the notation used in the classical reference (Rasmussen and Williams, 2005).

Starting from the same point as in subsection 3.1, we assume random variables X, Y with realisations $\mathbf{X} = [x^{(1)}, \dots, x^{(N)}]$ drawn according to ρ and $\mathbf{y} = [y^{(1)}, \dots, y^{(N)}]$ with $y^{(n)} = \Phi(x^{(n)})$ for all $n = 1, \dots, N$. The goal now is to approximate Φ by f for which we assign a Gaussian process (GP) prior distribution f_0 to f. A GP (prior) f_0 is characterised by a mean function $m_0 \colon \mathcal{X} \to \mathbb{R}$ and a symmetric positive-definite function $k_0 \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Thus, we denote it by

$$f_0(x) \sim \mathcal{GP}(m_0(x), k_0(x, x')).$$

For any finite number of inputs, the corresponding function values of the GP have a joint Gaussian distribution

$$\begin{aligned} f_0(\mathbf{X}) &= [f_0(x^{(1)}), \dots, f_0(x^{(N)})] \\ &\sim \mathcal{N}(m_0(\mathbf{X}), k_0(\mathbf{X}, \mathbf{X})), \end{aligned}$$

where the mean vector $m_0(\mathbf{X}) = [m_0(x^{(1)}), \ldots, m_0(x^{(N)})]$ and covariance matrix $k_0(\mathbf{X}, \mathbf{X}) = k_0(x^{(i)}, x^{(j)})$ for all $i, j = 1, \ldots, N$ are defined by the mean function and covariance function, respectively.

In order to predict at new points $\mathbf{X}_* = (x_*^{(n+1)}, \ldots, x_*^{(n+m)})$ we condition the GP on the data, yielding the posterior process

$$f_*(x) \sim \mathcal{GP}(m_*(x), k_*(x, x')).$$

In this case, we obtain updated mean and covariance functions m_*, k_* with corresponding mean vector $m_*(\mathbf{X}_*) = m_0(\mathbf{X}_*) - k_0(\mathbf{X}_*, \mathbf{X}) k_0(\mathbf{X}, \mathbf{X})^{-1}(m_0(\mathbf{X}) - \mathbf{y}))$ and covariance matrix $k_*(\mathbf{X}_*, \mathbf{X}_*) = k_0(\mathbf{X}_*, \mathbf{X}_*) - k_0(\mathbf{X}_*, \mathbf{X}) k_0(\mathbf{X}, \mathbf{X})^{-1} k_0(\mathbf{X}, \mathbf{X}_*)$.

Furthermore, assuming a mean-zero prior GP we have

$$m_*(x) = \sum_{n=1}^{N} c_n k_0(x, x^{(n)}) \quad \forall\, x \in \mathcal{X}, \tag{7}$$

with $\mathbf{c} = [c_1, \ldots, c_N]$ and $\mathbf{c} = k_0(\mathbf{X}, \mathbf{X})^{-1}\mathbf{y}$. This is the definition of a linear (kernel) predictor. Therefore, the prior covariance function identifies the function space of the posterior mean as a reproducing kernel Hilbert space, namely $m_* \in \mathcal{K} := \operatorname{span}\{k_0(\cdot, x),\ x \in \mathcal{X}\}$.

For our purposes it is important to note that the covariance function k_0 usually depends on some set of hyper-parameters $\boldsymbol{\theta}$, i.e.

$$k_0(x, x') = k_0(x, x' | \boldsymbol{\theta}).$$

For example, the common RBF kernel

$$k_{\text{RBF}}(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2\ell}\|x - x'\|^2\right)$$

has the prior standard deviation σ_f, known as the signal variance, and the lengthscale ℓ, which determines the rate of correlation decay with increasing distance between inputs, as hyper-parameters. Together with the noise variance, the set of hyper-parameters for this simple kernel is given by $\boldsymbol{\theta} = \{\sigma_f, \ell, \sigma_n\}$.

A GP is trained by optimising the kernel hyper-parameters. This can be done in a principled way by maximising the marginal likelihood of the targets $\mathbf{y}$ conditioned on the data $\mathbf{X}$ and the hyper-parameters $\boldsymbol{\theta}$ w.r.t. $\boldsymbol{\theta}$:

$$\log\ p(\mathbf{y}|X, \boldsymbol{\theta}) \propto -\mathbf{y}^\intercal K_{\boldsymbol{\theta}}^{-1}\mathbf{y} - \log|K_{\boldsymbol{\theta}}|.$$

Here, $K_{\boldsymbol{\theta}}$ denotes the covariance matrix of the targets $\mathbf{y}$ given the hyper-parameters $\boldsymbol{\theta}$, i.e. $K_{\boldsymbol{\theta}} = k_0(\mathbf{X}, \mathbf{X})$. The first term gives a measure of how well the chosen kernel represents the data, while the second term penalises complexity. Normally, this second term prevents over-fitting as the marginal likelihood naturally favors models of intermediate complexity. However, as shown in (van Amersfoort et al., 2021; Ober et al., 2021), when used together with deep neural networks the regularisation effect is not strong enough.

Appendix B. Risk Minimisation with the Alternating Linear Scheme

Following (Grelier et al., 2019), we can formulate the risk minimisation problem from statistical learning as follows

$$\min_{f \in \mathcal{M}_r} L(f) \coloneqq \min_{f \in \mathcal{M}_r} \int_{\mathcal{X}} \ell(f; x)\, \rho(\mathrm{d}x), \tag{8}$$

where we choose $\ell(f;x) = (\Phi(x) - f(x))^2$. Furthermore, as hypothesis class we assume a manifold $\mathcal{M}_r$ of functions with rank-$\mathbf{r}$ TT coefficient which is embedded into the infinite dimensional tensor product function space $\mathcal{F}$.

Due to the intractability of the integral, we compute the empirical risk for a set of realisations $(x^{(n)}, y^{(n)} = \Phi(x^{(n)}))$ for $n = 1, \dots, N$, i.e.

$$L(f) \approx \frac{1}{N} \sum_{n=1}^{N} (y^{(n)} - f(x^{(n)}))^2. \tag{9}$$

To reduce over-fitting, we introduce regularisation, meaning that we minimise

$$\hat{L}(f) = \sum_{n=1}^{N} (y^{(n)} - f(x^{(n)}))^2 + \delta \|f\|_{\mathcal{F}}^2, \tag{10}$$

where δ is a hyper-parameter and the type of regularisation is dependent on the function space $\mathcal{F}$. Assuming that the data is contained in a d-dimensional cube $\Omega = (a,b)^d$, we choose $\mathcal{F} := H^1_{\text{mix}}(\Omega)$, which is defined as the tensor product $\bigotimes_{i=1}^{d} H^1((a,b))$. Here, $H^1((a,b))$ denotes the Sobolev space of square integrable functions on (a,b) whose weak derivative is again square integrable. By penalizing large derivative values in this way, we impose smoothness on the learned function. This regularization in mixed Soboloev Spaces for TTs has been used successfully already, e.g. in (Oster et al., 2022).

Each function $f \in \mathcal{M}_r$ can be parametrised by a list of tensors $(V^{(i)})_{i=1}^{d}$. Furthermore, by the tensor structure and Parseval's identity we have $\|f(\cdot; (V^{(i)})_i)\|_{\mathcal{F}}^2 = \|(V^{(i)})_i\|_F^2$, where $\|(V^{(i)})_i\|_F$ denotes the Frobenius-norm of the full tensor corresponding to $(V^i)_i$. Hence, we arrive at the finite dimensional minimisation problem

$$\begin{aligned} &\min_{(V^{(i)})_i} \hat{L}((V^{(i)})_i) = \\ &\min_{(V^{(i)})_i} \frac{1}{N} \sum_{n=1}^{N} (y^{(n)} - f(x^{(n)}; (V^{(i)})_i))^2 \\ &+ \delta \|((V^{(i)})_i\|_F^2, \end{aligned} \tag{11}$$

over all component tensors $V^{(i)}$. We can further exploit the multi-linear model structure by using an alternating optimisation method. The idea is to fix all but one component tensor $V^{(j)}$ and sweep back and forth over the tensor network, sequentially performing

$$\min_{V^{(j)}} \frac{1}{N} \sum_{n=1}^{N} (y^{(n)} - f(x^{(n)}; (V^{(i)})_i))^2 + \delta \|V^{(j)}\|_F^2, \tag{12}$$

where $V^{(i)}$ is fixed for all $i \neq j$. After the j-th component has been optimised it is orthogonalised using a QR-decomposition where the non-orthogonal part is shifted to the next component in order to ensure stability of the method. This procedure is known as the Alternating Linear Scheme (ALS) and was introduced in (Holtz et al., 2011). It is a method specific to the TT format, due to the latter's linearised tree structure. Hence, we have relaxed the non-convex learning problem such that only a sequence of linear systems has to be solved. Additionally, regularisation is performed naturally by suitable choice of orthonormal basis functions and penalisation of the Frobenius-norm.

Appendix C. Specifics for Numerical Experiments

C.1. Data Sets

Three synthetic and six real-world data sets with differing feature dimensionality are used for evaluating model performance.

For the first data set called Grid we assume $\mathcal{X} = [0,1]^2$ and set the true function to $\Phi(x) = \sum_{i=1}^{2} \sin(2\pi i x_i) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 0.1)$. Furthermore, we discretise the domain by an equidistant grid with 256 vertices and evaluate Φ on this grid, resulting in 65 536 total data points.

The second data set is known as Friedman data set (Friedman, 1991). Here, we assume $\mathcal{X} = [0,1]^5$ with the data generating measure $\rho = \bigotimes_{i=1}^{5} \mathcal{U}(0,1)$ and target function $\Phi(x) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5$. We draw an overall number of 100 000 data points from ρ.

The third data set represents a high dimensional sine function with a reasonable amplitude, hence the name HighDimSine. We let $\mathcal{X} = [0,1]^{30}$. Thus, we set $\rho = \bigotimes_{i=1}^{30} \mathcal{U}(0,1)$ and sample 100 000 data points. This time the target function is given by $\Phi(x) = \sum_{i=1}^{30} \sin(\pi x_i)$.

Furthermore, we test on the six data sets from the UCI Machine Learning Repository. Namely, the Physicochemical Properties of Protein Tertiary Structure, KEGG Metabolic Relation Network (Directed), Kin40K, SkillCraft1 Master, Elevators and Housing data set.

In order to conduct experiments, we split each data set into 70% training, 15% validation and 15% test data, where we use the training set in conjunction with early stopping on the validation set.

C.2. Tensor-Train Kernel Learning Hyper-parameters

The budgets for pre-training, hyper-parameter optimisation and test set evaluation are 20, 50 and 100 epochs respectively. For all instances we use early stopping on the validation set.

All hyper-parameters for Tensor-Train kernel learning and their respective search spaces are:

- Tensor-Train ranks: $r \sim \mathcal{U}(2, 15)$,
- H^1 polynomial degree: $J \sim \mathcal{U}(2, 14)$,
- ALS regularisation coefficient: $\delta_1 = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-10}), \log(1 \times 10^{-1}))$
- orthogonalisation of TT after pre-training with equal probability

- latents dimensionality: $L \sim \mathcal{U}(1, d)$
- number of inducing points: $M \sim \mathcal{U}(10, 1000)$
- TT regularisation during end-to-end training with equal probability
- TT regularisation coefficient during end-to-end training: $\delta_2 = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-10}), \log(1 \times 10^{-1}))$
- initial TT learning rate: $\eta_1 = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-5}), \log(1 \times 10^{-1}))$
- initial RBF hyper-parameters learning rate: $\eta_2 = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-4}), \log(1 \times 10^{-1}))$
- initial VI related hyper-parameters learning rate: $\eta_3 = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-3}), \log(1 \times 10^{-1}))$
- data batch size: $S \sim \mathcal{U}(4, 1024)$

C.3. Baselines

In this part, we present all baseline models together with their respective hyper-parameters and search spaces.

C.3.1. Canonical-Polyadic Kernel Learning

In order to take the results from (Konstantinidis et al., 2021) into consideration, we implement the Canonical-Polyadic model in PyTorch and use it as feature extractor. Similarly to the TTKL training, we also use different learning rates for core tensors, GP kernel hyper-parameters and VI related hyper-parameters. However, due to the structure of the Canonical-Polyadic model, pre-training with ALS is not possible, since the orthogonalisation procedure required for ALS cannot be performed. Therefore, we train by use of ADAM (Kingma and Ba, 2015), as suggested in (Konstantinidis et al., 2021). The budgets for hyper-parameters optimisation and test set evaluation are set to 50 and 1000, due to slower convergence. Early stopping is used on the validation set.

Hyper-parameters and respective search spaces used in the random search:

- Canonical-Polyadic rank: $r \sim \mathcal{U}(2, 15)$,
- polynomial degree: $J \sim \mathcal{U}(2, 20)$,
- latents dimensionality: $L \sim \mathcal{U}(1, d)$
- number of inducing points: $M \sim \mathcal{U}(10, 1000)$
- CP regularisation during end-to-end training with equal probability
- CP regularisation coefficient during end-to-end training: $\delta_2 = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-10}), \log(1 \times 10^{-1}))$
- initial CP learning rate: $\eta_1 = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-5}), \log(1 \times 10^{-1}))$

- initial RBF hyper-parameters learning rate: $\eta_2 = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-4}), \log(1 \times 10^{-1}))$
- initial VI related hyper-parameters learning rate: $\eta_3 = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-3}), \log(1 \times 10^{-1}))$
- data batch size: $S \sim \mathcal{U}(4, 1024)$

C.3.2. Deep Neural Network

To test against a deep learning model, we choose a variant of a pre-activation ResNet (including batch normalisation layers) (He et al., 2016; He et al., 2016) with trainable residual units to account for changes in dimensionality between layers. For all data sets, we assume 4 hidden layers to be sufficient. Furthermore, we use the ADAM optimisation algorithm (Kingma and Ba, 2015) with default hyper-parameters, except for the initial learning rate. The budgets for hyper-parameter optimisation and test set evaluation are 50 and 500 epochs respectively. For all instances we use early stopping on the validation set.

Hyper-parameters and respective search spaces used in the random search:

- output dimension of first and second hidden layer: $h_{1,2} \sim \mathcal{U}(1000, 10\,000)$
- output dimension of third hidden layer: $h_3 \sim \mathcal{U}(100, 1000)$
- output dimension of fourth hidden layer: $h_4 \sim \mathcal{U}(10, 100)$
- hidden layer's non-linearity is with equal probability either ReLU, Tanh or quadratic
- data batch size: $S \sim \mathcal{U}(4, 1024)$
- initial learning rate: $\gamma = 10^a$, $a \sim \mathcal{U}(\log(1 \times 10^{-5}), \log(1 \times 10^{-1}))$
- regularisation is either applied or not with equal probability
- regularisation coefficient: $\delta = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-10}), \log(1 \times 10^{-1}))$

C.3.3. Tensor Network

As a further baseline, we perform high dimensional tensor regression using the TT decomposition. The tensorised basis consists of polynomials of degree J in the Sobolev space H^1. We constrain the rank uniformly to r and train the model by ALS using LU factorisation to solve the resulting linear systems. Moreover, we regularise the problem by the H^1 norm. The budgets for hyper-parameter optimisation and test set evaluation are 25 and 100 epochs respectively. For all instances we use early stopping on the validation set. Preliminary experiments have shown that stochastic training might also be possible. The investigation of this objective is deferred to future work.

Again, hyper-parameters and their search spaces are:

- Tensor-Train rank: $r \sim \mathcal{U}(2, 15)$,
- H^1 polynomial degree: $J \sim \mathcal{U}(2, 14)$,
- regularisation coefficient: $\delta = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-10}), \log(1 \times 10^{-1}))$.

C.3.4. Gaussian Process

For the GP model we use the same training methods as for TTKL. The budgets for hyper-parameter optimisation and test set evaluation are 50 and 100 epochs respectively. For all instances we use early stopping on the validation set.

The GP hyper-parameters search spaces are:

- Number inducing points: $M \sim \mathcal{U}(100, 1000)$
- initial learning rate: $\gamma = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-3}), \log(1 \times 10^{-1}))$
- Batch size: $S \sim \mathcal{U}(4, 1024)$.

C.3.5. Deep Kernel Learning

For deep kernel learning the same deep architecture (and hyper-parameters) is deployed as for the DNN. However, we specify an additional hyper-parameter. Namely, we optimise over the DNN's output dimension. That is, we specify the number of features used for the subsequent GP regression. We adjust the output dimension by simply duplicating the parameters of the last layer to fit the correct number of outputs.

Exactly as in the TKL case, we train by sparse variational inference (VI) (Leibfried et al., 2021) in conjunction with ADAM. Moreover, we specify separate learning rates for the DNN, RBF kernel hyper-parameters and VI related parameters. The budgets for hyper-parameter optimisation and test set evaluation are 50 and 500 epochs respectively. For all instances we use early stopping on the validation set.

Hyper-parameters and search spaces for DKL are:

- latents dimensionality: $L \sim \mathcal{U}(1, d)$
- number of inducing points for sparse VI: $M \sim \mathcal{U}(10, 1000)$
- initial learning rate GP: $\gamma_1 = 10^a$, $a \sim \mathcal{U}(\log(1 \times 10^{-3}), \log(1 \times 10^{-1}))$
- initial RBF hyper-parameters learning rate: $\gamma_2 = 10^a$, $a \sim \mathcal{U}(\log(1 \times 10^{-4}), \log(1 \times 10^{-1}))$
- initial DNN learning rate: $\gamma_3 = 10^a$, $a \sim \mathcal{U}(\log(1 \times 10^{-5}), \log(1 \times 10^{-1}))$
- data batch size: $S \sim \mathcal{U}(4, 1024)$