Tensor Train Kernel Learning for Gaussian Process Regression

Max Kirstein
David Sommer
Martin Eigel

ProFit Project ReLkat - Reinforcement Learning for complex automation engineering

COPA 2022

August 25, 2022

Gaussian Process Regression

Goal: learn approximation f of a function

$$\Phi\colon \Omega \longrightarrow \mathbb{R}, \qquad \Omega = [a,b]^d \subset \mathbb{R}^d,$$

based on N realizations of random variables

- $\mathbf{X} = [x^{(1)}, \ldots, x^{(N)}]$ with $x^{(i)} \in \Omega$, $x^{(i)} \sim \rho$,
- $\mathbf{y} = [y^{(1)}, \ldots, y^{(N)}]$ with $y^{(i)} = \Phi(x^{(i)})$

for $i = 1, \ldots, N$.

Gaussian Process Regression

Goal: learn approximation f of a function

$$\Phi \colon \Omega \longrightarrow \mathbb{R}, \qquad \Omega = [a,b]^d \subset \mathbb{R}^d,$$

based on N realizations of random variables

- $\mathbf{X} = [x^{(1)}, \ldots, x^{(N)}]$ with $x^{(i)} \in \Omega$, $x^{(i)} \sim \rho$,
- $\mathbf{y} = [y^{(1)}, \ldots, y^{(N)}]$ with $y^{(i)} = \Phi(x^{(i)})$

for $i = 1, \ldots, N$.

A Gaussian Process (GP) prior

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

over f is characterised by

- a mean function $m_0 \colon \mathbb{R}^d \to \mathbb{R}$
- a spd covariance function $k_0 \colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

Gaussian Process Regression

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

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

with

- mean vector $m_0(\mathbf{X}) = [m_0(x^{(1)}), \dots, m_0(x^{(N)})]$
- and covariance matrix $(k_0(\mathbf{X}, \mathbf{X}))_{ij} = k_0(x^{(i)}, x^{(j)})$ for $i, j = 1, \dots, N$

Gaussian Process Regression

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

$$f_0(\mathbf{X}) = [f_0(x^{(1)}), \ldots, f_0(x^{(N)})] \sim \mathcal{N}(m_0(\mathbf{X}), k_0(\mathbf{X}, \mathbf{X})),$$

with

- mean vector $m_0(\mathbf{X}) = [m_0(x^{(1)}), \ldots, m_0(x^{(N)})]$
- and covariance matrix $(k_0(\mathbf{X}, \mathbf{X}))_{ij} = k_0(x^{(i)}, x^{(j)})$ for $i, j = 1, \ldots, N$

To predict at new points $\mathbf{X}_* = (x_*^{(1)}, \ldots, x_*^{(M)})$ we condition the GP on the data, yielding the posterior

$$f_*(\mathbf{X}_*) \sim \mathcal{N}(m_*(\mathbf{X}_*), k_*(\mathbf{X}_*, \mathbf{X}_*))$$

with

$$m_*(\mathbf{X}_*) = m_0(\mathbf{X}_*) - k_0(\mathbf{X}_*, \mathbf{X}) k_0(\mathbf{X}, \mathbf{X})^{-1}(m_0(\mathbf{X}) - \mathbf{y})),$$
$$k_*(\mathbf{X}_*, \mathbf{X}_*) = k_0(\mathbf{X}_*, \mathbf{X}_*) - k_0(\mathbf{X}_*, \mathbf{X}) k_0(\mathbf{X}, \mathbf{X})^{-1} k_0(\mathbf{X}, \mathbf{X}_*).$$

Gaussian Process Regression

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

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

with

- mean vector $m_0(\mathbf{X}) = [m_0(x^{(1)}), \dots, m_0(x^{(N)})]$
- and covariance matrix $(k_0(\mathbf{X}, \mathbf{X}))_{ij} = k_0(x^{(i)}, x^{(j)})$ for $i, j = 1, \dots, N$

To predict at new points $\mathbf{X}_* = (x_*^{(1)}, \dots, x_*^{(M)})$ we condition the GP on the data, yielding the posterior

$$f_*(\mathbf{X}_*) \sim \mathcal{N}(m_*(\mathbf{X}_*), k_*(\mathbf{X}_*, \mathbf{X}_*))$$

with

$$\begin{aligned} m_*(\mathbf{X}_*) &= m_0(\mathbf{X}_*) - k_0(\mathbf{X}_*, \mathbf{X}) k_0(\mathbf{X}, \mathbf{X})^{-1} (m_0(\mathbf{X}) - \mathbf{y})), \\ k_*(\mathbf{X}_*, \mathbf{X}_*) &= k_0(\mathbf{X}_*, \mathbf{X}_*) - k_0(\mathbf{X}_*, \mathbf{X}) k_0(\mathbf{X}, \mathbf{X})^{-1} k_0(\mathbf{X}, \mathbf{X}_*). \end{aligned}$$

From now on we assume $m_0 \equiv 0$.

Gaussian Process Regression

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 the RBF kernel

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

we have

- the prior standard deviation σ_f, i.e. signal variance
- the lengthscale ℓ, determining correlation decay rate with increasing distance between inputs

Gaussian Process Regression

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 the RBF kernel

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

we have

- the prior standard deviation σ_f, i.e. signal variance
- the lengthscale ℓ, determining correlation decay rate with increasing distance between inputs

With the noise variance σ_n, the set of hyper-parameters is given by $\boldsymbol{\theta} = \{\sigma_f, \ell, \sigma_n\}$.

Gaussian Process Regression

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 the RBF kernel

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

we have

- the prior standard deviation σ_f, i.e. signal variance
- the lengthscale ℓ, determining correlation decay rate with increasing distance between inputs

With the noise variance σ_n, the set of hyper-parameters is given by $\boldsymbol{\theta} = \{\sigma_f, \ell, \sigma_n\}$.
GP training: maximise marginal (log-)likelihood of targets over $\boldsymbol{\theta}$:

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

with $K_{\boldsymbol{\theta}} = k_0(\mathbf{X}, \mathbf{X})$.

Feature Extraction for GPs

Let

$$f_W : \mathbb{R}^d \to \mathcal{Z}$$

denote a feature extractor with parameters W mapping to a latent space $\mathcal{Z}$.

Feature Extraction for GPs

Let

$$f_W\colon \mathbb{R}^d \to \mathcal{Z}$$

denote a feature extractor with parameters W mapping to a latent space $\mathcal{Z}$.
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. RBF or linear kernel, and

$$\boldsymbol{\theta} \coloneqq \{W, \theta\}$$

are the joint hyper-parameters.

Feature Extraction for GPs

Let

$$f_W \colon \mathbb{R}^d \to \mathcal{Z}$$

denote a feature extractor with parameters W mapping to a latent space $\mathcal{Z}$.
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. RBF or linear kernel, and

$$\boldsymbol{\theta} \coloneqq \{W, \theta\}$$

are the joint hyper-parameters.
Possible feature extractors:

- Deep neural networks, W are the weights $\hookrightarrow$ DKL [WHSX16]
- Tensorized function spaces, W is the coefficient tensor $\hookrightarrow$ TTKL

Tensor Kernel Learning

Consider the case $\mathcal{Z} = \mathbb{R}$.
Start with a set of basis functions

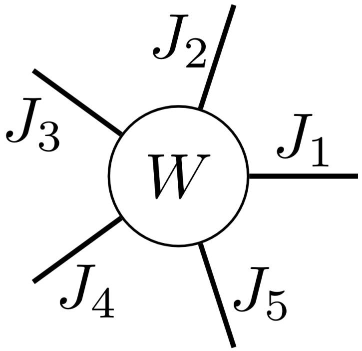
$$P_{\alpha_i} : \mathbb{R} \to \mathbb{R}, \qquad \alpha_i = 1, \dots, J_i, \qquad J_i \in \mathbb{N}, \qquad i = 1, \dots, d$$

and set for all $x = (x_1, \dots, x_d) \in \Omega$

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

with coefficient tensor

$$W \in \mathbb{R}^{J_1 \times \dots \times J_d}.$$



Tensor Kernel Learning

Consider the case $\mathcal{Z} = \mathbb{R}$.
Start with a set of basis functions

$$P_{\alpha_i} : \mathbb{R} \to \mathbb{R}, \qquad \alpha_i = 1, \ldots, J_i, \qquad J_i \in \mathbb{N}, \qquad i = 1, \ldots, d$$

and set for all $x = (x_1, \ldots, x_d) \in \Omega$

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

with coefficient tensor

$$W \in \mathbb{R}^{J_1 \times \ldots \times J_d}.$$

Remarks:

- Potentially high expressive power (think of P_{α_i} as polynomials up to degree $d-1$)
- For $J_i \equiv J$, W has storage complexity of $J^d \hookrightarrow$ *curse of dimensionality*

Alleviating the curse of dimensionality: Tensor Trains

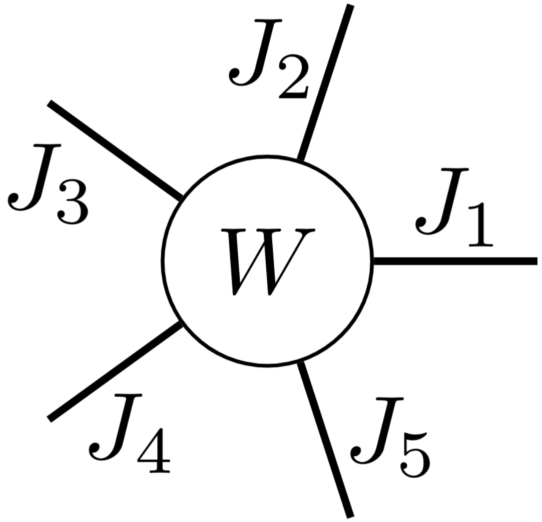


Figure: A full tensor W of order 5

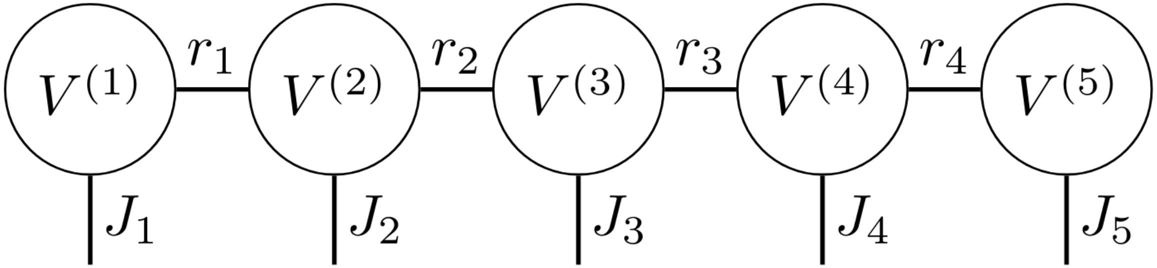


Figure: A Tensor Train (TT) decomposition of W with TT ranks (r_1, r_2, r_3, r_4)

Alleviating the curse of dimensionality: Tensor Trains

Recall

$$f_W(x) = \sum_{\alpha_1=1}^{J_1} \cdots \sum_{\alpha_d=1}^{J_d} W_{\alpha_1,\ldots,\alpha_d} \prod_{i=1}^{d} P_{\alpha_i}(x_i) \tag{2}$$

Alleviating the curse of dimensionality: Tensor Trains

Recall

$$f_W(x) = \sum_{\alpha_1=1}^{J_1} \cdots \sum_{\alpha_d=1}^{J_d} W_{\alpha_1,\ldots,\alpha_d} \prod_{i=1}^{d} P_{\alpha_i}(x_i) \tag{2}$$

A low-rank TT compression of the coefficient tensor W reads

$$W_{\alpha_1,\ldots,\alpha_d} \approx \sum_{k_0=1}^{r_0} \cdots \sum_{k_d=1}^{r_d} \prod_{i=1}^{d} V^{(i)}_{k_{i-1},\alpha_i,k_i} \tag{3}$$

with

- $V^{(i)} \in \mathbb{R}^{r_{i-1}\times J_i \times r_i}$, $i = 1,\ldots,d$,
- $r_0 = r_d = 1$
- TT ranks $r_1,\ldots,r_{d-1}$ determining accuracy of the compression.

Alleviating the curse of dimensionality: Tensor Trains

Recall

$$f_W(x) = \sum_{\alpha_1=1}^{J_1} \cdots \sum_{\alpha_d=1}^{J_d} W_{\alpha_1,\dots,\alpha_d} \prod_{i=1}^{d} P_{\alpha_i}(x_i) \tag{2}$$

A low-rank TT compression of the coefficient tensor W reads

$$W_{\alpha_1,\dots,\alpha_d} \approx \sum_{k_0=1}^{r_0} \cdots \sum_{k_d=1}^{r_d} \prod_{i=1}^{d} V^{(i)}_{k_{i-1},\alpha_i,k_i} \tag{3}$$

with

- $V^{(i)} \in \mathbb{R}^{r_{i-1}\times J_i \times r_i}$, $i = 1, \dots, d$,
- $r_0 = r_d = 1$
- TT ranks $r_1, \dots, r_{d-1}$ determining accuracy of the compression.

For every W, there exist TT-ranks such that equality holds in (3) [Ose11].

In our work: Optimise W only on a manifold of fixed TT rank.

Alleviating the curse of dimensionality: Tensor Trains

With $J_i \equiv J$, the coefficient tensor W of a fully tensorized function

$$f_W(x) = \sum_{\alpha_1,\ldots,\alpha_d} W_{\alpha_1,\ldots,\alpha_d} \prod_i P_{\alpha_i}(x_i) \tag{4}$$

has storage complexity $\mathcal{O}(J^d)$.

Alleviating the curse of dimensionality: Tensor Trains

With $J_i \equiv J$, the coefficient tensor W of a fully tensorized function

$$f_W(x) = \sum_{\alpha_1,\dots,\alpha_d} W_{\alpha_1,\dots,\alpha_d} \prod_i P_{\alpha_i}(x_i) \tag{4}$$

has storage complexity $\mathcal{O}(J^d)$.

On the other hand, the *coefficient Tensor Train* V of

$$f_V(x) = \sum_{\alpha_1,\dots,\alpha_d} \sum_{k_0,\dots,k_d} \prod_i V^{(i)}_{k_{i-1},\alpha_i,k_i} P_{\alpha_i}(x_i) \tag{5}$$

has storage complexity $\mathcal{O}(Jdr^2)$, where $r = \max_{i=1,\dots,d-1}\{r_i\}$.

The set of Tensor Trains with fixed TT-rank $\mathbf{r} = (r_1 \dots, r_d)$ is usually denoted $\mathcal{M}_{\mathbf{r}}$.

Pre-training with the Alternating Linear Scheme

The loss functional of the risk minimisation

$$L(f) := \int_{\Omega} (\Phi(x) - f(x))^2 \ \rho(\mathrm{d}x) \tag{6}$$

is approximated by the empirical risk minimisation for a set of iid realisations

$$x^{(n)} \sim \rho, \qquad y_n = \Phi(x^{(n)}),$$

for $n = 1, \dots, N$, i.e.

$$L(f) \approx \frac{1}{N} \sum_{n=1}^{N} \left(y^{(n)} - f(x^{(n)}) \right)^2 . \tag{7}$$

Pre-training with the Alternating Linear Scheme

The loss functional of the risk minimisation

$$L(f) \coloneqq \int_\Omega \left(\Phi(x) - f(x)\right)^2 \, \rho(\mathrm{d}x) \tag{6}$$

is approximated by the empirical risk minimisation for a set of iid realisations

$$x^{(n)} \sim \rho, \qquad y_n = \Phi(x^{(n)}),$$

for $n = 1, \dots, N$, i.e.

$$L(f) \approx \frac{1}{N} \sum_{n=1}^{N} \left(y^{(n)} - f(x^{(n)}) \right)^2 . \tag{7}$$

To reduce over-fitting, we introduce regularisation, minimising

$$\hat{L}(f) = \sum_{n=1}^{N} \left(y^{(n)} - f(x^{(n)}) \right)^2 + \delta \|f\|_{\mathcal{F}}^2, \tag{8}$$

for $\delta > 0$ and $\mathcal{F} \coloneqq H^1_{\text{mix}}(\Omega) = \bigotimes_{i=1}^{d} H^1([a, b])$.

Pretraining with the Alternating Linear Scheme

The minimisation problem reads

$$\min_{f_V} \hat{L}(f_V) = \sum_{n=1}^{N} \left(y^{(n)} - f_V(x^{(n)}) \right)^2 + \delta \| f_V \|_{\mathcal{F}}^2. \tag{9}$$

for f_V parametrised by a Tensor Train $V \in \mathcal{M}_{\mathbf{r}}$.

Pretraining with the Alternating Linear Scheme

The minimisation problem reads

$$\min_{f_V} \hat{L}(f_V) = \sum_{n=1}^{N} \left(y^{(n)} - f_V(x^{(n)}) \right)^2 + \delta \| f_V \|_{\mathcal{F}}^2. \tag{9}$$

for f_V parametrised by a Tensor Train $V \in \mathcal{M}_{\mathbf{r}}$.

Choosing the basis functions P_{α_i} as $H^1([a,b])$-orthonormal polynomials, the tensor structure and Parseval's identity yield

$$\| f_V \|_{\mathcal{F}}^2 = \| V \|_F^2,$$

where $\|.\|_F$ denotes the Frobenius-norm in full tensor space.

Pretraining with the Alternating Linear Scheme

The minimisation problem reads

$$\min_{f_V} \hat{L}(f_V) = \sum_{n=1}^{N} \left(y^{(n)} - f_V(x^{(n)}) \right)^2 + \delta \| f_V \|_{\mathcal{F}}^2. \tag{9}$$

for f_V parametrised by a Tensor Train $V \in \mathcal{M}_{\mathbf{r}}$.

Choosing the basis functions P_{α_i} as $H^1([a,b])$-orthonormal polynomials, the tensor structure and Parseval's identity yield

$$\| f_V \|_{\mathcal{F}}^2 = \| V \|_F^2,$$

where $\|.\|_F$ denotes the Frobenius-norm in full tensor space.

Hence, we arrive at the finite dimensional minimisation problem

$$\min_{V \in \mathcal{M}_{\mathbf{r}}} \hat{L}(f_V) = \min_{V \in \mathcal{M}_{\mathbf{r}}} \frac{1}{N} \sum_{n=1}^{N} \left(y^{(n)} - f_V(x^{(n)}) \right)^2 + \delta \| V \|_F^2, \tag{10}$$

Pretraining with the Alternating Linear scheme (ALS)

Idea of ALS [HRS11]: Sweep back and forth over the tensor network, sequentially performing

$$\min_{V^{(j)}} \frac{1}{N} \sum_{n=1}^{N} \Big(\Phi(x^{(n)}) - f_V(x^{(n)}) \Big)^2 + \delta \|V^{(j)}\|_F^2, \tag{11}$$

where $V^{(i)}$ is fixed for all $i \neq j$.

ALS optimization monotincally decreases continuously differentiable cost functionals.

Intermediate Recap: Tensor Train Kernel Learning

We train a GP with composite kernel

$$k(x, x'|\boldsymbol{\theta}) := \hat{k}(f_V(x), f_V(x')|\theta),$$

where

$$\hat{k} \colon \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}$$

is a base kernel with hyper-parameters θ, e.g. RBF, and

$$f_V \colon \mathbb{R}^d \longrightarrow \mathcal{Z}$$

is a TT function with H^1-orthonormal basis functions and coefficient tensor V.

The totality of hyperparameters is absorbed into $\boldsymbol{\theta}$.

Numerical Experiments - Overview

Following models[1] compared on three synthetic and six real-world (UCI) data sets:

- TTKL
- TT model
- Sparse GP
- Fully-connected deep neural network (DNN)
- DKL with DNN as feature extractor
- Canonical-Polyadic model from [KLM21] (CPKL)

We use random search [BB12] together with advanced early-stopping [LJR+20] for hyper-parameter optimisation. Additionally, we repeat model evaluation six times with randomly selected seeds and report the resulting mean and standard deviation.

[1] All models were implemented with the PyTorch [PGM+19] and GPyTorch [GPB+18] frameworks. Distributed model selection and evaluation were facilitated by means of Ray [MNW+18] and Tune [LLN+18].

Numerical Experiments - TTKL Set-up

- H^1-orthonormal polynomials P_{α_i} for fixed degree $\alpha_i = 1, \dots, J$
- TT rank uniformly constrained to r
- Base kernel $\hat{k} = k_{RBF}$
- ALS solved using LU factorisation
- Sparse variational inference (VI) [LDJD21] with mean-field Gaussian
- Individual learning rates for TT components, RBF kernel and VI hyper-parameters
- ADAM [KB15] with default hyper-parameters, except for the initial learning rates

Numerical Experiments - Results

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

Numerical Experiments - Results

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	TTKL (ours)	CPKL	DKL	DNN	TT	GP
			Test MSE / Num Params					
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	$\mathbf{2.69 \times 10^{-14}}$ $\mathbf{\pm 1.87 \times 10^{-15}}$ 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	$\mathbf{2.34 \times 10^{-16}}$ $\mathbf{\pm 1.25 \times 10^{-16}}$ 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	$\mathbf{9.02 \times 10^{-8}}$ $\mathbf{\pm 7.66 \times 10^{-12}}$ 52	2.65×10^{-6} $\pm 4.61 \times 10^{-7}$ 1030
Kegg (Real)	48 827	22	$\mathbf{1.93 \times 10^{-3}}$ $\mathbf{\pm 3.85 \times 10^{-4}}$ 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	$\mathbf{3.61 \times 10^{-2}}$ $\mathbf{\pm 1.21 \times 10^{-2}}$ 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	$\mathbf{9.07 \times 10^{-3}}$ $\mathbf{\pm 1.74 \times 10^{-4}}$ 52 200	4.99×10^{-2} $\pm 2.15 \times 10^{-4}$ 7346
Housing (Real)	506	13	$\mathbf{1.62 \times 10^{-1}}$ $\mathbf{\pm 6.75 \times 10^{-2}}$ 2116	8.17×10^{1} $\pm 4.76 \times 10^{-1}$ 21 023	1.88×10^{1} ± 1.22 89 226 791	2.82×10^{1} ± 1.16 2 982 619	1.42×10^{1} $\pm 1.54 \times 10^{-1}$ 3770	3.83×10^{1} ± 2.62 1835
Protein (Real)	45 730	9	$\mathbf{5.18 \times 10^{-2}}$ $\mathbf{\pm 7.96 \times 10^{-3}}$ 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	$\mathbf{3.49 \times 10^{-4}}$ $\mathbf{\pm 1.84 \times 10^{-4}}$ 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

[BB12] James Bergstra and Yoshua Bengio, *Random search for hyper-parameter optimization*, Journal of Machine Learning Research (2012), no. 13, 281–305.

[Fri91] Jerome H. Friedman, *Multivariate adaptive regression splines*, The Annals of Statistics **19** (1991), no. 1, 1–67.

[GPB+18] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson, *GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration*, Advances in Neural Information Processing Systems, 2018.

[HRS11] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider, *The alternating linear scheme for tensor optimization in the Tensor Train format*, SIAM Journal on Scientific Computing (2011).

[KB15] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (Yoshua Bengio and Yann LeCun, eds.), 2015.

[KLM21] Kriton Konstantinidis, Shengxi Li, and Danilo P. Mandic, *Kernel learning with tensor networks*, ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021, pp. 2920–2924.

[LDJD21] Felix Leibfried, Vincent Dutordoir, ST John, and Nicolas Durrande, *A tutorial on sparse Gaussian processes and variational inference*, 2021.

[LJR+20] 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*, Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020 (Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, eds.), mlsys.org, 2020.

[LLN+18] 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.

[MNW+18] 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*, 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18) (Carlsbad, CA), USENIX Association, October 2018, pp. 561–577.

[Ose11] I. V. Oseledets, *Tensor-train decomposition*, SIAM Journal on Scientific Computing **33** (2011), no. 5, 2295–2317.

[PGM+19] 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*, Advances in Neural Information Processing Systems 32 (H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, eds.), Curran Associates, Inc., 2019, pp. 8024–8035.

[WHSX16] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing, *Deep kernel learning*, Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016 (Arthur Gretton and Christian C. Robert, eds.), JMLR Workshop and Conference Proceedings, vol. 51, JMLR.org, 2016, pp. 370–378.

Numerical Experiments - Ablations

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} $\pm 8.29 \times 10^{-7}$

Numerical Experiments - Data Sets

Grid:

- $\Omega = [0,1]^2$
- $\Phi(x) = \sum_{i=1}^{2} \sin(2\pi i x_i) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 0.1)$
- $65\,536$ total data points (equidistant grid with 256 vertices)

Friedman [Fri91]:

- $\Omega = [0,1]^5$
- $\rho = \bigotimes_{i=1}^{5} \mathcal{U}(0,1)$
- $\Phi(x) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5$
- $100\,000$ data points

HighDimSine:

- $\Omega = [0,1]^{30}$
- $\rho = \bigotimes_{i=1}^{30} \mathcal{U}(0,1)$
- $\Phi(x) = \sum_{i=1}^{30} \sin(\pi x_i)$
- $100\,000$ data points

UCI Machine Learning Repository:

- Physicochemical Properties of Protein Tertiary Structure
- KEGG Metabolic Relation Network (Directed)
- Kin40k
- SkillCraft1 Master
- Housing

Hyper-Parameters

TTKL:

- 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: $S \sim \mathcal{U}(4, 1024)$

Hyper-Parameters

CPKL:

- 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: $S \sim \mathcal{U}(4, 1024)$

Hyper-Parameters

DNN:

- 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: $\delta = 10^b$, $b \sim \mathcal{U}(\log(1 \times 10^{-10}), \log(1 \times 10^{-1}))$

Hyper-Parameters

TT:

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

Hyper-Parameters

GP:

- 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: $S \sim \mathcal{U}(4, 1024)$.

Hyper-Parameters

DKL:

- 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: $S \sim \mathcal{U}(4, 1024)$