Tensor Train Decomposed Bayesian Kernel Machines.

D.A.G. van Haasteren

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Tensor Train Decomposed Bayesian Kernel Machines.

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

D.A.G. van Haasteren

May 26, 2026

Faculty of Mechanical Engineering (ME) · Delft University of Technology

Abstract

Kernel machines are a class of machine learning models which are powerful in terms of predictive power, but are limited to low dimensional data, as their computational complexity scales according to $\mathcal{O}(N^3)$. Kernel methods are able to reduce this computational scaling to be linear in D by assuming a low rank structure on the weights vector. Existing tensor network kernel machines operate mostly in a deterministic setting and lack uncertainty quantification. On top of this, hyperparameter tuning for these types of models can be costly and complex. A fully Bayesian kernel machine, employing a CP decomposition has been shown to perform well on regression and classification tasks without additional computational cost. hyperparameter tuning was aided for this model through the introduction of ARD priors. This model is still limited to a relatively small data dimension D, due to numerical instabilities caused by repeated Hadamard products in the CPD formulation. Alternative tensor decompositions such as the tensor train decomposition exist that do not rely on Hadamard products in their formulation and may therefore be more numerically stable.

This thesis proposes a Bayesian Tensor Train Kernel Machine, a fully Bayesian tensor train decomposed kernel machine. The proposed model will include ARD hyperpriors to aid in model selection, these priors allow for automatic inference of model complexity, as well as increasing model interpretability. Mean-field variational inference is used to approximate posterior distributions of all model parameters and hyperparameters. Experiments on synthetic datasets show the functioning of the ARD hyperpriors, and the increased numerical stability compared to the CP based model. Experiments on real world datasets showcases the performance of the proposed model in terms of prediction accuracy and uncertainty quantification compared to currently available models.

Table of Contents

Acknowledgements

I would like to thank my supervisor dr.ir. K. Bastselier for his time and assistance throughout the process of writing this thesis. In addition, I would like to thank A. Kilic for undertaking this learning journey with me, and for all her help, especially with the derivations of the update equations. I would also like to thank dr. M. Mazo for taking time out of his schedule to read my thesis and be present for my presentation. Lastly thanks to Paolo, Ilse and Saar for proofreading and catching my many spelling errors.

My entire student life has been coloured by my housemates at Koevoet who I want to thank for making the past 9 years great. As well as my family, who will now need to find a different catchphrase come December. Lastly I would like to thank Saar for being the very best as always, giving me endless support, a listening ear and endless good (occasionally recycled) advice.

Delft, University of Technology
May 26, 2026

D.A.G. van Haasteren

Chapter 1

Introduction

In recent years, the interest in Artificial Intelligence (AI) has increased drastically. Kernel machines are universal function estimators, which in many settings meet or exceed the performance of state-of-the-art neural networks [14, 17, 32, 33, 39]. Kernel machines such as Support Vector Machine (SVM) [8] models have broad applications ranging from medical image classification [34], text classification [16] and EMG classification [2]. Kernel models such as Kernel Ridge Regression (KRR) [23] and Gaussian Process (GP) [43] are used for regression tasks and have applications in the prediction of material properties from molecular and chemical descriptors [48], and prediction of exoplanet characteristics [1] to non-linear system identification [11].

Kernel machines simplify the learning of high dimensional, highly non-linear datamodels, by employing the *kernel trick*. Within the kernel trick we employ a kernel function $k(x, x') = \langle \varphi(x), \varphi(x') \rangle$ to implicitly map input data into a Reproducing Kernel Hilbert Space (RKHS) [4], where initially non-linear problems can be solved using linear techniques. Traditional machine learning methods, often suffer from the *curse of dimensionality*, which is the exponential increase of storage and training complexity with respect to data dimension D. Although this *kernel trick* effectively circumvents the curse of dimensionality, they instead scale poorly with the data size N. Kernel machines scale according to $\mathcal{O}(N^2)$ in memory and $\mathcal{O}(N^3)$ in training complexity [46]. This limits their use to relatively small datasets [13].

Numerous techniques exist to approximate kernel machines in order to reduce their complexity, however, each of these come with their own drawbacks. Sampling techniques are popular due to their simplicity. They work by projecting training data onto finite dimensional basis functions, reducing the complexity to $\mathcal{O}(NM^2)$, where M is the dimensionality of the basis functions. The number of basis functions can be chosen to be far fewer than the number of datapoints $M \ll N$ without a major loss in performance [12, 41, 51]. While these techniques are able to reduce the computational complexity, they are slow to converge. Due to the nature of sampling techniques they converge at the Monte Carlo rate of $\mathcal{O}(1/\sqrt{M})$. This introduces a trade-off between training complexity and convergence time.

Deterministic approaches that do not require random sampling exist [9, 37, 21, 45], and are

able to converge much faster. These however, cause M to grow exponentially with the data dimensionality D, and are therefore limited to low-dimensional applications.

Tensor decompositions such as Tensor Train (TT) [40] and Canonical Polyadic Decomposition (CPD) [19, 6] (explained in detail in Chapter 1-2) are able to reduce the complexity without compromising on the convergence rate, While offering greater freedom in the dimensionality of the basis functions. These models are able to reduce the number of parameters to be learned, from M^D to DMR^2 or DMR depending on the decomposition used, where R is a rank parameter, and M and R can be chosen so that $M, R \ll N, D$.

Tensor Network Kernel Machine (TNKM) models consider a datamodel

$$f(\mathbf{x}_n) = \varphi(\mathbf{x}_n)^\intercal \mathbf{w}, \tag{1-1}$$

where $\mathbf{w}$ is a weights vector with learnable weights. And $\varphi(\cdot)$: $\mathbb{R}^D \to \mathbb{R}^{M^D}$ is a feature map, restricted to a Kronecker product form

$$\varphi(\mathbf{x}_n) = \varphi^{(1)}(x_n^{(1)}) \otimes \varphi^{(2)}(x_n^{(2)}) \otimes \cdots \otimes \varphi^{(D)}(x_n^{(D)}). \tag{1-2}$$

Here each $\varphi^{(d)} \in \mathbb{R}^{M_d}, \forall d \in [1, D]$ is a feature vector. A low rank constraint is placed on the feature vector $\mathbf{w}$, allowing it to be decomposed into a Tensor Network (TN). The Kronecker product form limits the choice of kernels to the family of product kernels, which include common kernels such as the polynomial and Gaussian kernels. The Kronecker structure in the feature map $\varphi(\cdot)$ and low rank assumption on the weights vector $\mathbf{w}$ allow the weights to be learned via an Alternating Least Squares (ALS) algorithm, with computational complexity of $\mathcal{O}(NM^2R^2 + DM^3R^3)$ or $\mathcal{O}(NM^2R^4 + DM^3R^6)$ depending on the decomposition used [27]. Currently, a number of TNKMs exist [44, 20, 47, 58, 18, 24, 51, 41, 50, 49] that have shown that efficient learning via TNs is possible. One drawback of tensor networks is that they require specifying model complexity and rank in advance through hyperparameters such as the feature dimension M and tensor ranks R.

Most of the existing models operate in a deterministic setting, and lack methods to quantify uncertainty. Uncertainty quantification can give crucial confidence metrics when dealing with parameter uncertainty [22], or real world uncertainty [10]. Bayesian Inference provides a principled method for handling model and data uncertainty and allows for the incorporation of prior knowledge. Furthermore, it aids in limiting the complexity of the model and can promote sparsity, further reducing the model size. Within tensor network kernel machines, the incorporation of a probabilistic framework remains largely unexplored.

The extension into a probabilistic setting also allows the incorporation of Automatic Relevance Detection (ARD) priors that aid in determining the model's hyperparameters. ARD priors penalize irrelevant components to shrink towards zero during training, allowing for automatic inference of the tensor rank and feature dimension. Additionally, the sparsity parameter placed on the feature dimension highlights the most relevant features, aiding in model interpretability. These priors are based on the work by Neal [38] and MacKay [35], who introduced them in the setting of neural networks.

Currently one model exists that employs a fully Bayesian tensor network kernel machine with ARD priors for both rank and feature dimension inference. This Bayesian Tensor Network (BTN)-Kernels model, by Kilic et al. [27], employs a CPD with its parameters learned via mean field variational inference. While this model is shown to be computationally efficient and

reduces the computational complexity to $\mathcal{O}(NM^2R^2 + DM^3R^3)$, it is susceptible to numerical over and underflow issues at high data dimensionality D. For probabilistic tensor network kernel machines to become a viable and scalable machine learning model, a more numerically robust tensor network kernel machine is required, which will be explored in this thesis.

1-1 Research Questions

In summary, the aim of this thesis will be to create a scalable probabilistic tensor network kernel machine, through the use of a TT decomposition, incorporating ARD priors to aid in hyperparameter tuning as well as increasing model interpretability. This goal is summarised in the main research question of this thesis:

Can we construct a scalable, fully Bayesian, tensor train decomposed kernel machine including ARD priors that automatically infers both tensor ranks and feature dimensions?

In order to answer this question and evaluate the proposed model against existing models, we will break it into three sub-questions. These are formulated as:

1. *Can ARD priors automatically infer TT-ranks R_d and feature dimensions M_d of a tensor train kernel machine for all $d \in [1, D]$ of a groundtruth datamodel?*
2. *Is a Bayesian TT kernel machine more robust to over and underflow issues than a comparable model using a CP decomposed model weights vector?*
3. *Can a Bayesian TT kernel machine achieve predictive accuracy and uncertainty quantification performance comparable to currently available models?*

In order to answer these questions, in the next chapter we will first introduce some required mathematical background in Chapter 1-2, before exploring existing probabilistic TNKMs in Chapter 1-3. Chapter 2 will describe the proposed model, with its construction and priors in Chapter 2-1, as well as the learning algorithm that is used in Chapter 2-2. In Chapter 2-3 a description is given how this model has been implemented, as well as an exploration into the robustness to over and underflow issues. The three sub-questions are investigated in order in Chapter 3, where three experiments are described and their results are shown and discussed. The first of these is an experiment that explores the TT-rank and feature dimension recovery of the ARD priors in Chapter 3-1. The numerical robustness is further explored in Chapter 3-2. Lastly, the model is benchmarked against existing models on six UCI datasets in Chapter 3-3. Finally, Chapter 4 concludes this thesis and discusses possible future work.

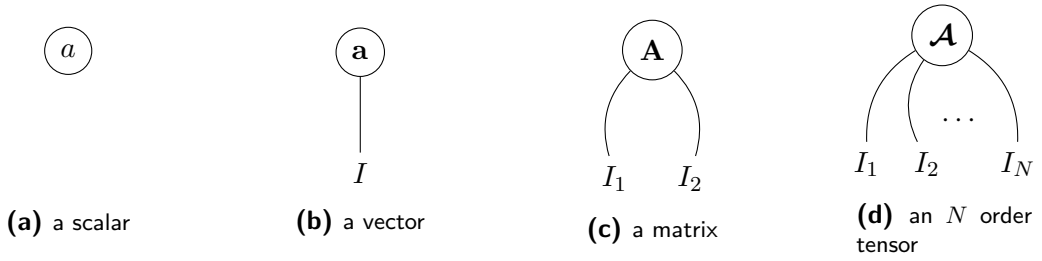
1-2 Mathematical Background

Many mathematical operations that exist for matrices can be extended to work for tensors, on top of this some operations exist that are unique for tensors. In the following section, some of the mathematical notation and operations that are used in this paper will briefly be explained.

1-2-1 Preliminaries and Notation

A *tensor* is a multilinear collection of numbers, where the *order* of the tensor is the number of *ways* or *modes* in which the numbers are organised. A scalar, a single number, denoted by a lowercase letter, a is a 0th order tensor. A vector, which is a collection of numbers usually ordered into a column, is denoted by a bold lowercase letter $\mathbf{a}$, equivalently is a 1-st order tensor. A matrix, a collection of numbers ordered into rows and columns, is denoted by an uppercase bold letter $\mathbf{A}$ is a 2-nd order tensor. Stacking a number of matrices back to back, over an extra dimension results in a 3-rd order tensor. A three or more way ordering of numbers is called a tensor and is denoted by an uppercase calligraphic boldface letter $\boldsymbol{\mathcal{A}}$. For a D-th order tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$, the $(i_1, i_2, \ldots, i_D)$-th element is denoted by $a_{i_1, i_2, \ldots, i_D}$.

Tensor diagrams can be used to express a tensor network graphically. A tensor is denoted as a node, with edges equal to the order of the tensor.



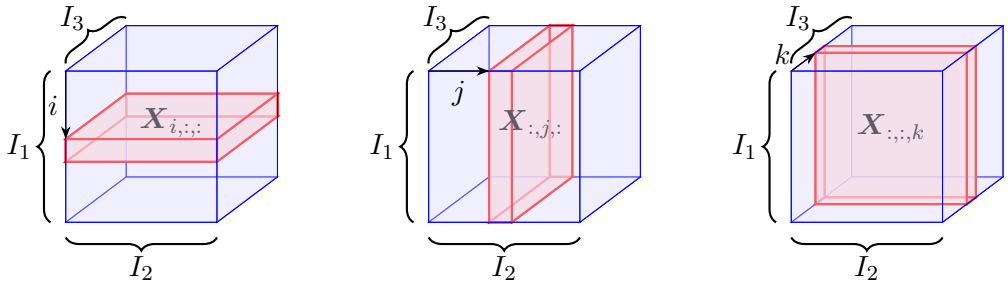
(a) a scalar **(b)** a vector **(c)** a matrix **(d)** an N order tensor

Figure 1-1: (a) Graphical representations of a scalar denoted as a lowercase letter a and graphically as a node with no edges. **(b)** A vector (boldface lowercase $\mathbf{a}$) is a node with a single edge, with size I. **(c)** A matrix (boldface uppercase $\mathbf{A}$) is denoted by a node with two edges, with sizes I_1 and I_2. **(d)** A tensor denoted as a boldface, calligraphic uppercase letter $\boldsymbol{\mathcal{A}}$, or graphically by a node with N edges, with sizes $I_1, I_2, \ldots I_N$.

The *vectorization* operator $\text{vec}(\cdot)$, applied to a tensor D^{th} order $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ returns a vector $\mathbf{a} \in \mathbb{R}^{I_1 \ldots I_D}$, containing the same elements as $\boldsymbol{\mathcal{A}}$ where the relationship between the indices of the elements in the matrix and those of the tensor are given by

$$i = i_1 + \sum_{d=2}^{D} (i_d - 1) \prod_{k=1}^{d-1} I_k. \tag{1-3}$$

A subset of a tensor can be extracted by fixing one or more indices and collecting all the elements with that particular index. For a three dimensional tensor the objects resulting from fixing all but the first, second and third index are called *rows*, *columns* and *tubes* respectively. Similarly, fixing all but two of the indices results in a matrix. The resulting 2-nd order subtensors are also called *slices*. The slices created from a 3-rd order tensor by fixing its first second and third indices are called *horizontal*, *lateral* and *frontal* slices [29]. These are shown graphically in Figure 1-2.



(a) A horizontal slice **(b)** A lateral slice **(c)** A frontal slice

Figure 1-2: Visual representation of the slices of a three way tensor.

The *Frobenius Norm* of a tensor, denoted as $||\cdot||_F$ is defined as the square root of the sum of the absolute squares of its elements.

While many mathematical operations are defined for tensor objects, in implementation these are often performed by transforming tensors into matrices, performing a matrix-matrix operation and transforming the product back into a tensor. This transformation is achieved by an *n-mode unfolding*. An n-mode unfolding is constructed by stacking all the slices of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ by fixing the n-th index such that [28]

$$\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times \prod_{m \neq n} I_m}. \tag{1-4}$$

More general transformations, such as the permutation of matrix rows or columns can be performed by tensorizing a matrix, and unfolding it in a different order. These types of operations are denoted by the *reshape* operator $\mathcal{R}\{\cdot\}_{I \times J}$. This operation reshapes a tensor into the shape $I \times J$ while maintaining the column-wise ordering consistent with the vec$(\cdot)$ operation. The *diagonal* operator diag$(\cdot)$, when applied to a matrix, will return the values on the main diagonal of the matrix, while when applied to a vector, it will return a diagonal matrix, with the values of the vector on the main diagonal of the matrix. The diagonal of a tensor of all elements for which all indices are equal. A diagonal slice of a three way tensor with at least two sides with equal dimensions, is all elements for which those two indices are equal.

A *Kronecker* product between two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}, \mathbf{B} \in \mathbb{R}^{K \times L}$ is denoted as $(\mathbf{A} \otimes \mathbf{B}) \in \mathbb{R}^{KI \times LJ}$ and defined as the block matrix

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & \cdots & a_{1,J}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{I,1}\mathbf{B} & \cdots & a_{I,J}\mathbf{B} \end{bmatrix}.$$

The *Khatri-Rao* product, which is a column-wise Kronecker product, between two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}, \mathbf{B} \in \mathbb{R}^{K \times J}$ is denoted as $\mathbf{A} \odot \mathbf{B} \in \mathbb{R}^{JI \times J}$ [26]. Similarly, a *row-wise Khatri-Rao* product exists, as a row-wise Kronecker product with $\mathbf{A} \in \mathbb{R}^{I \times J}, \mathbf{B} \in \mathbb{R}^{I \times K}$ resulting in $\mathbf{A} \odot_R \mathbf{B} \in \mathbb{R}^{I \times KJ}$. A *Hadamard* product between two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}, \mathbf{B} \in \mathbb{R}^{I \times J}$ denoted by $\mathbf{A} \circledast \mathbf{B} \in \mathbb{R}^{I \times J}$ is the element-wise multiplication between $\mathbf{A}$ and $\mathbf{B}$.

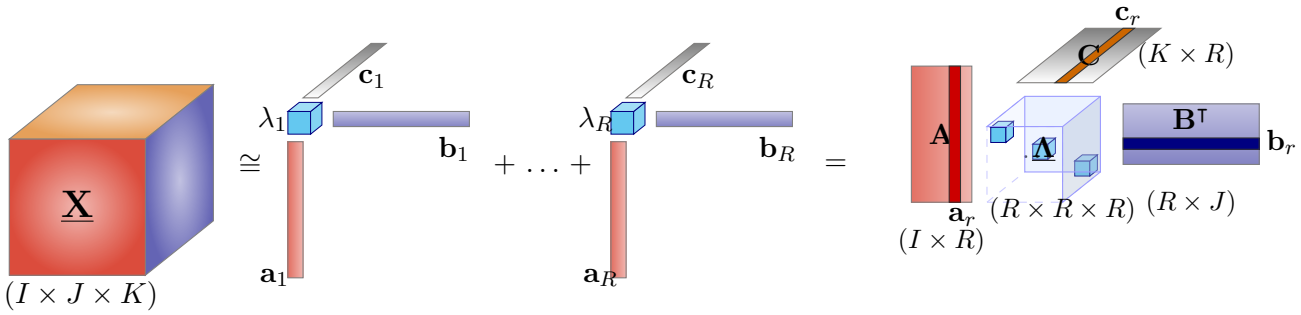


Figure 1-3: A 3-way Tensor, can be decomposed into a sum of rank-1 terms via a Canonical Polyadic Decomposition. The rank-1 terms can be collected into a core tensor and three factor matrices.

1-2-2 Tensor Decompositions

While tensors can be a logical format for some multidimensional data, decomposing a tensor into a tensor network can be advantageous as it might lead to better interpretability and lower storage and computational cost. Many Tensor Decompositions exist with their own uses and advantages and disadvantages. In this section the CPD and TT decomposition will be introduced and their advantages and disadvantages discussed.

Canonical Polyadic Decomposition

A Canonical Polyadic Decomposition (CPD) decomposes a D-way tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{M_1 \times \ldots \ M_D}$ into a summation of R rank one terms. These rank one terms can be collected into D *factor matrices* $\mathbf{W}^{(d)} \in \mathbb{R}^{M_d \times R}$ such that

$$\boldsymbol{\mathcal{W}} = \mathbf{W}^{(1)} \times_1 \mathbf{W}^{(2)} \times_2 \cdots \times_D \mathbf{W}^{(D)} = \sum_{r=1}^{R} \mathbf{w}_r^{(1)} \circ \mathbf{w}_r^{(2)} \circ \cdots \circ \mathbf{w}_r^{(D)}. \tag{1-5}$$

Here $'\circ'$ denotes a vector outer product, and $\mathbf{w}_r^{(d)}$ denotes the r-th column of the d-th factor matrix. This reduces the storage complexity of the decomposed tensor from $\mathcal{O}(M^D)$ to $\mathcal{O}(MRD)$. A CPD is unique under mild conditions [7]. That is to say, they are unique up to scaling and permutation ambiguities.

Tensor Train Decomposition

A Tensor Train (TT) decomposes a tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{M_1 \times \cdots \times M_D}$ into D 3-way tensors $\boldsymbol{\mathcal{W}}^{(d)} \in \mathbb{R}^{R_d \times M_d \times R_{d+1}}$ such that the elements of $\boldsymbol{\mathcal{W}}$ can be expressed as

$$w_{i_1, m_2, \ldots, i_D} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_{D+1}=1}^{R_{D+1}} w_{r_1 m_1 r_2}^{(1)} \ldots w_{r_D m_D r_{D+1}}^{(D)}.$$

The constituent tensors of the tensor train are called *TT-cores* and their ranks $R_1, R_2, \ldots, R_D$ are called TT-ranks. In order to ensure that the decomposed tensor has the same dimensions as the original tensor, boundary conditions $R_1 = R_{D+1} = 1$ are enforced. TT-decompositions are non-unique, and have a storage complexity of $\mathcal{O}(R^2 MD)$ [40].

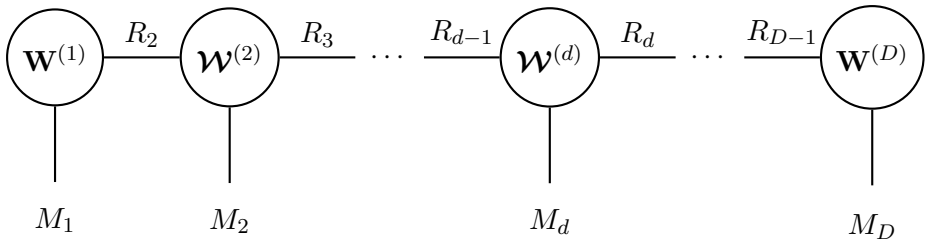


Figure 1-4: Tensor Network representation of a Tensor Train with D cores, each with dimension $R_d \times M_d \times R_{d+1}$, resulting in a reconstructed tensor $\mathcal{W} \in \mathbb{R}^{M_1 M_2 \ldots M_D}$.

1-3 Existing Probabilistic Tensor Network Kernel Machines

Several tensor network kernel machine models exist that operate in a probabilistic setting. Xiong et al. employed tensor factorization for the completion of temporal relational incomplete data, specifically using a CP decomposition. They formulated a Maximum A Posterior (MAP) scheme for factor estimation, and use Gibbs sampling for posterior estimation, thus limiting scalability [54]. Similarly Rai et al. employed Gibbs sampling in a probabilistic tensor decomposition, using it to learn the factors of a CP decomposition [42]. Zhao et al. employed variational Bayesian inference to learn the factors of a CP decomposed model [57]. Konstantinidis et al. proposed the Structured Posterior Bayesian Tensor Network (SP-BTN) model, improving upon the previous CP based models by imposing a low-rank structure on the mean of the weight tensor, thereby reducing the parameter complexity. Additionally, they impose a Kronecker structure on the local covariances. While this model has been shown to achieve a high degree of accuracy on standard datasets, it is limited in uncertainty quantification by the diagonal structure of the covariance matrices.

Kilic and Batselier proposed a fully bayesian CP based model learned via mean field variational Bayes and incorporating ARD priors. This model achieves high accuracy, and incorporates uncertainty quantification at no extra cost compared to both deterministic and other probabilistic models. The incorporation of ARD priors reduces the complexity of hyperparameter selection, and is shown to be able to reduce the rank of a model to the true rank of the underlying data on synthetic data. This model achieves high prediction accuracy and reduces the computational strain compared to existing models. However, at large D, the model is sensitive to numerical issues. These issues originate from the construction of the CPD decomposed weights vector $\mathbf{w}$. Construction of the CP decomposed model requires performing D Hadamard products for each training iteration. This causes the elements of the factor matrices to scale according to $|a|^D$, where a is an element in the factor matrices and D is the dimension of the model.

Other tensor decompositions that do not rely on Hadamard products for their construction exist that are able to achieve comparable reductions in parameters to be learned, and have previously been shown to be able to perform efficient parameter learning. One such decomposition is the TT decomposition. Kernel machines with tensor train decomposed weights vectors $\mathbf{w}$ have previously been shown to reduce the computational complexity of traditional kernel machines in a similar way to the previously discussed CPD based models.

Three relevant TT based probabilistic models were identified. Hinrich and Mørup introduced a

probabilistic tensor train decomposition, with its parameters learned using variational Bayes, and estimating the TT-ranks via the ELBO [22]. Xu et al. introduced a similar TT decomposed model with sparsity inducing hyperpriors for automatic rank inference [55]. These model however, focus on learning from tensor type data. Mezen et al. introduced a tensor network approximation of a GP. This model learns the parameters of all but one of the cores via a standard alternating linear scheme. The last core is then learned via Bayesian inference in order to enable uncertainty quantification. As only one core is learned via Bayesian inference, this model cannot be considered to be a fully probabilistic model [36].

1-3-1 BTN-Kernel Machine

The most relevant of the discussed models is the BTN-Kernels model by Kilic et al. [27], due to the similarity between it and the proposed TT based model. The BTN-Kernel machine model by Kilic et al. [27] assumes a datamodel

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w} + \mathbf{e} \tag{1-6}$$

where $\mathbf{y} \in \mathbb{R}^N$ denotes the observations vector and $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \tau^{-1}\mathbf{I}_N)$ is a noise vector containing zero mean Gaussian noise. The feature map $\mathbf{\Phi} \in \mathbb{R}^{M^D \times N}$ is constructed such that its n-th column contains $\varphi(\mathbf{x}_n)$, and can be constructed as $\mathbf{\Phi} = \mathbf{\Phi}^{(1)} \odot \mathbf{\Phi}^{(2)} \odot \cdots \odot \mathbf{\Phi}^{(D)}$. The columns of the component matrices $\mathbf{\Phi}^{(d)}$ are the feature vectors $\varphi^{(d)}$, for all $d \in [1, D]$. Furthermore, the weight vector $\mathbf{w}$ is decomposed into a CPD form, allowing formulation of the likelihood of the observed outputs as

$$p(\mathbf{y}| \left\{\mathbf{W}^{(d)}\right\}_{d=1}^{D}, \tau) = \prod_{n=1}^{N} \mathcal{N}\left(y_n|\varphi(x_n)^\intercal \mathbf{w}, \tau^{-1}\right), \tag{1-7}$$

where τ denotes the noise precision. The elements in $\mathbf{w}$ are learned via mean field variational inference. The factor matrix rank R_d and feature dimensions M_d are hyperparameters that can be tuned to control the model complexity. Finding appropriate values for these hyperparameters can be complex and computationally expensive. In order to aid in finding values for these hyperparameters, sparsity inducing hyperpriors, or ARD priors are introduced. These ARD priors control the variance of the rows and columns of the factor matrices $\mathbf{W}^{(d)}$. This pushes unnecessary components towards zero, promoting simpler models during training without requiring prior knowledge about the complexity of the model. These ARD priors have been introduced by Neal [38] and MacKay [35], and have previously been shown to work for Bayesian CP decomposed models by Zhao et al. [57].

Chapter 2

Probabilistic Tensor Train Kernel Machines

In this chapter we will introduce the Bayesian Tensor Train Kernel Machine (BTTKM) model. This model employs a tensor train decomposed weights vector, as well as a Khatri-Rao structured feature map. The construction of the datamodel, and the likelihood of the observations are shown in Section 2-1. This section also introduces the noise precision parameter τ, and the precision parameters $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$, that are used as the ARD parameters. The prior distributions for all the parameters of the datamodel are also given in Section 2-1. Using all these parameters and variables, the log-joint distribution is derived in section 2-1-3. In Section 2-2, we describe mean field variational inference, which is the method of inference that is used to estimate the model parameters. Sections 2-2-1 to 2-2-4 derive the update equations for each parameter, with the updates for the tensor cores in Section 2-2-1. The updates for the ARD factors are shown in Sections 2-2-2 and 2-2-3. Lastly, update for the noise precision parameter τ in Section 2-2-4.

Section 2-3 gives more information about how this model is implemented, and gives information about how the different hyperparameters can be used for model tuning.

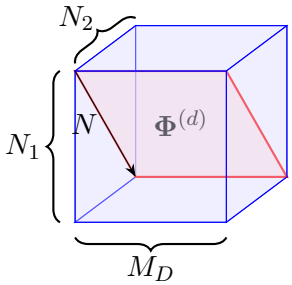
In order to evaluate the scalability of the proposed model, Section 2-4-1 reasons about the numerical robustness of the TT based model compared to the CPD based model by Kilic et al. [27]. And lastly, Section 2-4-2 gives the computational complexity of the training algorithm described.

2-1 Bayesian Tensor Train Kernel Machine Model

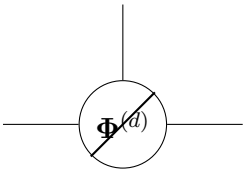
Much like the BTN-Kernels model by Afra Kilic, a BTTKM considers a datamodel

$$\mathbf{y} = \boldsymbol{\Phi}\mathbf{w} + \mathbf{e}, \tag{2-1}$$

Where $\mathbf{w}$ denotes the model weights vector, and $\mathbf{e}$ is a zero-mean Gaussian noise vector $\sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. The feature map $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M^D}$ is constructed such that its columns contain



(a) Visualisation of a diagonal three-way tensor, with values placed along the diagonal slice (shown in red) where $N_1 = N_2$.



(b) A tensor network diagram of diagonal tensor is shown as a node with a diagonal line.

Figure 2-1: Graphical representation of component matrix $\mathbf{\Phi}^{(d)}$ placed inside an $N_1 \times M_d \times N_2$ tensor, with values along the diagonal where $N_1 = N_2 = N$ and zeros elsewhere.

$\boldsymbol{\varphi}(\mathbf{x}_n)$. From the product form previously defined in Chapter 1-2, the matrix $\mathbf{\Phi}$ can be expressed as a series of row-wise Khatri-Rao products of component matrices

$$\mathbf{\Phi} = \mathbf{\Phi}^{(1)} \odot_R \mathbf{\Phi}^{(2)} \odot_R \cdots \odot_R \mathbf{\Phi}^{(D)}. \tag{2-2}$$

From here on, mainly the row-wise Khatri-Rao product will be used, so for brevity of notation, the row-wise prefix, as well as the $_R$ subscript will be dropped, and instead column-wise Khatri-Rao products will be explicitly referenced. The columns of the component matrices $\mathbf{\Phi}^{(d)} \in \mathbb{R}^{N \times M_d}$ contain the feature vectors $\boldsymbol{\varphi}$. In order to fit the component matrices of the feature map into the tensor train model, they must be transformed into three-way tensors. This is done by constructing a tensor in $\mathbb{R}^{N_1 \times M_d \times N_2}$ and placing the columns of $\mathbf{\Phi}^{(d)}$ along the columns where $N_1 = N_2$. Since this results in a tensor with values only along the diagonal slice where $N_1 = N_2$, these two separate indices are redundant, and the subscript can be dropped. This is shown graphically in Figure 2-1.

The weights vector $\mathbf{w}$ is decomposed into a tensor train composed of tensor cores $\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D} \in \mathbb{R}^{R_d \times M_d \times R_{d+1}}$. Combined with the feature map expressed as a TT, we can graphically express our model as a tensor network as in Figure 2-2.

Additionally, we can then express the likelihood of the observed data as a function of the decomposed weights vector. Assuming the Gaussian noise model from Equation (2-1), we can express the likelihood of the observed data as

$$p\left(\mathbf{y} \middle| \left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D}, \tau\right) = \prod_{n=1}^{N} \mathcal{N}\left(y_n | \boldsymbol{\varphi}(\mathbf{x}_n)^\intercal \mathbf{w}, \tau^{-1}\right), \tag{2-3}$$

where τ is a noise precision parameter.

The datamodel $\mathbf{\Phi w}$ can be constructed from its component tensors according to the *ziplining* technique. Figure 2-3 graphically shows the ziplining procedure.

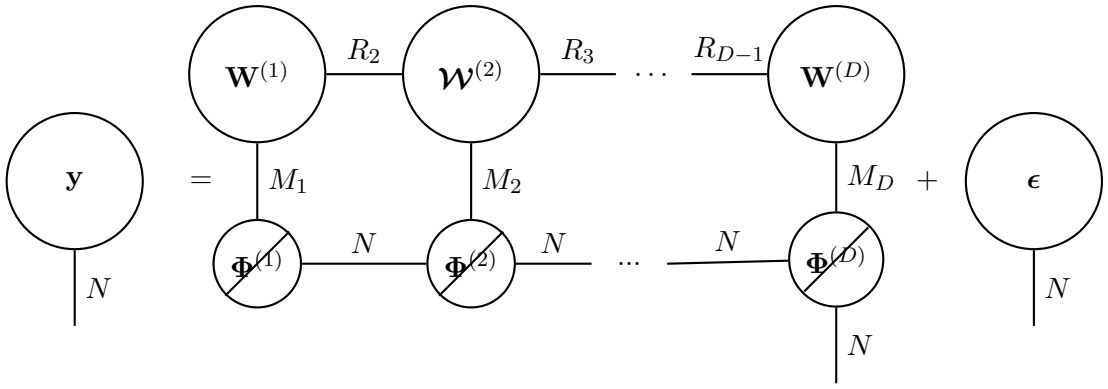


Figure 2-2: Graphical representation of the datamodel given in Equation (2-1). Due to the construction of the tensors $\mathbf{\Phi}^{(d)}$, $\mathbf{\Phi}^{(1)}$ can be thought of as a matrix, in order to maintain the correct dimensionality of the inner product $\mathbf{\Phi w}$.

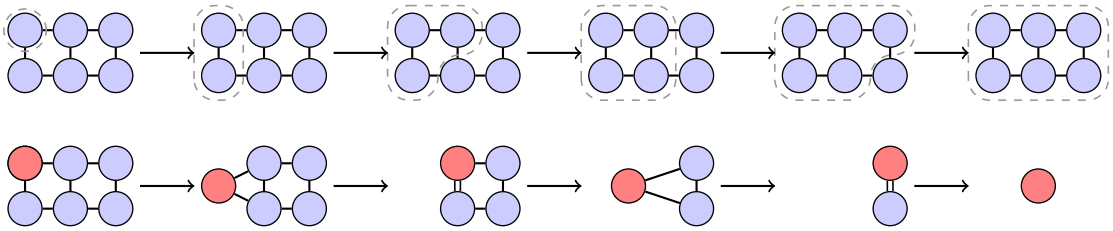


Figure 2-3: Ziplining technique for calculating the inner product between two vectors represented as tensor trains.

Starting from one side, we contract over the edge connecting the two trains, reshape the resulting tensor into a matrix, and absorb it into the next core of one of the TTs, before repeating for what is now the last core.

For the computation of the inner product between the feature map $\mathbf{\Phi}$ and the weights vector $\mathbf{w}$, we define a recursive accumulator $\mathbf{G}_{<d} \in \mathbb{R}^{N \times R_{d+1}}$ that collects the contributions of the TT-cores $\left\{\mathcal{W}^{(k)}\right\}_{k<d}$ and the component feature matrices $\left\{\mathbf{\Phi}^{(k)}\right\}_{k<d}$. From the definition of the feature map given in Equation (2-2), component matrix $\mathbf{\Phi}^{(k+1)}$ can be collected into the accumulator $\mathbf{G}_{<k}$ via a Khatri-Rao product. The TT-core $\mathcal{W}^{(k+1)} \in \mathbb{R}^{R_{k+1} \times M_{k+1} \times R_{k+2}}$ can be collected via a matrix multiplication with the mode-3 unfolded view $\mathbf{W}^{(k+1)}_{(3)} \in \mathbb{R}^{R_{k+2} \times R_{k+1} M_{k+1}}$. With initial condition $\mathbf{G}_{<1} := \mathbf{1} \in \mathbb{R}^N$, we define the recursive accumulator $\mathbf{G}_{<d}$ as

$$\mathbf{G}_{<k} := \left(\mathbf{G}_{<k-1} \odot \mathbf{\Phi}^{(k-1)}\right) \mathbf{W}^{(k-1)\intercal}_{(3)}, \quad k = 2, \ldots, d. \tag{2-4}$$

Where $\mathbf{W}^{(k)}_{(3)} \in \mathbb{R}^{R_{k+1} \times (R_k M_k)}$ is the mode-3 unfolded view as defined in Chapter 1-2.

The same procedure can be performed from right to left. Starting at $d = D$ with initial condition $\mathbf{G}_{>D} := \mathbf{1} \in \mathbb{R}^N$ we can define $\mathbf{G}_{>d} \in \mathbb{R}^{N \times R_d}$ as

$$\mathbf{G}_{>k} := \left(\mathbf{G}_{>k+1} \odot \mathbf{\Phi}^{(k+1)}\right) \mathbf{W}^{(k+1)\intercal}_{(1)}, \quad k = D-1, \ldots, d, \tag{2-5}$$

where $\mathbf{W}^{(k)}_{(1)} \in \mathbb{R}^{R_k \times (R_{k+1} M_d)}$ is constructed by permuting the dimensions of $\mathcal{W}^{(k)}$ before performing a mode-1 unfolding.

2-1-1 ARD Factors

As mentioned previously, the hyperparameter tuning of the feature dimensions M_d and TT-ranks R_d can be complex and computationally expensive to perform. To aid in this process, we introduce sparsity-inducing hierarchical priors over the tensor cores $\{\boldsymbol{\mathcal{W}}^{(d)}\}_{d=1}^{D}$. Two sets of precision parameters are introduced for each core $\boldsymbol{\mathcal{W}}^{(d)}$, $\boldsymbol{\delta}_d \in \mathbb{R}^{M_d}$ and $\boldsymbol{\lambda}_d \in \mathbb{R}^{R_d}$, governing the precision of the three modes of the tensor cores $\{\boldsymbol{\mathcal{W}}^{(d)}\}_{d=1}^{D}$. The first set of parameters $\boldsymbol{\lambda}_d$ governs the TT-ranks R_d. Due to the shared nature of the TT-ranks, each $\boldsymbol{\lambda}_d$ has an influence on two consecutive cores. We define $\boldsymbol{\lambda}_d := [\lambda_{1_d}, \lambda_{2_d}, \dots, \lambda_{R_d}]$. Each factor $\lambda_{r_d} \in \boldsymbol{\lambda}_d$ governs the strength of the frontal slice (see Figure 1-2c) of the d-th core, as well as the horizontal slice (see Figure 1-2a) of the $(d+1)$-th core. Due to the boundary condition that $R_1 = R_{D+1} = 1$, the $\boldsymbol{\lambda}_d$ parameter runs from $d = 2$ to $d = D$. The second set of parameters $\boldsymbol{\delta}_d$ are related to the feature dimension M_d. We define $\boldsymbol{\delta}_d := [\delta_{1_d}, \delta_{2_d}, \dots, \delta_{M_d}]$, where each δ_{m_d} regulates the strength of the m-th lateral slice (see Figure 1-2b) of the d-th core. A larger λ_{r_d} or δ_{r_d} value means that the r-th slice of the d-th core is less relevant, meaning it carries less predictive power. When the relevance of a specific slice falls below some boundary, it can be truncated in order to reduce the size and computational complexity of the model.

2-1-2 Prior Distributions

In order to enable us to employ a Bayesian learning technique to infer the parameters $[\boldsymbol{\mathcal{W}}^{(d)}, \boldsymbol{\lambda}_d, \boldsymbol{\delta}_d, \tau]$ and enable us to perform uncertainty quantification, we must define prior distributions over each of these parameters.

The prior of the TT-cores $\{\boldsymbol{\mathcal{W}}^{(d)}\}_{d=1}^{D}$ are defined for their vectorized view as a zero-mean Gaussian prior as

$$p(\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})|\boldsymbol{\lambda}_d, \boldsymbol{\delta}_d, \boldsymbol{\lambda}_{d+1}) = \mathcal{N}\left(\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})|\mathbf{0}, \boldsymbol{\Lambda}_{d+1}^{-1} \otimes \boldsymbol{\Delta}_d^{-1} \otimes \boldsymbol{\Lambda}_d^{-1}\right), \quad \forall\, d \in [1, D] \tag{2-6}$$

where $\boldsymbol{\Lambda}_d = \text{diag}(\boldsymbol{\lambda}_d)$ and $\boldsymbol{\Delta}_d = \text{diag}(\boldsymbol{\delta}_d)$, and the Kronecker product $(\boldsymbol{\Lambda}_{d+1}^{-1} \otimes \boldsymbol{\Delta}_d^{-1} \otimes \boldsymbol{\Lambda}_d^{-1})$ gives the covariance matrix.

For the sparsity parameters $\{\boldsymbol{\lambda}_d\}_{d=2}^{D-1}$, $\{\boldsymbol{\delta}_d\}_{d=1}^{D}$, gamma priors are chosen. These are chosen as it is a continuous non-negative distribution in the exponential family of probability distributions. These characteristics make it a suitable choice for a hierarchical hyperprior. The hyperprior over the rank sparsity parameter $\boldsymbol{\lambda}_d$ is factorized across the rank components as

$$p(\boldsymbol{\lambda}_d) = \prod_{r_d=1}^{R_d} \text{Ga}(\lambda_{r_d}|c_0, d_0), \tag{2-7}$$

while the feature sparsity parameter $\boldsymbol{\delta}_d$ has a hyperprior factorized over the feature dimensions

$$p(\boldsymbol{\delta}_d) = \prod_{m_d=1}^{M_d} \text{Ga}(\delta_{m_d}|g_0, h_0), \tag{2-8}$$

where $\text{Ga}(x|a, b) = b^a x^{a-1} e^{-bx}/\Gamma(a)$ is a Gamma distribution and $\Gamma(a) = \int_0^\infty e^{-t} t^{a-1} dt$ is a Gamma function.

Lastly, a Gamma hyperprior is placed over the noise precision parameter τ as

$$p(\tau) = \text{Ga}(\tau|a_0, b_0). \tag{2-9}$$

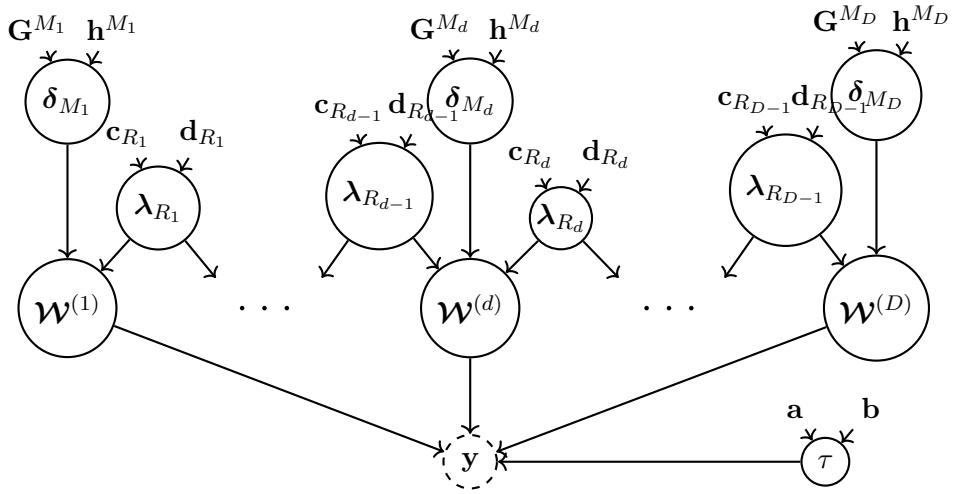


Figure 2-4: Representation of the TT decomposed Bayesian Kernel machine, showing the influences between parameters. Note how the parameters $\{\boldsymbol{\lambda}_R\}$ influence two consecutive $\boldsymbol{\mathcal{W}}$ nodes, not all cores are shown for simplicity.

2-1-3 Log Joint Distribution

Gathering all the latent variables and hyperparameters into the set

$$\Theta := \left\{ \left\{ \boldsymbol{\mathcal{W}}^{(d)} \right\}_{d=1}^{D}, \{\boldsymbol{\delta}_d\}_{d=1}^{D}, \{\boldsymbol{\lambda}_d\}_{d=2}^{D}, \tau \right\},$$

Figure 2-4 shows the probabilistic graph model from which we can derive the joint distribution $p(\mathbf{y}, \Theta)$ as

$$p(\mathbf{y}, \Theta) = p\left(\mathbf{y} | \left\{ \boldsymbol{\mathcal{W}}^{(d)} \right\}_{d=1}^{D}, \tau\right) \prod_{d=1}^{D} p\left(\boldsymbol{\mathcal{W}}^{(d)} | \boldsymbol{\lambda}_d, \boldsymbol{\delta}_d\right) p(\boldsymbol{\delta}_d) p(\boldsymbol{\lambda}_d) p(\tau). \tag{2-10}$$

Leading to the log-joint distribution

$$\begin{aligned} l(\Theta) =& \frac{\tau}{2} ||\mathbf{y} - \boldsymbol{\Phi}\mathbf{w}||_F^2 - \frac{1}{2} \sum_{d=1}^{D} \sum_{r_d=1}^{R_d} \sum_{m_d=1}^{M_d} \left(w_{r_d m_d r_{d+1}}^{(d)} \lambda_{r_d}^d \delta_{m_d}^d \lambda_{r_{d+1}}^d w_{r_d m_d r_{d+1}}^{(d)} \right) + \left(\frac{N}{2} + a_0 - 1 \right) \ln \tau - b_0 \tau \\ &+ \sum_{d=1}^{D} \sum_{r_d=1}^{R_d} \left(\left(\frac{M_d R_{d+1}}{2} + (C_0^{dr_d} - 1) \right) \ln \lambda_{r_d}^d + \frac{M_d R_d}{2} \ln \lambda_{r_{d+1}}^{d+1} - d_0^{dr_d} \lambda_{r_d}^d \right) \\ &+ \sum_{d=1}^{D} \sum_{m_d=1}^{M_d} \left(\left(\frac{R_d R_{d+1}}{2} + (g_0^{dr_r} - 1) \right) \ln \delta_{m_d}^d - h_0^{dm_d} \delta_{m_d}^d \right). \end{aligned} \tag{2-11}$$

The derivation of this distribution can be found in Appendix A-1.

2-2 Model Learning

From the joint distribution $p(\mathbf{y}, \Theta)$ we can find the full posterior distribution over all variables and the data using Bayesian inference. We can find the posterior distribution as

$$p(\Theta|\mathbf{y}) = \frac{p(\mathbf{y}, \Theta)}{\int p(\mathbf{y}, \Theta) d\Theta}. \tag{2-12}$$

Using this posterior distribution, we can find the predictive distribution over previously unseen data y_i^*, allowing us to predict the output of previously unseen inputs. The predictive distribution is defined as

$$p(y_i^*|\mathbf{y}) = \int p(y_i^*|\Theta) p(\Theta|\mathbf{y}) d\Theta. \tag{2-13}$$

Finding this distribution via exact Bayesian inference, as described above, would require explicit integration over all latent variables. Since the number of latent variables, especially for high dimensional data or for a high dimensional feature space can grow very large, this becomes infeasible.

Instead we approximate the posterior distribution $p(\Theta|\mathbf{y})$. In order to approximate the posterior distribution, mean-field variational inference [52] is used.

Within variational inference we aim to approximate the true (complex) posterior distribution $p(\Theta|\mathbf{y})$ with a (simpler) *variational distribution* $q(\Theta)$. We therefore want to find the best variational distribution, to most closely approximate the true distribution.

In order to optimize this variational distribution, we introduce the *Kullback-Leibler* (KL) divergence [31]

$$\mathrm{KL}(q(x)||p(x)) = \int q(x) \ln \frac{q(x)}{p(x)} dx = \mathbb{E}_{q(x)} \left[\ln \frac{q(x)}{p(x)} \right]. \tag{2-14}$$

This is an asymmetric measure of dissimilarity between two distributions. Logically, minimising this KL divergence, will cause the two distributions to be maximally similar. Substituting our true posterior and variational distributions into Equation 2-14 yields the KL divergence between the true posterior distribution $p(\Theta|\mathbf{y})$, and our approximation $q(\Theta)$. Formulated as

$$\mathrm{KL}(q(\Theta)||p(\Theta|\mathbf{y})) = \int q(\Theta) \ln \frac{q(\Theta)}{p(\Theta|\mathbf{y})} dx = \mathbb{E}_{q(\Theta)} \left[\ln \frac{q(\Theta)}{p(\Theta|\mathbf{y})} \right]. \tag{2-15}$$

This formulation of the KL divergence still relies on the unknown posterior distribution. As calculation of this distribution is still intractable, we rewrite the KL divergence into

$$\mathrm{KL}\left(q(\Theta)||p(\Theta|\mathbf{y})\right) = \ln p(\mathbf{y}) - \left(\mathbb{E}\left[\ln p(\Theta, \mathbf{y})\right] - \mathbb{E}\left[\ln q(\Theta)\right]\right). \tag{2-16}$$

Notice that this formulation no longer relies on the intractable posterior distribution. We are left with three distinct terms. The $\ln p(\mathbf{y})$ term, called the *evidence*, is solely dependent on the data and will thus remain constant. It can therefore be disregarded in minimising the KL divergence. The other two terms together are called the Evidence Lower Bound (ELBO), denoted as

$$\mathcal{L}(q) = \mathbb{E}\left[\ln p(\Theta, \mathbf{y})\right] - \mathbb{E}\left[\ln q(\Theta)\right]. \tag{2-17}$$

We can minimise the KL divergence, by maximizing this lower bound, without having to calculate the exact posterior distribution. From this formulation we can see that the maximum of the lower bound occurs when $q(\Theta) = p(\Theta|\mathbf{y})$. Which is to say that the variational distribution exactly coincides with the posterior distribution we aim to approximate.

In order to perform mean-field variational inference, we make the mean-field assumption that all the variables $\theta_j \in \Theta$ are independent. This means that the variational distribution $q(\Theta)$ factorizes across the variables in Θ as

$$q(\Theta) = \prod_{d=1}^{D} q_{\boldsymbol{\mathcal{W}}^{(d)}} q_{\boldsymbol{\delta}_d} q_{\boldsymbol{\lambda}_d} q_{\tau}. \tag{2-18}$$

It is important to note that this factorization assumption is the only assumption placed on the variational distributions. The specific functional forms $q_j(\theta_j)$ for each variable $\theta_j \in \Theta$ follow from the joint distribution given in Equation (2-11), and will be derived in turn in Chapters 2-2-1 to 2-2-4.

Because of their assumed independence, the posterior distribution of each variational distribution can be calculated in turn. The optimal solution for each $\theta_j \in \Theta$ occurs when the lower bound $\mathcal{L}(q)$ is at its maximum. This occurs when

$$\ln q_j(\theta_j) = \mathbb{E}_{q(\Theta \setminus \theta_j)}[\ln p(\mathbf{y}, \Theta)] + C, \tag{2-19}$$

where $\mathbb{E}_{q(\Theta \setminus \theta_j)}$ denotes the expectation taken with respect to all variables excluding θ_j. This means that, for a variable $\theta_j \in \Theta$, the expectation of the log-joint distribution, taken with respect to the variable θ_j is the optimal distribution of that variable θ_j. As this solution is optimal for one θ_j, given all other (possibly suboptimal) variables $\theta_{k \neq j}$, we must iteratively update each variable in Θ in turn, repeating this until some convergence criterion is reached, in order to move towards a more global optimum.

In the following sections, we will derive optimal posterior distributions for all the parameters in Θ in turn, using the log-joint distribution given in Equation (2-11), and the optimal lower bound given in Equation (2-19).

2-2-1 Posterior Distribution of Tensor Cores $\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D}$

As can be seen in Figure 2-4, the posterior update for each core $\boldsymbol{\mathcal{W}}^{(d)} \in \mathbb{R}^{R_d \times M_d \times R_{d+1}}$ for all $d \in [1, D]$ takes information from the observed data and related variables, namely all other factor matrices $\boldsymbol{\mathcal{W}}^{(k)}$ for $k \neq d$, and the noise precision parameter τ, as well as the sparsity parameters $\boldsymbol{\lambda}_d$, $\boldsymbol{\delta}_d$ and $\boldsymbol{\lambda}_{d+1}$. In order to collect the contribution of all other TT-cores, we define the matricized, partially constructed model $\mathbf{G}_{\setminus d} \in \mathbb{R}^{N \times (R_d M_d R_{d+1})}$. The construction of this employs the left hand and right hand accumulators $\mathbf{G}_{<d}$ and $\mathbf{G}_{>d}$ defined in Equations (2-5) and (2-4) to collect the contributions of all cores smaller than d and larger than d respectively. Combined with the component feature map $\boldsymbol{\Phi}^{(d)}$ corresponding to the d-th core, we formulate $\mathbf{G}_{\setminus d}$ as

$$\mathbf{G}_{\setminus d} = \mathbf{G}_{<d} \odot \boldsymbol{\Phi}^{(d)} \odot \mathbf{G}_{>d}. \tag{2-20}$$

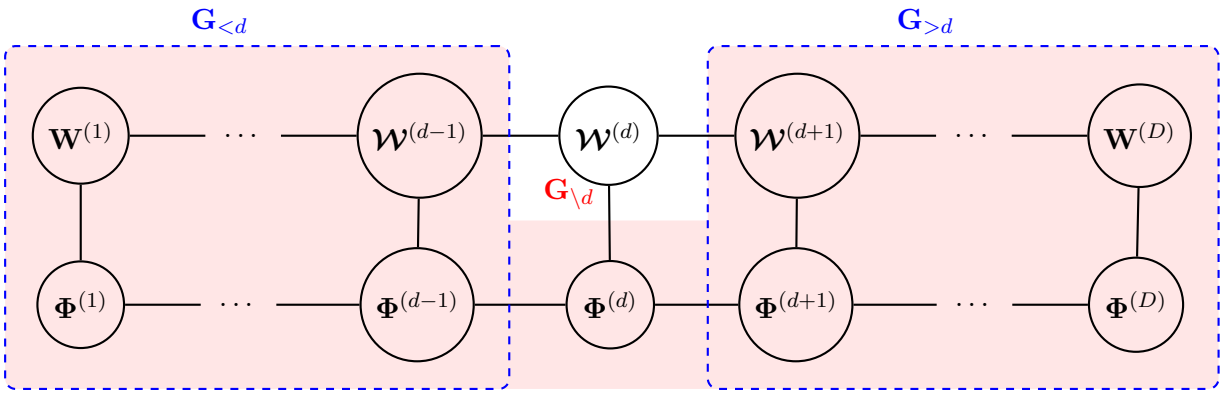


Figure 2-5: Partially constructed model, excluding $\mathcal{W}^{(d)}$, resulting in the $\mathbf{G}_{\setminus d}$ tensor. Constructed from the contribution of all cores preceding d, $\mathcal{W}^{(k)}$, $k < d$, and all cores following d, $\mathcal{W}^{(k)}$, $k > d$.

The construction of $\mathbf{G}_{\setminus d}$ is shown graphically in Figure 2-5. From the construction of $\mathbf{G}_{\setminus d}$, we can construct the full datamodel by incorporating the excluded d-th core into the datamodel, or by performing the recursion over all cores

$$\mathbf{\Phi w} = \text{vec}\left(\mathcal{W}^{(d)}\right)\mathbf{G}_{\setminus d} = \mathbf{G}_{<D+1} = \mathbf{G}_{>0}. \tag{2-21}$$

From the log-joint distribution given in Equation (2-11), combined with the optimal distribution from Equation (2-19), we can find the optimal posterior distribution of core $\mathcal{W}^{(d)}$. by collecting all terms of the joint distribution dependent on $\mathcal{W}^{(d)}$ and completing the square, we find the optimal posterior is a Gaussian distribution

$$q(\text{vec}(\mathcal{W}^{(d)})) = \mathcal{N}\left(\text{vec}(\mathcal{W}^{(d)})|\text{vec}(\tilde{\mathcal{W}}^{(d)}), \mathbf{\Sigma}^{(d)}\right). \tag{2-22}$$

With optimal mean and covariance calculated as

$$\begin{aligned}\mathbf{\Sigma}^{(d)} &= \left(\mathbb{E}_q[\tau]\mathbb{E}_q[\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}^\intercal] + \mathbb{E}_q[\mathbf{\Lambda}_{R_{d+1}}] \otimes \mathbb{E}_q[\mathbf{\Delta}_d] \otimes \mathbb{E}_q[\mathbf{\Lambda}_{R_d}]\right)^{-1}, \\ \text{vec}\left(\tilde{\mathcal{W}}^{(d)}\right) &= \mathbb{E}_q[\tau]\mathbf{\Sigma}^{(d)}\mathbb{E}_q\left[\mathbf{G}_{\setminus d}^\intercal\right]\mathbf{y},\end{aligned} \tag{2-23}$$

where $\tilde{\mathcal{W}}^{(d)} \in \mathbb{R}^{R_d \times M_d \times R_{d+1}}$ is the estimated mean of the d-th core, and $\mathbf{\Sigma}^{(d)} \in \mathbb{R}^{(R_d M_d R_{d+1}) \times (R_d M_d R_{d+1})}$ is the covariance matrix corresponding to the d-th core. Intuitively, the matrix $\mathbf{G}_{\setminus d} \in \mathbb{R}^{N \times R_d M_d R_{d+1}}$ can be interpreted as the design matrix corresponding to the d-th core $\mathcal{W}^{(d)}$. The design matrix, together with the covariance matrix and the noise precision term update the estimated mean of the core. The covariance matrix in turn is updated in part by the matrix $\mathbb{E}_q[\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}] \in \mathbb{R}^{(R_d M_d R_{d+1}) \times (R_d M_d R_{d+1})}$ representing the expected Gram matrix of the model, excluding core $\mathcal{W}^{(d)}$, and weighted by the noise precision τ. The other contribution comes from the hierarchical hyperpriors $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$. The derivation of these results is found in Appendix A-2.

The expected Gram matrix $\mathbb{E}_q[\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}]$ is not trivial to calculate directly. In order to calculate $\mathbb{E}_q[\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}]$ we define recursive accumulators $\mathbf{H}_{<d} \in \mathbb{R}^{N \times R_{d+1} R_{d+1}}$ and $\mathbf{H}_{>d} \in \mathbb{R}^{N \times R_d R_d}$ in a

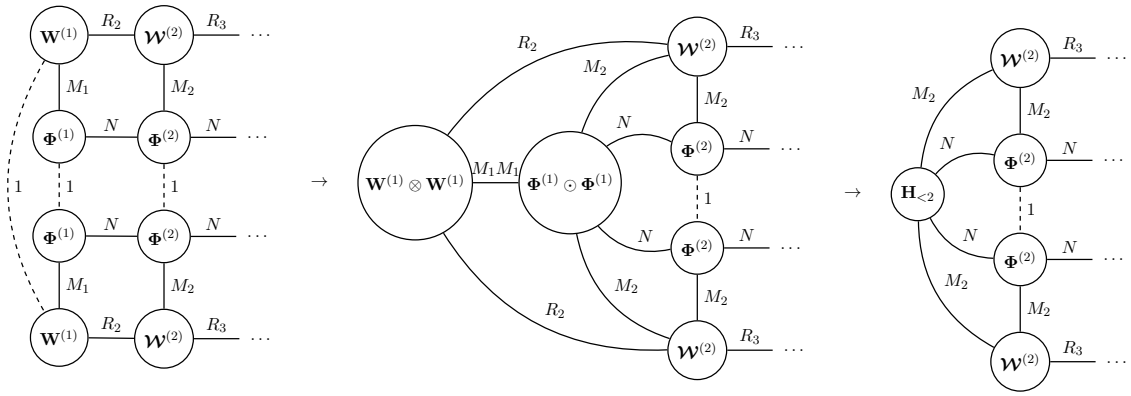


Figure 2-6: Construction of accumulator $\mathbf{H}_{<d}$. The two identical $\boldsymbol{\mathcal{W}}^{(1)}$ cores are contracted together by a Kronecker product, while the feature map component matrices $\mathbf{\Phi}^{(1)}$ are contracted via a Khatri-Rao product. The two resulting tensors are contracted by matrix multiplication over their unfoldings.

similar procedure to the ziplining technique. Starting from initial conditions $\mathbf{H}_{<1} := \mathbf{1} \in \mathbb{R}^N$ and $\mathbf{H}_{>D} := \mathbf{1} \in \mathbb{R}^N$ we define

$$\begin{aligned} \mathbf{H}_{<k} &= \left(\left(\mathbf{\Phi}^{(k-1)} \odot \mathbf{\Phi}^{(k-1)}\right) \odot \mathbf{H}_{<k-1}\right) \mathbb{E}_q\left[\mathbf{W}_{(3)}^{(k-1)\intercal} \otimes \mathbf{W}_{(3)}^{(k-1)\intercal}\right], \quad k = 2, \dots, d, \\ \mathbf{H}_{>k} &= \left(\left(\mathbf{\Phi}^{(k+1)} \odot \mathbf{\Phi}^{(k+1)}\right) \odot \mathbf{H}_{>k+1}\right) \mathbb{E}_q\left[\mathbf{W}_{(1)}^{(k+1)\intercal} \otimes \mathbf{W}_{(1)}^{(k+1)\intercal}\right], \quad k = D-1, \dots, d, \end{aligned} \tag{2-24}$$

with $\mathbf{W}_{(3)}^{(k)}$ and $\mathbf{W}_{(1)}^{(k)}$ as previously used in Equations (2-5) and (2-4) respectively. The different products in one step of this accumulator are shown graphically in Figure 2-6.

The expectation terms $\mathbb{E}_q\left[\mathbf{W}_{(3)}^{(k-1)\intercal} \otimes \mathbf{W}_{(3)}^{(k-1)\intercal}\right]$ and $\mathbb{E}_q\left[\mathbf{W}_{(1)}^{(k+1)\intercal} \otimes \mathbf{W}_{(1)}^{(k+1)\intercal}\right]$ can be calculated as

$$\mathbb{E}_q\left[\mathbf{W}_{(3)}^{(k)\intercal} \otimes \mathbf{W}_{(3)}^{(k)\intercal}\right] = \mathcal{R}\left\{\mathbb{E}_q\left[\mathbf{W}_{(3)}^{(k)\intercal}\right] \otimes \mathbb{E}_q\left[\mathbf{W}_{(3)}^{(k)\intercal}\right] + \mathbf{\Sigma}^{(k)}\right\}_{(R_d R_d M_d M_d)\times(R_{d+1}R_{d+1})} \tag{2-25}$$

and

$$\mathbb{E}_q\left[\mathbf{W}_{(1)}^{(k)\intercal} \otimes \mathbf{W}_{(1)}^{(k)\intercal}\right] = \mathcal{R}\left\{\mathbb{E}_q\left[\mathbf{W}_{(1)}^{(k)\intercal}\right] \otimes \mathbb{E}_q\left[\mathbf{W}_{(1)}^{(k)\intercal}\right] + \mathbf{\Sigma}^{(k)}\right\}_{(R_{d+1}R_{d+1}M_d M_d)\times(R_d R_d)}. \tag{2-26}$$

As shown in Appendix A-3. By employing the recursions $\mathbf{H}_{<d}$ and $\mathbf{H}_{>d}$, we can construct the full expected Gram matrix excluding the d-th core $\mathbb{E}_q\left[\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}^{\intercal}\right]$ as

$$\mathbb{E}_q\left[\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}^{\intercal}\right] = \mathcal{R}\left\{\left(\mathbf{H}_{<d} \odot \mathbf{\Phi}^{(d)}\right)^{\intercal}\left(\mathbf{\Phi}^{(d)} \odot \mathbf{H}_{>d}\right)\right\}_{(R_d M_d R_{d+1})\times(R_d M_d R_{d+1})}. \tag{2-27}$$

2-2-2 Posterior Distribution of $\boldsymbol{\lambda}_d$

As can be seen in Figure 2-4, the update for the $\boldsymbol{\lambda}_d$ parameter takes information from two subsequent cores $\boldsymbol{\mathcal{W}}^{(d-1)}$ and $\boldsymbol{\mathcal{W}}^{(d)}$, as well as its prior. We can calculate the optimal posterior distribution of the TT-rank precision parameter $\boldsymbol{\lambda}_d$ from the log-joint distribution given in Equation (2-11) and Equation 2-19. By gathering all terms of the log-joint distribution

dependent on $\boldsymbol{\lambda}_d$, we gather that the distributions for all $\boldsymbol{\lambda}_d,\ d \in [2, D-1]$ are independent Gamma distributions

$$q_{\boldsymbol{\lambda}_d}(\boldsymbol{\lambda}_d) = \prod_{r_d=1}^{R_d} \mathrm{Ga}(\lambda_{r_d} | c_N^{r_d}, d_N^{r_d}), \tag{2-28}$$

where $c_N^{r_d}$ and $d_N^{r_d}$ are the shape and rate parameter of the Gamma distribution, respectively. The parameters $c_N^{r_d}$ and $d_N^{r_d}$ are updated by

$$\begin{aligned}
\mathbf{c}_N^d &= \mathbf{c}_0^d + \frac{1}{2}(R_{d-1}M_{d-1} + M_d R_d), \\
\mathbf{d}_N^d &= \mathbf{d}_0^d + \frac{1}{2}\left(\mathbb{E}_q\left[\mathbf{W}_{(3)}^{(d-1)}(\boldsymbol{\Delta}_{d-1} \otimes \boldsymbol{\Lambda}_{d-1})\mathbf{W}_{(3)}^{(d-1)\intercal}\right] + \mathbb{E}_q\left[\mathbf{W}_{(1)}^{(d)}(\boldsymbol{\Lambda}_{d+1} \otimes \boldsymbol{\Delta}_d)\mathbf{W}_{(1)}^{(d)\intercal}\right]\right).
\end{aligned} \tag{2-29}$$

This means that the parameter $\mathbf{d}_N^d$, is updated by its prior, the L_2 norm of the $(d-1)$th core, weighted by $\boldsymbol{\delta}_{d-1}$ and $\boldsymbol{\lambda}_{d-1}$, and the L_2 norm of the d-th core, weighted by $\boldsymbol{\lambda}_{d+1}$ and $\boldsymbol{\delta}_d$. Derivations of these results can be found in Appendix A-4. The two expectation terms left in this result can be further simplified as explained in Appendix A-3. The first expectation term can be computed as

$$\begin{aligned}
\mathbb{E}_q\left[\mathbf{W}_{(3)}^{(d-1)}(\boldsymbol{\Delta}_{d-1} \otimes \boldsymbol{\Lambda}_{d-1})\mathbf{W}_{(3)}^{(d-1)\intercal}\right] =& \mathrm{diag}\left(\tilde{\mathbf{W}}_{(3)}^{(d-1)}(\boldsymbol{\Delta}_{d-1} \otimes \boldsymbol{\Lambda}_{d-1})\tilde{\mathbf{W}}_{(3)}^{(d-1\intercal)}\right) \\
&+ \mathbf{V}_{(3)}^{(d-1)}(\boldsymbol{\delta}_{d-1} \otimes \boldsymbol{\lambda}_{d-1}).
\end{aligned} \tag{2-30}$$

In order to reach this result we define a tensor

$$\boldsymbol{\mathcal{V}}^{(d)} := \mathcal{R}\left\{\mathrm{diag}(\Sigma^{(d)})\right\}_{R_d \times M_d \times R_{d+1}}, \tag{2-31}$$

which is to say $\boldsymbol{\mathcal{V}}^{(d)}$ contains the main diagonal of the covariance matrix Σ corresponding to the d-th TT core, reshaped into an $R_d \times M_d \times R_{d+1}$ tensor. The matrix $\mathbf{V}_{(3)}^{(d)} \in \mathbb{R}^{R_{d+1} \times (R_d M_d)}$ is the mode-3 unfolding of this tensor as described in Equation (1-4).

Similarly, the second expectation term can be calculated as

$$\begin{aligned}
\mathbb{E}_q\left[\mathbf{W}_{(1)}^{(d)}(\boldsymbol{\Lambda}_{d+1} \otimes \boldsymbol{\Delta}_d)\mathbf{W}_{(1)}^{(d)\intercal}\right] =& \mathrm{diag}\left(\tilde{\mathbf{W}}_{(1)}^{(d)}(\boldsymbol{\Lambda}_{d+1} \otimes \boldsymbol{\Delta}_d)\tilde{\mathbf{W}}_{(1)}^{(d)\intercal}\right) \\
&+ \mathbf{V}_{(1)}^{(d)}(\boldsymbol{\lambda}_{d+1} \otimes \boldsymbol{\delta}_d),
\end{aligned} \tag{2-32}$$

with a similar definition for $\mathbf{V}_{(1)} \in \mathbb{R}^{R_d \times M_d R_{d+1}}$.

2-2-3 Posterior Distribution of $\boldsymbol{\delta}_d$

As seen in the information model in Figure 2-4, the update of the feature dimension sparsity inducing posterior $\boldsymbol{\delta}_d$ takes information from the d-th TT core, as well as its hyperprior. From the optimal solution for the ELBO, given in Equation (2-19), we find that the posterior distribution of each $\boldsymbol{\delta}_d \in \boldsymbol{\delta}_d, d \in [1, D]$ by gathering all the terms in the log-joint distribution

given in Equation (2-11), that are dependent on $\boldsymbol{\delta}_d$. We find that the optimal posterior is an independent gamma distribution

$$q_{\boldsymbol{\delta}_d}(\boldsymbol{\delta}_d) = \prod_{m_d=1}^{M_d} \text{Ga}(\delta_{m_d} | g_N^{m_d}, h_N^{m_d}), \tag{2-33}$$

where the parameters $g_N^{m_d}, h_N^{m_d}$ are updated according to

$$\begin{aligned} \mathbf{g}_N^d &= \mathbf{g}_0^d + \frac{1}{2}(R_d R_{d+1}) \\ \mathbf{h}_N^d &= \mathbf{h}_0^d + \frac{1}{2}\mathbb{E}_q \left[\mathbf{W}_{(2)}^{(d)\intercal} (\boldsymbol{\Lambda}_{d+1} \otimes \boldsymbol{\Lambda}_d) \mathbf{W}_{(2)}^{(d)} \right]. \end{aligned} \tag{2-34}$$

Thus, the factor $\mathbf{h}_N^d$ is updated by its prior, plus the L_2 norm of the d-th TT-core, weighted by $\boldsymbol{\lambda}_d$ and $\boldsymbol{\lambda}_{d+1}$. Derivations of these equations can be found in Appendix A-5.

The expectation term of the inner product of the lateral slices can be computed similarly to the expectation terms in the previous section as

$$\begin{aligned} \mathbb{E}_q \left[\mathbf{W}_{(2)}^{(d)\intercal} (\boldsymbol{\Delta}_{d+1} \otimes \boldsymbol{\Lambda}_d) \mathbf{W}_{(2)}^{(d)} \right] =& \text{diag} \left(\tilde{\mathbf{W}}_{(2)}^{(d)\intercal} (\boldsymbol{\Lambda}_{d+1} \otimes \boldsymbol{\Lambda}_d) \tilde{\mathbf{W}}_{(2)}^{(d)} \right) \\ &+ \mathbf{V}_{(2)}^{(d)} (\boldsymbol{\lambda}_{d+1} \otimes \boldsymbol{\lambda}_d). \end{aligned} \tag{2-35}$$

The updates for the TT-ranks precision parameter $\boldsymbol{\lambda}_d$ and feature dimension precision parameter $\boldsymbol{\delta}_d$ given in Equations (2-28) and (2-33) respectively, result in a negative correlation between the two. This is illustrated using an example.

Consider the update for δ_{m_d}. This factor is updated by the L_2 norm of its corresponding core, weighted by the relevance of its horizontal and frontal slices. If the m-th lateral slice (the relevance of which is governed by δ_{m_d}) contains large non-zero entries, while the corresponding λ_{r_d} and $\lambda_{r_{d+1}}$ values are also large, these entries are heavily penalized by the large $\boldsymbol{\lambda}$ values. This will cause the rate parameter h_N^d, to also become large, in turn causing δ_{m_d} to become small.

This mechanism enables the model to compensate for penalization imposed by other factors, enabling it to preserve the influence of important slices, allowing for more flexibility. However, as the update for the precision variables $\boldsymbol{\lambda}_d$, is again dependent on the value of $\boldsymbol{\delta}_d$, the now shrunken $\boldsymbol{\delta}_d$ value causes the values of $\boldsymbol{\lambda}_d$ and $\boldsymbol{\lambda}_{d+1}$ to grow again. If left unchecked, this could cause the precision values to alternately grow and shrink unbounded. A method to prevent this is discussed in Section 2-3.

2-2-4 Posterior Distribution of Noise Precision

The noise precision parameter τ is inferred from information from the observed data and all factor matrices, as well as their hyperpriors, as can be seen in Figure 2-4. Using Equation (2-19), we can find the optimal posterior distribution of the noise precision parameter τ. By collecting the terms of the log-joint distribution given in Equation (2-11), that are dependent on τ, we can see that the optimal posterior distribution $q_\tau(\tau)$ is a Gamma distribution

$$q_\tau(\tau) = \text{Ga}(\tau | a_N, b_N). \tag{2-36}$$

With the parameters a_N and b_N updated by

$$\begin{aligned} a_N &= a_0 + \frac{N}{2} \\ b_N &= b_0 + \frac{1}{2}\mathbb{E}_q\left[||\mathbf{y} - \mathbf{\Phi w}||_F^2\right], \end{aligned} \tag{2-37}$$

as derived in Appendix A-6. The posterior expectation of the model residuals $\mathbb{E}_q\left[||\mathbf{y} - \mathbf{\Phi w}||_F^2\right]$ is constructed from several parts, as calculating it directly is challenging. The posterior expectation of the model residuals can be expanded to

$$\mathbb{E}_q\left[||\mathbf{y} - \mathbf{\Phi w}||_F^2\right] = ||\mathbf{y}||_F^2 - 2\mathbf{y}^\intercal \mathbb{E}_q\left[\mathbf{\Phi w}\right] + \mathbb{E}_q\left[(\mathbf{\Phi w})^\intercal(\mathbf{\Phi w})\right]. \tag{2-38}$$

The accumulator previously defined in Equation (2-5) can be used to substitute the term $\mathbf{\Phi w}$, as gathering the entire model, results in the fully constructed model

$$\mathbf{G}_{<D+1} = \mathbf{\Phi w} \in \mathbb{R}^N, \tag{2-39}$$

resulting in

$$\mathbb{E}_q\left[||\mathbf{y} - \mathbf{\Phi w}||_F^2\right] = ||\mathbf{y}||_F^2 - 2\mathbf{y}^\intercal \mathbb{E}_q\left[\mathbf{G}_{<D+1}\right] + \mathbb{E}_q\left[\mathbf{G}_{<D+1}^\intercal \mathbf{G}_{<D+1}\right]. \tag{2-40}$$

The last term can be substituted by the the $\mathbf{H}$ accumulator defined in Equation (2-24), without excluding any cores as

$$\mathbb{E}_q\left[\mathbf{G}_{<D+1}^\intercal \mathbf{G}_{<D+1}\right] = \mathbf{1}^\intercal \mathbb{E}_q\left[\mathbf{G}_{<D+1} \odot \mathbf{G}_{<D+1}\right] = \mathbf{H}_{<D+1}. \tag{2-41}$$

2-2-5 Lower Bound Model Evidence

The update equations presented above maximize the ELBO given in Equation (2-19), with respect to each variable in turn. Since the posterior distributions found by these update equations converge to a (local) maximum, the ELBO can be used as a convergence criteria. The expanded version of the ELBO given in Equation (2-17) is

$$\begin{aligned} \mathcal{L}(q) =& \mathbb{E}\left[||\mathbf{y} - \mathbf{\Phi w}||_F^2\right] - -\frac{1}{2}\mathrm{Tr}\left\{\sum_{d=1}^{D}\left(\tilde{\mathbf{\Lambda}}_{d+1} \otimes \tilde{\mathbf{\Delta}}_d \otimes \tilde{\mathbf{\Lambda}}_d\right)\left(\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)^\intercal + \mathbf{V}^{(d)}\right)\right\} \\ &+ \sum_{d=1}^{D}\left\{\sum_{r_d=1}^{R_d}\left\{\ln\Gamma(c_N^{r_d}) + c_N^{r_d}\left(1 - \ln d_N^{r_d} - \frac{d_0^{r_d}}{d_N^{r_d}}\right)\right\} + \sum_{m_d=1}^{M_d}\left\{\ln\Gamma(g_N^{m_d}) + g_N^{m_d}\left(1 - \ln h_N^{m_d} - \frac{h_0^{m_d}}{h_N^{m_d}}\right)\right\}\right\} \\ &+ \sum_{d=1}^{D}\left\{\frac{1}{2}\ln|\mathbf{V}^{(d)}|\right\} + \ln\Gamma(a_N) + a_N\left(1 - \ln b_N - \frac{b_0}{b_N}\right) + C. \end{aligned} \tag{2-42}$$

The residual error $\mathbf{E}_q\left[||\mathbf{y} - \mathbf{\Phi w}||_F^2\right]$ can be found using Equation (2-40). The first term captures the model residual, and penalizes predictions that differ from the dataset. The second term penalizes the L_2^2 norm of the tensor cores, including their uncertainty. The last terms measure the KL divergence between the posterior and prior distributions for all hyperparameters. Details of this derivation can be found in Appendix A-7.

2-2-6 Predictive Distribution

Having approximated the posterior distribution $p(\Theta|\mathbf{y})$, we can now find the predictive distribution over unseen inputs, given the training data $p(y_i^*|\mathbf{X}, \mathbf{y})$. Using the posterior distributions we have determined, we find

$$\begin{aligned} p(y_i^*|\mathbf{y}) &= \int p(y_i|\Theta)p(\Theta|\mathbf{y})d\Theta \\ &= \int\int p\left(y_i^*|\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}, \tau^{-1}\right) q\left(\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}\right) q(\tau) d\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\} d\tau. \end{aligned} \tag{2-43}$$

Substituting the posterior distributions found and integrating over $\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D}$ and τ yields a Student's t-distribution $p(y_i|\mathbf{y}) \sim \mathcal{T}(y_i^*, \mathcal{S}_i, \nu_y)$ with

$$\begin{aligned} y_i^* &= \boldsymbol{\Phi}\mathbf{w}, \qquad \nu_y = 2a_N, \\ \mathcal{S}_i &= \left\{\frac{b_N}{a_N}\sum_{d=1}^{D} \mathbf{G}^{(d)}(x_n)^\intercal \boldsymbol{\Sigma}^{(d)} \mathbf{G}^{(d)}(x_n)\right\}^{-1}. \end{aligned} \tag{2-44}$$

This results in a predictive variance of $\mathrm{Var}(y_i) = \frac{\nu_y}{\nu_y - 2}\mathcal{S}_i$. Derivations of these results can be found in Appendix A-8.

2-3 Implementation

Model initialization

Since mean field variational inference is only guaranteed to converge to a local minima, it is important to choose appropriate initializations for the model parameters to avoid getting stuck in local minima far removed from the global minimum. In order to promote convergence to a global minimum, *uninformative priors* are often chosen as initialization hyperparameters.

In general, all $c_0^d, d_0^d, g_0^d, h_0^d$ are initialized as 10^{-6} for all $d \in [1, D]$. This means that all λ_{r_d} and δ_{m_d} initialize to 1, while the prior will have little influence of the updates of the parameters $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$.

The factor τ is dependent on a_N and b_N. The a_N term in turn is dependent only on its prior a_0 and datasize N, as both of these are constant, a_N will also remain constant after the first update. Meanwhile, as b_N is dependent on the residual error, it tends to shrink during training. This would cause τ to grow very large during training, causing more and more weight to be given to the data, potentially causing the model to overfit and cause over or underflow. In order to avoid this, a_0 and b_0 are initialized slightly larger at 10^{-3}, which still causes τ to initialize to 1, but will prevent τ from shrinking to 0. Additionally, τ is kept constant during the training process and is only updated after the final full sweep of the model for the purpose of uncertainty quantification.

A smaller τ makes the priors more dominant in the training process, while increasing τ will increase the weight given to the likelihood during training. Larger initial values for $\boldsymbol{\delta}_d$ and $\boldsymbol{\lambda}_d$ more strongly promote sparsity in the tensor cores. As the values within a slice are shrunk by

the precision parameters, they may become so small that they are essentially 0. At this point the core can be truncated, in order to reduce the complexity of the model. In the current implementation, only TT-rank pruning is implemented. Once the variance explained of a certain slice falls below a threshold, set to 10^{-4}, the slice is removed. While it is currently not implemented for the feature dimensions M_d, this could also be implemented, so long as the corresponding row in the feature map is also removed. This would also require storing which rows are removed, as the feature map is recalculated when performing predictions over unseen data.

The TT-cores $\boldsymbol{\mathcal{W}}^{(d)}$ are initialized by drawing its elements from a standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, which can be scaled depending on the dataset in order to prevent numerical issues. The covariance matrices $\boldsymbol{\Sigma}^{(d)}$ are initialized as $\sigma^2\mathbf{I}$ with $\sigma^2 = 0.1$.

Learning Algorithm

During the learning process, updates are performed in order: TT-cores, feature dimension precision parameter, TT-rank precision parameter, as summarized in Algorithm 1. This is done in an order from most local effect to most global, as feature sparsity promoting factors only affect the corresponding core, while the rank sparsity promoting factors affect two consecutive cores. Performing the updates in this order increases stability, as a more stable input is provided to the more global updates.

In order to limit the computational load of one full sweep of updates, at the cost of memory usage, the backward accumulators $\mathbf{H}_{>d}$ and $\mathbf{G}_{>d}$ are precomputed and saved for all d before performing the updates of the TT-cores. The backward accumulators are initialized as a vector of ones, then for each successive core update, only one accumulator step needs to be performed.

In order to combat the boundless growth or shrinkage of sparsity parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\delta}$, they are lower bounded to 10^{-5}. Placing a lower bound on these parameters also prevents continuous growth, due to their inverse relationship.

2-4 Scalability

In order for the proposed BTTKM to be considered scalable, we must evaluate its robustness against over and underflow errors, as well as the computational complexity.

2-4-1 Over/Underflow Errors

In order to evaluate the robustness of the BTTKM model against finite precision related errors, upper and lower bounds on the magnitude of the tensor core updates were calculated. In order to perform this calculation, two assumptions have to be made. The first assumption is that all the columns of all the component feature matrices are normalized to have unit norms. The second assumption, is that both the mode-1 as mode-2 unfoldings of the TT-cores in the BTTKM model, as well as the factor matrices in the BTKM model, have identical singular values

$$\sigma(\mathbf{W}^{(1)}_{(1)}) = \sigma(\mathbf{W}^{(1)}_{(3)}) = \sigma(\mathbf{W}^{(2)}_{(1)}) = \cdots = \sigma(\mathbf{W}^{(D)}_{(1)}) = \sigma(\mathbf{W}^{(D)}_{(3)}) := \sigma(\mathbf{W}).$$

Algorithm 1 Learning algorithm

Require: input data $\mathbf{X} \in \mathbb{R}^{N \times D}$, output data $\mathbf{y} \in \mathbb{R}^N$
Initialize $\boldsymbol{\mathcal{W}}^{(d)}, \boldsymbol{\Sigma}^{(d)}, a_0, b_0, \mathbf{c}_0^{(d)}, \mathbf{d}_0^{(d)}, \mathbf{G}_0^{(d)}, \mathbf{h}_0^{(d)} \quad \forall d \in [1, D]$
set $\tau = a_0/b_0$, $\boldsymbol{\lambda}^{(d)} = \mathbf{c}_0^{(d)}/\mathbf{d}_0^{(d)}$ and $\boldsymbol{\delta} = \mathbf{G}_0^{(d)}/\mathbf{h}_0^{(d)} \quad \forall d \in [1, D]$
while not converged **do**
  calculate $[\mathbf{G}_{>1}, \mathbf{G}_{>2}, \ldots, \mathbf{G}_{>D}]$
  set $\mathbf{G}_{<d} = \mathbf{1} \in \mathbb{R}^N$
  for $d \in [1, D]$ **do**
    update posterior distribution $q\left(\text{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)\right)$ using Equation (2-22)
    set $\mathbf{G}_{<d}$ using Equation (2-5)
  end for
  for $d \in [1, D]$ **do**
    update posterior distribution $q(\boldsymbol{\delta}^{(d)})$ using Equation (2-33)
    if truncation criteria met **then**
      truncate M_d
    end if
  end for
  for $d \in [1, D]$ **do**
    update posterior distribution $q(\lambda^{(d)})$ using Equation (2-28)
    if truncation criteria met **then**
      truncate R_d
    end if
  end for
  Calculate ELBO via Equation (2-42)
  Check if convergence criteria reached
end while
update posterior distribution $q(\tau)$ using Equation (2-36)

From the formulation of the matrix $\mathbf{G}_{\setminus d}$ in the BTN-Kernels model

$$\mathbf{G}_{CPD}^{(d)} := \mathbf{\Phi}^{(d)} \odot \Big(\circledast_{k \neq d} \mathbf{W}^{(k)\intercal} \mathbf{\Phi}^{(k)} \Big), \tag{2-45}$$

we find that $\mathbf{G}_{CPD}^{(d)}$ is bounded by

$$\sigma_{\max}(\mathbf{W}^{(k)})^{D-1} \leq ||\mathbf{G}_m^{(d)}|| \leq \sigma_{\min}(\mathbf{W}^{(k)})^{D-1}. \tag{2-46}$$

Likewise, from the BTN-Kernels formulation for $\mathbb{E}_q \left[\mathbf{G}_{\setminus d} \mathbf{G}_{\setminus d}^{\intercal} \right]$

$$\mathbb{E}_q \left[\mathbf{G}_{\setminus d} \mathbf{G}_{\setminus d}^{\intercal} \right] = \left(\mathbf{\Phi}^{(d)} \odot \mathbf{\Phi}^{(d)} \right) \overset{D}{\underset{k \neq d}{\circledast}} \left(\mathbf{\Phi}^{(k)} \odot \mathbf{\Phi}^{(k)} \right)^{\intercal} \mathbb{E}_q \left[\mathbf{W}^{(k)} \otimes \mathbf{W}^{(k)} \right], \tag{2-47}$$

The bounds on $\mathbb{E}_q \left[\mathbf{G}_{\setminus d} \mathbf{G}_{\setminus d}^{\intercal} \right]$ can be calculated as

$$\sigma_{\min}(\mathbf{W})^{2(D-1)} \leq \mathbb{E}_q \big[\mathbf{G}^{(d)} \mathbf{G}^{(d)\intercal} \big] \leq \sigma_{\max}(\mathbf{W})^{2(D-1)}. \tag{2-48}$$

These result are derived in Appendix A-9. As all the matrices in these formulations are not necessarily square, there exists an edge condition where the columns of the component matrices of the feature map, lie within the null space of the factor matrix $\mathbf{W}^{(d)}$, or the unfolded core $\mathcal{W}^{(d)}$. From the formulations of $\mathbf{G}_{\setminus d}$ and $\mathbb{E}_q[\mathbf{G}_{\setminus d} \mathbf{G}_{\setminus d}^{\intercal}]$ we find the exact same bounds, suggesting that the behaviour is the same.

As the upper and lower bounds of the scaling are identical, we get no improvement on these bounds between a CPD model or a TT model. However, these bounds tell us nothing about how the cores will behave between those bounds, and the bounds only give us any information about possible over or underflow issues, when $\sigma_{\min}(\mathbf{W}) > 1$, or $\sigma_{\max}(\mathbf{W}) < 1$. Even when this is found to be the case for any particular core, the previously made assumption that these values will be identical for all cores, will not hold, giving us very little information about the behaviour of the cores in a TT-model. For a CPD model, we know that due to the repeated Hadamard product between the factor matrices, an element $a_{i,j}$ will scale roughly according to $||a_{i,j}||^D$, under the assumption that the value $a_{i,j}$ will be the same for the (i,j)th element of each factor matrix. While this assumption will also not hold in the real world, we can see that when all the elements in the (i,j)th position have a magnitude greater than one, the model will tend to suffer from overflow issues, and if the magnitude is less than one, it will lead to underflow issues, for sufficiently large D.

2-4-2 Computational Complexity

The update of the TT-cores is limited by the inversion of the covariance matrix update, which is an inversion of a $R_d M_d R_{d+1} \times R_d M_d R_{d+1}$ matrix, performed for D number of cores, as well as the matrix multiplications $\mathbf{\Sigma}^{(d)} \mathbb{E}_q \left[\mathbf{G}_{\setminus d} \right]^{\intercal} \mathbf{y}$. This results in a computational complexity of the order $\mathcal{O}(DM^3R^6 + DR^2MN)$, meaning that it is extremely important for our TT-ranks to be small, while it scales linearly with dimension D and data size N. The $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$ updates scale according to $\mathcal{O}(DM^2R^3)$ and $\mathcal{O}(DMR^4 + DM^2R^2)$. This yields a final complexity of $\mathcal{O}(DM^3R^6 + DR^2MN)$, in terms of data size this is significantly down with respect to datasize N from traditional kernel machines with complexity $\mathcal{O}(N^3)$. The high complexity has been

moved from this data dependent parameter N to a model dependent parameter R, which we can control and choose to be significantly smaller than N. This allows us to train models on significantly larger datasets both in term of dimension D as well as number of datapoints N, as long as we choose sufficiently small tensor ranks R_d.

Note this is seemingly a worse complexity than that of the BTN-Kernels model, which has a complexity of $\mathcal{O}(DNM^2R^2 + DM^3R^3)$ [27], however, it is also important to note that the number of parameters for the CPD based model scales according to $\mathcal{O}(DMR)$, while that of the TT based model scales roughly according to $\mathcal{O}(DMR^2)$, meaning that for a roughly equal number of model parameters, the complexity scaling will be equal.

These calculations exclude the computational cost of the reshape operations performed in training the BTTKM. While the `np.reshape` and `np.transpose` are themselves not very costly operations as they return a view of the tensor or matrix, performing operations with them is more costly as memory needs to be allocated.

Chapter 3

Experiments

Three sets of experiments were performed in order to evaluate the model's functioning and performance and compare it to other existing models. The first experiment investigates the ability of the ARD priors to infer the TT-ranks and feature dimensions of a synthetic ground truth datamodel with fewer TT-ranks and feature dimensions than the BTTKM model. The second experiment tested whether the BTTKM model was indeed less prone to over or underflow issues than the BTNKM model introduced by Kilic [27]. Finally, the model is compared to existing state-of-the-art models: T-KRR [49], SP-BTN [30], GP [43], and BTNKM [27] on a number of standard UCI datasets.

3-1 Feature Dimension and Tensor-Rank Recovery

3-1-1 Experimental Design

The first experiment was designed in order to validate that the ARD factors $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$ are able to find the true TT-ranks and feature dimensions of a ground truth datamodel. In order to test this, a synthetic ground truth datamodel is generated with TT-ranks R_d and feature dimensions M_d lower than the BTTKM we will train on it. In Figure, 3-1 the trained parameters $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$ are shown, with a clear difference in precision parameters.

A synthetic training dataset $\mathbf{X}_{train} \in \mathbb{R}^{N \times D}$ and a test dataset $\mathbf{X}_{test} \in \mathbb{R}^{N \times D}$ with $N = 100, D = 3$ was created by drawing elements from a standard normal distribution $\sim \mathcal{N}(0, 1)$. Training and validation labels were generated from them as $\mathbf{y} = \boldsymbol{\Phi}\mathbf{w} + \mathbf{e}$, where the feature map $\boldsymbol{\Phi}$ is created using using a polynomial feature mapping, and the noise term $\mathbf{e} \in \mathbb{R}^N$ is drawn from a Gaussian distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ with $\boldsymbol{\Sigma} = 0.001\mathbf{I}$.

The ground-truth datamodel $\mathbf{w}$ was created by constructing a tensor train with 3 cores with TT-ranks $\{R_d\}_{d=2}^{3}$ $\{M_d\}_{d=1}^{3}$ being chosen randomly between 2 and 4. For the feature map, a same number of columns is selected as M_d. The resulting ground truth datamodel dimensions are $R_d = [1, 3, 3, 1], \quad M_d = [4, 4, 3]$.

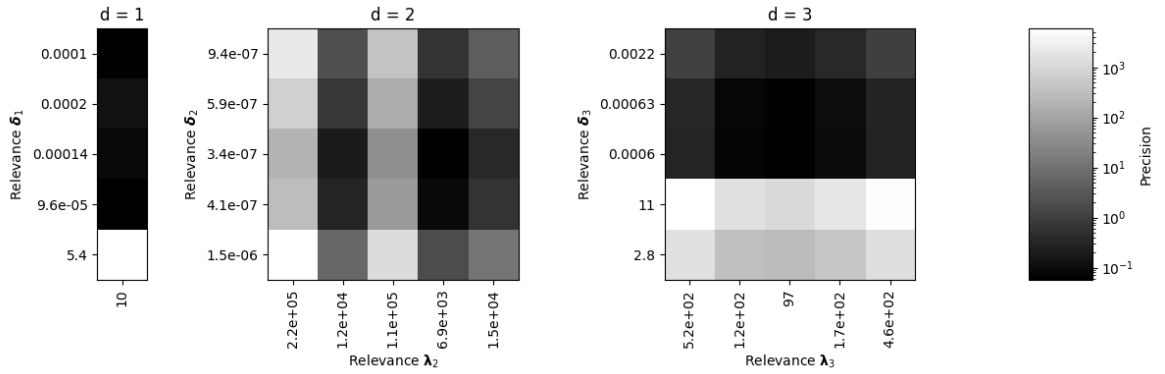


Figure 3-1: Posterior means of $\boldsymbol{\delta}$ and $\boldsymbol{\lambda}$ distributions, for cores $\boldsymbol{\mathcal{W}}^{(d)}$ for $d \in [1, 3]$. As the $\boldsymbol{\lambda}$ parameter is shared between consecutive cores, there are 2 $\boldsymbol{\lambda}$ distributions that carry any information, while $\boldsymbol{\delta}$ is unique for all three cores.

The elements of the cores were drawn from a normal distribution $\mathcal{N}(0, 10)$. The variance is set to 10 in order to increase the chance of non-zero elements. The labels $\mathbf{y} \in \mathbb{R}^N$ are calculated using the relationship (2-39) before the noise vector is added to it.

A larger prediction model with TT-ranks $R_{pred} = [1, 5, 5, 1]$ and feature ranks $M = [5, 5, 5]$ is initialized, by drawing its elements from a normal distribution $\sim \mathcal{N}(0, 1)$. The sparsity inducing hyperparameters $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$ are initialized with $c_0^{r_d}, g_0^{m_d} = 10^{-5}$ for all $c_0^{r_d} \in \mathbf{c}_0^d$, $g_0^{r_d} \in \mathbf{g}_0^d$ for all cores $d \in [1, D]$ and $d_0^{r_d}, h_0^{r_d} = 10^{-6}$ for all $d_0^{r_d} \in \mathbf{d}_0^d$, $h_0^{m_d} \in \mathbf{h}_0^d$ for all cores $d \in [1, D]$. This results in $\boldsymbol{\lambda}_d = \mathbf{10}$, $\boldsymbol{\delta}_d = \mathbf{10}$ $\forall d \in [1, D]$, these initialisations are chosen in order to promote sparsity. This model is then trained on the training set $\{\mathbf{X}_{train}, \mathbf{y}_{train}\}$. Figure 3-1 shows the results of this training process

3-1-2 Results

Figure 3-1 shows the outer product between $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$ for $d \in [1, 3]$. Since $R_1 = R_3 = 1$, this causes the first outer product to result in a vector, with the other two products taking the form of matrices. The final $\boldsymbol{\lambda}_4$ is also excluded $\boldsymbol{\mathcal{W}}^{(3)}$ only contains one frontal slice.

The resulting factors $\boldsymbol{\delta}_1$ shows extremely clear differentiation between relevant and irrelevant features with a difference of ~ 5 orders of magnitude between relevant and irrelevant features. In the third slice, the difference is slightly less, but still very clear with roughly 4 orders of magnitude difference between the relevant and irrelevant slices. For the middle slice, the difference is far less pronounced, with the value δ_5^2, corresponding to the irrelevant feature dimension, only being ~ 1.5 times larger than the relevant slice with the highest precision value.

The $\boldsymbol{\delta}_d$ values also show clear differentiation between relevant and irrelevant slices. Within $\boldsymbol{\lambda}_2$, the difference between relevant and irrelevant tensor ranks is around one order of magnitude. Within $\boldsymbol{\lambda}_3$, the difference is again less pronounced with the difference only being around 2.7 times.

These results indicate that the ARD factors $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$ are able to recover the true rank and feature dimension of ground truth datamodels. It is interesting to note that the difference

between relevant and irrelevant features can differ by quite a lot between cores. This is possibly due to inverse relationship between the factors $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$ described in Section 2-2-3. Where the ARD factors described in the BTN-Kernels model only need to compensate for the relevance of one other factor, the ARD factors in the BTTKM model must compensate for two other factors.

Another possible explanation is that the model converged with the effect of the moel dispersed over two slices. Due to the training data being made by a relatively simplistic datamodel, the BTTKM is able to approximate this model within only a few sweeps of posterior updates. As the relative difference of the TT-cores between sweeps is one of the convergence criteria used for early stopping of the training procedure, the model might have exited the training loop before the precision variables $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$ had fully converged.

3-2 Numerical Over and Underflow

3-2-1 Experimental Design

In order to test whether the TT-model is undeed less susceptible to over and underflow problems an experiment was designed where a TN-BKM model and a BTTKM model were trained on identical datasets, with identical initializations, with increasing data dimensionality.

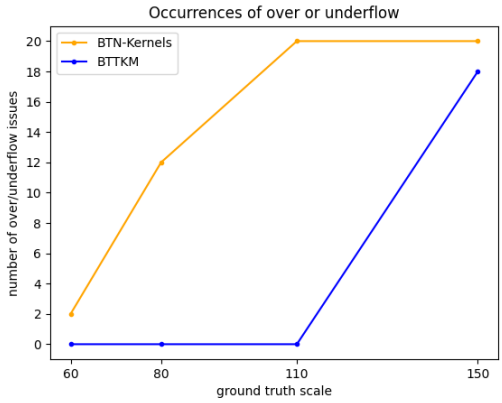
For each D in $[80, 100, 105, 110]$, a training set $\mathbf{X} \in \mathbb{R}^{N \times D}$ was created with $N = 100$ data points, by drawing from a normal distribution $\sim \mathcal{N}(0, 1)$.

These data points were then transformed into the feature space through the kernel function φ using a polynomial kernel with maximum power $M = 3$. The resultant feature map $\boldsymbol{\Phi}$, was vectorized and labels $\mathbf{y}$ were generated by taking the inner product of the vectorized feature map with a ground truth weights vector $\mathbf{w} \in \mathbb{R}^{DM}$, with its elements also drawn from a normal distribution $\sim \mathcal{N}(0, 1)$.

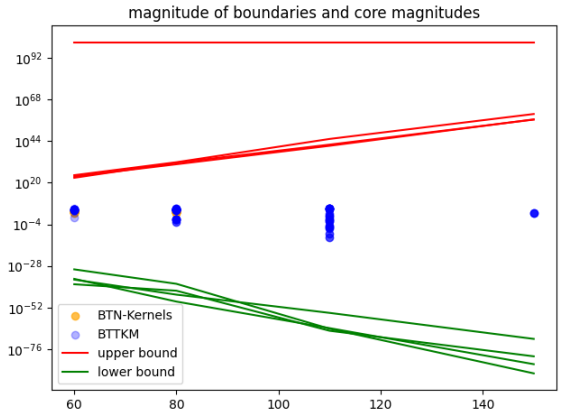
Due to the correspondence between the CPD and TT decomposition, the BTTKM was able to be initialized in the identical initial point. This was done by first initializing a CPD according to the description in chapter 3.2.5 of the 'Interpretable Bayesian Tensor Network Kernel Machines with Automatic Rank and Feature Selection' paper by Afra Kilic [27]. The first and last TT-cores were then initialized as the factor matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(1)}$, and intermediate TT-cores were initialized with the corresponding factor matrices placed on the diagonal where $R_d = R_{d+1}$.

Both models have a hyperparameter $\boldsymbol{\delta}_d$ unique to all the factor matrices or TT-cores respectively. This hyperparameter is characterized by the parameters $\mathbf{g}_0^d$ and $\mathbf{h}_0^d$. These are initialized identically to 10^{-6} for all $g_0^d \in \mathbf{g}_0^d$ and $h_0^d \in \mathbf{h}_0^d$ for all $d \in [1, D]$.

The hyperparameter $\boldsymbol{\lambda}$ is shared across all factor matrices in the BTN-Kernels model, while it is unique to each core in the BTTKM model. This parameter is initialized with $\lambda_0 = 10^{-6}$ for all $\lambda_0 \in \boldsymbol{\lambda}_0$ in the BTN-Kernels model, and $\lambda_0^d = 10^{-6}$ for all $\lambda_0^d \in \boldsymbol{\lambda}_0^d$ for all $d \in [1, D]$ in the BTTKM model.



(a) Number of occurrences of over and underflow issues for TN-BKM and BTTKM models, with $D = 80, 100, 105, 110$, showing that the BTTKM is more robust against these issues.



(b) Upper and lower bounds as calculated by Equation 2-48, with the magnitude of the constructed models staying within these bounds, when no over or underflow occurs.

Figure 3-2: Occurrences of over and underflow errors for models of varying sizes, as well as boundaries on their magnitude.

3-2-2 Results

As can be seen in Figure 3-2a, the number of occurrences of over and underflow issues grows gradually from 2 at $D = 60$ to 20 at $D = 110$ for the TT based BTN-Kernels model. Meanwhile, for the TT based BTTKM, no over or underflow issues occur until $D > 110$, after which the number of occurrences increases rapidly to 18 at $D = 150$. This suggests that the BTTKM is indeed somewhat more stable than the BTN-Kernels model. On top of this, Figure 3-2b shows that when no over or underflow issues occur, the magnitude of the constructed model stays within the calculated bounds. This figure also show that these bounds are quite loose, as even for the smallest models, the bounds span from 10^{-28} to 10^{20}, nearly 50 orders of magnitude. At larger dimensions D, over and underflow errors can occur within these bounds, as single precision floating point numbers can only represent numbers with a magnitude up to $\sim 10^{38}$, with a precision of $\sim 10^{-38}$. The precise mechanism that decides the magnitude of the elements within the bounds is still unknown, and requires further investigation in order to increase the stability further.

Almost all the over and underflow errors occurred during the first sweep of updates, pointing to the fact that the initial point is extremely important in preventing these issues. Due to this, it is not too computationally expensive to try several initial points for one itteration. A simple search algorithm might be to try some initial point, if an underflow error occurs within the first training sweep, scale all the elements in the initial cores $\{\mathcal{W}^{(d)}\}_{d=1}^{D}$ by some increment larger than one, and if an overflow error occurs, scale them by some increment smaller than one. If no error occurs, one can proceed with the rest of the training algorithm.

From these results we can see that while the BTTKM model is indeed somewhat more stable than the BTN-Kernels model, there is more research needed in order to improve the stability further.

3-3 UCI Datasets

3-3-1 Experimental Design

		BTTKM			BTN-Kernels [27]		
Dataset	M_{max}	R_{max}	a_0	c_0	R_{max}	a_0	c_0
Yacht	20	5	5	10^{-5}	25	110^{-2}	10^{-5}
Energy	20	5	0.1	10^{-5}	25	10^{-2}	10^{-5}
Concrete	20	5	10^{-2}	10^{-5}	25	10^{-3}	10^{-5}
Airfoil	20	5	10^{-2}	10^{-5}	25	10^{-3}	10^{-5}
spambase	30	5	0.1	10^{-5}	25	10^{-2}	10^{-5}
Adult	40	3	10^{-2}	10^{-6}	10	10^{-2}	10^{-6}

Table 3-1: Tuning parameters used for BTN-Kernels and BTTKM on different datasets. Some parameters are excluded as they are identical for every dataset.

The BTTKM model was tested on 6 UCI datasets in order to compare its performance against existing models. These datasets included 4 popular regression datasets: yacht ($N = 308$, $D = 6$) [15], energy ($N = 768$, $D = 9$) [53], concrete ($N = 1030$, $D = 8$) [56] and airfoil ($N = 1503$, $D = 5$) [5]. As well as 2 classification task: spambase ($N = 4601$, $D = 57$) [25] and the adult data set ($N = 48842$, $D = 96$) [3].

For each of these datasets, the data was split into 90% training data and 10% test data. For each dataset, 10 models were trained and evaluated, and their Root Mean Square Error (RMSE) (for regression) or their misclassification score (for classification), as well as their Negative Log-Likelihood (NLL) scores were recorded, and the covariance in their scores. For the Adult dataset, only one BTTKM model was trained, due to the long training time. For each dataset their features are normalized prior to training and evaluation, and for the classification tasks their labels were normalized as well. For the regression task, their labels were mapped to positive and negative one, and predictions were performed based on the sign of the model response.

The existing models that the BTTKM is compared to are: a tensorized GP model and a T-KRR model by Wesel and Batselier [49], the SP-BTN model by Konstantinidis et al. [30], and the BTN-Kernels model by Kilic. The results for the SP-BTN model are taken directly from their original paper, while all other scores are taken as reported by Kilic [27]. As the T-KRR model does not provide uncertainty quantification, it has no NLL score. While the SP-BTN model is able to perform uncertainty quantification, there are no NLL scores reported in their original paper. As their model is not publicly available, there is also no NLL score reported here for the SP-BTN model.

In order to compare these models to each other, all model uses a polynomial feature mapping with unit norm. For all regression models, the feature space dimensionality is set to $M = 20$. Two classification tasks have a much higher data dimensionality, therefore the feature dimensionality was increased to $M = 30$ for the spambase dataset, and $M = 40$ for the adult

	RMSE/Misclassification Rate					NLL		
Dataset	GP [49]	BTN-Kernels [27]	T-KRR [49]	SP-BTN [30]	BTTKM	GP	BTN-Kernels	BTTKM
Yacht	0.402 ± 0.131	0.379 ± 0.132	0.894 ± 0.478	0.506 ± 0.091	$\mathbf{0.121 \pm 0.039}$	0.105 ± 0.336	0.598 ± 0.782	$\mathbf{0.066 \pm 0.855}$
Energy	1.296 ± 0.290	0.456 ± 0.069	0.641 ± 0.135	0.549 ± 0.200	$\mathbf{0.243 \pm 0.062}$	1.680 ± 0.223	1.530 ± 0.271	$\mathbf{0.788 \pm 0.045}$
Concrete	5.565 ± 0.520	5.452 ± 1.242	4.959 ± 0.620	5.500 ± 0.230	$\mathbf{1.944 \pm 0.401}$	3.078 ± 0.080	3.387 ± 0.171	$\mathbf{2.59 \pm 0.060}$
Airfoil	2.293 ± 0.199	1.723 ± 0.151	1.806 ± 0.144	–	$\mathbf{0.555 \pm 0.055}$	2.281 ± 0.084	2.865 ± 0.152	$\mathbf{1.45 \pm 0.007}$
Spambase	0.095 ± 0.016	0.075 ± 0.015	$\mathbf{0.066 \pm 0.012}$	–	0.141 ± 0.215	0.720 ± 0.146	$\mathbf{0.499 \pm 0.047}$	0.868 ± 0.524
Adult	N/A	$\mathbf{0.144 \pm 0.005}$	0.1658 ± 0.007	–	0.153	N/A	$\mathbf{0.674 \pm 0.003}$	0.862

Table 3-2: Average RMSE or misclassification scores as well as NLL with standard deviation, best scores for a given dataset are shown in bold. For all the regression tasks, the BTTKM achieves the best scores, both in terms of RMSE and NLL. On the larger classification models, T-KRR and BTN-Kernels perform better.

dataset. For the T-KRR model whose scores are reported here, the rank is set to $R = 10$ for all datasets. The BTN-Kernels model uses an initial rank $R = 25$, which may get pruned during the training process. In order for the BTTKM model to have a roughly equal number of parameters, it is initialized with $R_d = 5$ for $d \in [2, D-1]$, as the the boundary condition prescribes $R_1 = R_D = 1$. Due to their respective structures, this results in an BTTKM model with slightly fewer initial parameters than the BTN-Kernels model. the BTN-Kernels model and BTTKM have very similar tuning parameters, with R_{max} setting the maximal feature matrix rank for BTN-Kernels and the TT-rank for the BTTKM, describing the complexity of the model, and M_{max} setting the maximum power for the polynomial kernel. The parameters a_0 and b_0 together decide the noise parameter τ, b_0 is set to 10^{-3} for all datasets, in order to be non-informative without being dominant in the posterior distribution. The values c_0 and d_0 are the shape and scale parameters of the Gamma prior over the sparsity inducing $\boldsymbol{\lambda}_d$ hyperprior, d_0 is set to 10^{-6} for all datasets to be uninformative. The hyperprior $\boldsymbol{\delta}_d$ is initialized with g_0 and h_0, both of which are set to 10^{-6} for both models for all datasets. The other variables M_{max}, R_{max}, a_0, c_0, differ per dataset. Table 3-1 shows the values used for all other hyperparameters.

3-3-2 Results

	BTN-Kernels [27]				BTTKM			
	nr. features	time $[s]$	mean RMSE	mean NLL	nr. features	time $[s]$	mean RMSE	mean NLL
Yacht	540	1.45	0.532	0.971	354	51	0.107	0.110
Concrete	800	4.39	5.522	3.38	1838	190	2.04	2.53
Airfoil	820	4.46	1.76	2.05	998	1662	0.562	1.57
Energy	1665	7.61	0.761	1.65	2912	232	0.312	0.789
Spambase	2970	328	0.137	0.361	15030	1464	0.141	0.868
Adult	3840	3670	0.144	0.674	33840	471029	0.153	0.863

Table 3-3: Diferences in timing between BTN-Kernels and BTTKM. While the BTTKM model is able to perform better in terms of RMSE and NLL, it does so at the cost of significantly longer training durations. The last column shows what the factor that the BTTKM is slower than the BTN-Kernels. All training times reported are for 10 repetitions, except for the Adult dataset.

Figure 3-3: Training durations of BTTKM and BTN-Kernels models plotted against number of features in the actual model.

As can be seen in table 3-2, the BTTKM model significantly outperforms the existing models on the four regression tasks, both in terms of RMSE and NLL. The BTTKM model consistently reaches an RMSE score 50% to 70% lower than the BTN-Kernels model and the T-KRR model, which in turn consistently outperform the GP model. In terms of NLL, the BTTKM model also far outperform both the BTN-Kernels model and the TT-decomposed GP model.

On the larger classification tasks however, the BTTKM model is not able to exceed the performance of existing models. Reaching an avarage missclassification rate almost twice as high for the spambase dataset, and landing in between the T-KRR model and BTN-Kernels model for the adult dataset. A part of this difference in performance might be attributed to differences in tuning parameters and model initialization. Which were found to be very influential in both training duration and final prediction performance.

Two other possible reasons for the increased performance on the regression tasks is the increased model freedom granted by the sparsity parameter $\boldsymbol{\lambda}_d$ being specific to each core, rather than shared across all factor matrices. Another possible explanation is a difference in the number of parameters in the trained model.

In order to further investigate the differences between the BTN-Kernels model, and the BTTKM model, the number of model parameters of the trained model and the training time for the training procedure was recorded. The results of this are summarized in table 3-3.

As can be seen in table 3-3, the current BTTKM takes significantly longer to train than the BTN-Kernels model, taking between $130 - 4$ times longer to train. At the same time, most of the trained BTTKM models have significantly more features than the trained BTN-Kernels model. Only for the Yacht dataset does the BTTKM model reduce the number of features in the prediction model further than the BTN-Kernels model. This is likely due to the less pronounced difference between relevant and irrelevant features in the sparsity inducing parameters $\boldsymbol{\lambda}_d$ and $\boldsymbol{\delta}_d$, in the BTTKM model than in the BTN-Kernels model. On op of this, the BTN-Kernels model, truncates columns of the factor matrices, of which there are 25, while the BTTKM model truncates slices of the TT-cores, of which there are only 5. This means that the BTN-Kernels model is able to prune ranks more aggressively, and earlier,

significantly speeding up the model training. With their respective complexities being roughly equal per trainable parameter. This accounts for a portion of the increase in computational time. On top of this, the current implementation of the BTTKM may not be optimal. Figure 3-3 shows how the training time for both the BTTKM model and the BTN-Kernels increases as a function of the total number of feature parameters. From this graph, we can see that while the training duration for the BTTKM is initially longer for small models, the training time grows slower for the BTTKM than for the BTN model. This means that there is a tipping point, at around ~ 2500 model parameters, that the BTTKM is faster to train than the BTN-Kernels model. Most of the increased training time for the BTTKM model can therefore be attributed to the larger number of model parameters, as well as a higher fixed cost. However, for the larger models, this increased number of parameters does not seem to translate to an increase in performance.

While the BTTKM is able to match or exceed the performance of existing Bayesian Tensor Network Kernel Machines, in its current form, the training procedure results in extremely large models, leading to extremely long training times. Further optimization of the current implementation, as well as research into better rank pruning is required to enable the model to be used on larger datasets.

Chapter 4

Conclusions

We have presented a Bayesian Tensor Train Kernel Machine (BTTKM), a fully bayesian tensor network kernel machine, employing a tensor train decomposition. Automatic relevance detection priors are used to automatically infer the true TT-rank and feature space dimensionality of an underlying data model, and are shown to be able to differentiate between relevant and irrelevant data. This ARD framework helps in reducing computational cost and aids in model interpretability. Employing the TT-decomposition makes the BTTKM somewhat less prone to over and underflow errors than existing models employing CP decompositions, although the exact mechanism by which this takes place needs more research. Experiments show that while the BTTKM model scales better with model size and is competitive with existing models in terms of predictive accuracy and uncertainty quantification, the number of model features that are needed to accomplish this is so large that training these larger models is costly.

At the start of this thesis, a research question was presented as

Can we construct a scalable fully Bayesian Tensor Train kernel machine including ARD priors that automatically infers both tensor ranks and feature dimensions?

With three subquestions formulated in order to aid in answering the main research question.

1. *Can ARD priors automatically infer TT-ranks R_d and feature dimensions M_d of a tensor train kernel machine for all $d \in [1, D]$ of a groundtruth datamodel?*
2. *Is a Bayesian Tensor Train (TT) kernel machine more robust to over and underflow issues than a comparable model using a CP decomposed model weights vector?*
3. *Can a Bayesian TT kernel machine achieve predictive accuracy and uncertainty quantification comparable to currently available models?*

We will now aim to answer these questions.

In Chapter 3-1, we have shown that the ARD priors are indeed able to differentiate between relevant and irrelevant feature dimensions and TT-ranks. However, the difference in magnitude between relevant slices and irrelevant slices, differs wildly between cores, this limits the our ability to prune irrelevant slices in the TT-cores, leading to extremely large models. On top of this, currently there is no feature dimension pruning implemented. Implementing this could further reduce the size, and with it computational complexity of the training procedure. The removal of irrelevant features could in turn also decrease the degree to which the sparsity parameter $\boldsymbol{\lambda}_d$ must compensate for variations in $\boldsymbol{\delta}_d$, perhaps increasing the ability to perform rank pruning. Thus further decreasing model size and training complexity.

Chapter 3-2 has shown that the BTTKM model is indeed more robust to numerical issues than the previous BTN-Kernels model. Although the increased robustness increases the scalability of the BTTKM model, under certain conditions, the proposed model does suffer from these issues, limiting the scalability of the model. A tuning mechanism to avoid this is proposed, although not tested. Further research is required to test this mechanism, as well as investigating the precise behaviour of the cores, within their magnitude bounds. Increased knowledge into this mechanism could aid in avoiding these issues.

Lastly, chapter 3-3 compares the predictive accuracy in terms of RMSE, misclassification rate, and NLL against existing models. From these experiments, we can see that the proposed model is capable of meeting and in many cases exceeding the predictive accuracy of existing models. This may in part be due to the fact that less rank pruning occurs in the training procedure of the BTTKM model than the BTN-Kernels model, resulting in a larger and more complex model. This increased model size is also reflected in the training duration for the proposed model, which far exceeds the training time for the BTN-Kernels model on the same datasets.

All in all, we can conclude that a fully Bayesian tensor train kernel machine with ARD priors, can be constructed, and the ARD priors are indeed able to infer the correct tensor ranks and feature dimensions. While the model displays increased robustness to numerical issues over the comparable BTN-Kernels model, it cannot be considered more scalable than existing models, due to the time required to train the model, with an increased number of model parameters.

4-1 Limitations and Future Work

A number of improvements can be made on this model in order to increase the scalability of the current model. Currently, the largest bottleneck is the time required for model training. The current implementation of the model is most likely far from optimal, and improvements can be made within the code to improve its efficiency. In the estimation of the computational complexity, the many reshape operations are neglected. These operations likely contribute heavily to the actual computational complexity of the current version of the model. Future research into reformulation of the model to not require performing as many reshape operations could greatly increase the speed of the training procedure of the current model. On top of this, further research into pruning of the TT-cores and feature dimensions, could help reduce the number of parameters after one or two sweeps of the posterior updates, and in turn reducing the training complexity.

If this was to be successful, the next greatest bottleneck is the remaining vulnerability to numerical over and underflow issues. Research into the exact mechanism behind this could aid in mitigating these issues. Furthermore, it was noted that the initialization of the model greatly determined the likelihood of the model to experience these errors. Further research into this, and a framework to automatically find appropriate initializations could already increase the scalability of the proposed BTTKM.

Appendix A

Appendix

A-1 Derivation of Log-Joint Distribution

We find the log joint distribution by taking the natural logarithm of the joint distribution

$$p\left(\mathbf{y},\Theta\right)=p\left(\mathbf{y}|\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D},\tau\right)\prod_{d=1}^{D}p\left(\boldsymbol{\mathcal{W}}^{(d)}|\boldsymbol{\lambda}_d,\boldsymbol{\delta}_d\right)p(\boldsymbol{\delta}_d)p(\boldsymbol{\lambda}_d)p(\tau).$$

Yielding

$$\ln(p\left(\mathbf{y},\Theta\right))=\ln\left(p\left(\mathbf{y}|\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D},\tau\right)\right)+\sum_{d=1}^{D}\ln\left(p\left(\boldsymbol{\mathcal{W}}^{(d)}|\boldsymbol{\lambda}_d,\boldsymbol{\delta}_d\right)\right)+\ln\left(p(\boldsymbol{\delta}_d)\right)+\ln\left(p(\boldsymbol{\lambda}_d)p(\tau)\right).$$

Substituting the probability density of the respective distributions gives the complete form

$$\begin{aligned}l(\Theta)=&\sum_{n=1}^{N}\left(\frac{1}{2}-\frac{\tau}{2}\left(y_n-\varphi(x_n)^{\intercal}\mathbf{w}\right)^2\right)+\sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d}\left(\frac{1}{2}\ln|\lambda_{r_d}^{d}\delta_{m_d}^{d}\lambda_{r_{d+1}}^{d}|-\frac{1}{2}\left(w_{r_d m_d r_{d+1}}^{(d)}\lambda_{r_d}^{d}\delta_{m_d}^{d}\lambda_{r_{d+1}}^{d}w_{r_d m_d r_{d+1}}^{(d)}\right)\right)\\&+\sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\left((c_0^{dr_d}-1)\ln\lambda_{r_d}^{d}-d_0^{dr_d}\lambda_{r_d}^{d}\right)+\sum_{d=1}^{D}\sum_{m_d=1}^{M_d}\left((g_0^{dm_d}-1)\ln\delta_{m_d}^{d}-g_0^{dm_d}\delta_{m_d}^{d}\right)+(a_0-1)\ln\tau-b_0\tau+\mathcal{C}.\end{aligned}$$

We can simplify this further by expanding the first two terms

$$
\begin{aligned}
l(\Theta) =& \frac{N}{2}\ln\tau - \frac{\tau}{2}\sum_{n=1}^{N}(y_n - \varphi(x_n)^\intercal \mathbf{w})^2 + \sum_{d=1}^{D}\left(\frac{M_d R_{d+1}}{2}\ln|\mathbf{\Lambda}_d| + \frac{R_d R_{d+1}}{2}\ln|\mathbf{\Delta}_d| + \frac{R_d M_d}{2}|\mathbf{\Lambda}_d|\right) \\
&- \frac{1}{2}\sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d}\left(w^{(d)}_{r_d m_d r_{d+1}}\lambda^d_{r_d}\delta^d_{m_d}\lambda^d_{r_{d+1}}w^{(d)}_{r_d m_d r_{d+1}}\right) + \sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\left((c_0^{dr_d}-1)\ln\lambda^d_{r_d} - d_0^{dr_d}\lambda^d_{r_d}\right) \\
&+ \sum_{d=1}^{D}\sum_{m_d=1}^{M_d}\left((g_0^{dm_d}-1)\ln\delta^d_{m_d} - g_0^{dm_d}\delta^d_{m_d}\right) + (a_0-1)\ln\tau - b_0\tau + \mathcal{C},
\end{aligned}
$$

and gathering like terms

$$
\begin{aligned}
l(\Theta) =& \frac{\tau}{2}||\mathbf{y}-\mathbf{\Phi}\mathbf{w}||_F^2 - \frac{1}{2}\sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d}\left(w^{(d)}_{r_d m_d r_{d+1}}\lambda^d_{r_d}\delta^d_{m_d}\lambda^d_{r_{d+1}}w^{(d)}_{r_d m_d r_{d+1}}\right) + \left(\frac{N}{2}+a_0-1\right)\ln\tau - b_0\tau \\
&+ \sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\left(\left(\frac{M_d R_{d+1}}{2} + (C_0^{dr_d}-1)\right)\ln\lambda^d_{r_d} + \frac{M_d R_d}{2}\ln\lambda^{d+1}_{r_{d+1}} - d_0^{dr_d}\lambda^d_{r_d}\right) \\
&+ \sum_{d=1}^{D}\sum_{m_d=1}^{M_d}\left(\left(\frac{R_d R_{d+1}}{2} + (g_0^{dr_r}-1)\right)\ln\delta^d_{m_d} - h_0^{dm_d}\delta^d_{m_d}\right)
\end{aligned}
\tag{A-1}
$$

A-2 Update Rule $\mathcal{W}^{(d)}$

From the local optimum of the ELBO with respect to the core $\boldsymbol{\mathcal{W}}^{(d)}$ in Equation 2-19, we find

$$
\begin{aligned}
\ln q\left(\text{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)\right) =& \mathbb{E}_{q(\Theta\setminus\boldsymbol{\mathcal{W}}^{(d)})}\left[\ln p(\mathbf{y},\Theta)\right] + C \\
=& \mathbb{E}\left[-\frac{\tau}{2}||\mathbf{y}-\mathbf{\Phi}\mathbf{w}||_F^2 - \frac{1}{2}\sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d} w^{(d)}_{r_d m_d r_{d-1}}\lambda^d_{r_d}\delta^d_{m_d}\lambda^{d+1}_{r_{d+1}}w^{(d)}_{r_d m_d r_{d-1}}\right] + C
\end{aligned}
$$

Summing over D, R_d and M_d

$$
= \mathbb{E}\left[-\frac{\tau}{2}(\mathbf{y}-\mathbf{\Phi}\mathbf{w})^2 - \frac{1}{2}\text{vec}(\boldsymbol{\mathcal{W}}^{(d)\intercal})(\mathbf{\Lambda}_{d+1}\otimes\mathbf{\Delta}_d\otimes\mathbf{\Lambda}_d)\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})\right] + C,
$$

and expanding the first term

$$
\begin{aligned}
=& \mathbb{E}\left[-\frac{\tau}{2}\left(\mathbf{y}-\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})\mathbf{G}_{\setminus d}\right)^2 - \frac{1}{2}\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})(\mathbf{\Lambda}_{d+1}\otimes\mathbf{\Delta}_d\otimes\mathbf{\Lambda}_d)\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})^\intercal\right] + C \\
=& \mathbb{E}\Big[-\frac{\tau}{2}\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})^\intercal\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}^\intercal\text{vec}(\boldsymbol{\mathcal{W}}^{(d)}) \\
&- \frac{1}{2}\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})^\intercal(\mathbf{\Lambda}_{d+1}\otimes\mathbf{\Delta}_d\otimes\mathbf{\Lambda}_d)\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})^\intercal + \tau\text{vec}(\boldsymbol{\mathcal{W}}^{(d)})\mathbf{G}_{\setminus d}^\intercal\mathbf{y}\Big] + C
\end{aligned}
$$

We find the log posterior distribution as

$$\begin{aligned}\ln q\left(\operatorname{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)\right) = &-\frac{1}{2}\operatorname{vec}(\boldsymbol{\mathcal{W}}^{(d)})^{\intercal}\left(\mathbb{E}[\tau]\mathbb{E}\left[\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}^{\intercal}\right]+\mathbb{E}[\boldsymbol{\Lambda}_{d+1}\otimes\boldsymbol{\Delta}_d\otimes\boldsymbol{\Lambda}_d]\right)\operatorname{vec}(\boldsymbol{\mathcal{W}}^{(d)})\\ &+\operatorname{vec}(\boldsymbol{\mathcal{W}}^{(d)})^{\intercal}\mathbb{E}[\tau]\mathbb{E}\left[\mathbf{G}_{\setminus d}^{\intercal}\right]\mathbf{y}+C\end{aligned} \tag{A-2}$$

From this form, we can see that the posterior distribution for the TT-cores is a Gaussian distribution. By completing the square, we can find the mean and covariance updated as

$$\begin{aligned}\operatorname{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right) =&\mathbb{E}_q[\tau]\boldsymbol{\Sigma}^{(d)}\mathbb{E}_q\left[\mathbf{G}_{\setminus d}^{\intercal}\right]\mathbf{y},\\ \boldsymbol{\Sigma}^{(d)} =&\left(\mathbb{E}_q[\tau]\mathbb{E}_q[\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}]^{\intercal}+\mathbb{E}_q[\boldsymbol{\Lambda}_{R_{d+1}}]\otimes\mathbb{E}_q[\boldsymbol{\Delta}_d]\otimes\mathbb{E}_q[\boldsymbol{\Lambda}_{R_d}]\right)^{-1}.\end{aligned} \tag{A-3}$$

A-3 Expectation of $\mathbf{W}\otimes\mathbf{W}$

$$\begin{aligned}\mathbb{E}\left[\mathbf{W}\otimes\mathbf{W}\right] &= \mathbb{E}\left[\operatorname{vec}\left(\mathbf{W}\right)\operatorname{vec}\left(\mathbf{W}\right)^{\intercal}\right]\\ &= \mathbb{E}\left[\operatorname{vec}\left(\mathbf{W}\right)\right]\mathbb{E}\left[\operatorname{vec}\left(\mathbf{W}\right)^{\intercal}\right]+\operatorname{Var}\left(\operatorname{vec}\left(\mathbf{W}\right)\operatorname{vec}\left(\mathbf{W}\right)^{\intercal}\right)\\ &= \mathbb{E}\left[\operatorname{vec}\left(\mathbf{W}\right)\right]\otimes\mathbb{E}\left[\operatorname{vec}\left(\mathbf{W}\right)\right]+\boldsymbol{\Sigma}\end{aligned} \tag{A-4}$$

A-4 Update Rule $\boldsymbol{\lambda}_d$

From Equation 2-19 we find

$$\begin{aligned}\ln q\left(\operatorname{vec}\left(\boldsymbol{\lambda}^{(d)}\right)\right) =&\mathbb{E}_{q(\Theta\setminus\boldsymbol{\lambda}^{(d)})}\left[\ln p(\mathbf{y},\Theta)\right]+C\\ =&\mathbb{E}\left[\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d} w^{(d)}_{r_d m_d r_{d-1}}\lambda^d_{r_d}\delta^d_{m_d}\lambda^d_{r_{d+1}}w^{(d)}_{r_d m_d r_{d-1}}+w^{(d-1)}_{r_{d-1}m_d r-1_d}\lambda^{d-1}_{r_{d-1}}\delta^{d-1}_{m_{d-1}}\lambda^d_{r_d}w^{(d)}_{r_d m_d r_{d-1}}\right.\end{aligned}.$$

We can simplify this by gathering like terms

$$\begin{aligned}&+\sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\left[\frac{M_d R_{d+1}}{2}+\frac{M_{d-1}R_{d-1}}{2}+(c_0^{dr_d}-1)\right]\ln\lambda^d_{r_d}-\sum_{r_d=1}^{R_d}d_0^{dr_d}\lambda^d_{r_d}\Bigg]+C\\ =&\mathbb{E}\Bigg[\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d}\left(w^{(d)}_{r_d m_d r_{d-1}}\delta^d_{m_d}\lambda^d_{r_{d+1}}w^{(d)}_{r_d m_d r_{d-1}}+w^{(d-1)}_{r_{d-1}m_d r-1_d}\lambda^{d-1}_{r_{d-1}}\delta^{d-1}_{m_{d-1}}w^{(d)}_{r_d m_d r_{d-1}}\right)\lambda^d_{r_d}\\ &+\sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\left[\frac{M_d R_{d+1}}{2}+\frac{M_{d-1}R_{d-1}}{2}+(c_0^{dr_d}-1)\right]\ln\lambda^d_{r_d}-\sum_{r_d=1}^{R_d}d_0^{dr_d}\lambda^d_{r_d}\Bigg]+C\\ =&\mathbb{E}\Bigg[\sum_{r_d=1}^{R_d}\left(\sum_{m_d=1}^{M_d}\left(w^{(d)}_{r_d m_d r_{d-1}}\delta^d_{m_d}\lambda^d_{r_{d+1}}w^{(d)}_{r_d m_d r_{d-1}}+w^{(d-1)}_{r_{d-1}m_d r-1_d}\lambda^{d-1}_{r_{d-1}}\delta^{d-1}_{m_{d-1}}w^{(d)}_{r_d m_d r_{d-1}}\right)-d_0^{dr_d}\right)\lambda^d_{r_d}\\ &+\sum_{r_d=1}^{R_d}\left[\frac{M_d R_{d+1}}{2}+\frac{M_{d-1}R_{d-1}}{2}+(c_0^{dr_d}-1)\right]\ln\lambda^d_{r_d}\Bigg]+C.\end{aligned}$$

This can then be shortened by summing over R_d and M_d

$$\begin{aligned}=&\mathbb{E}\Bigg[\left(\mathbf{W}_{(3)}^{(d)}\left(\boldsymbol{\Lambda}_{d+1}\otimes\boldsymbol{\Delta}_d\right)\mathbf{W}_{(3)}^{(d)}+\mathbf{W}_{(1)}^{(d-1)}\left(\boldsymbol{\Delta}_{d-1}\otimes\boldsymbol{\Lambda}_{d-1}\right)\mathbf{W}_{(1)}^{(d-1)}-\mathbf{d}_0^d\right)\boldsymbol{\lambda}^d\\&+\left[\frac{M_dR_{d+1}}{2}+\frac{M_{d-1}R_{d-1}}{2}+(\mathbf{c}_0^d-\mathbf{1})\right]\ln\boldsymbol{\lambda}^d\Bigg]+C.\end{aligned}$$

Finally, the expectation can be distributed, giving us

$$\begin{aligned}=&\left(\mathbb{E}\left[\mathbf{W}_{(3)}^{(d)}\left(\boldsymbol{\Lambda}_{d+1}\otimes\boldsymbol{\Delta}_d\right)\mathbf{W}_{(3)}^{(d)}\right]+\mathbb{E}\left[\mathbf{W}_{(1)}^{(d-1)}\left(\boldsymbol{\Delta}_{d-1}\otimes\boldsymbol{\Lambda}_{d-1}\right)\mathbf{W}_{(1)}^{(d-1)}\right]-\mathbf{d}_0^d\right)\boldsymbol{\lambda}^d\\&+\left(\frac{1}{2}(M_dR_{d+1}+M_{d-1}R_{d-1})+(\mathbf{c}_0^d-\mathbf{1})\right)\ln\boldsymbol{\lambda}^d+C\end{aligned}\tag{A-5}$$

From this it can be seen that the posterior is also a Gamma distribution, with updated parameters $\mathbf{c}$ and $\mathbf{d}$

$$\begin{aligned}\mathbf{c}_N^d&=\mathbf{c}_0^d+\frac{1}{2}(R_{d-1}M_{d-1}+M_dR_d),\\\mathbf{d}_N^d&=\mathbf{d}_0^d+\frac{1}{2}\left(\mathbb{E}_q\left[\mathbf{W}_{(3)}^{(d-1)}(\boldsymbol{\Delta}_{d-1}\otimes\boldsymbol{\Lambda}_{d-1})\mathbf{W}_{(3)}^{(d-1)\intercal}\right]+\mathbb{E}_q\left[\mathbf{W}_{(1)}^{(d)}(\boldsymbol{\Lambda}_{d+1}\otimes\boldsymbol{\Delta}_d)\mathbf{W}_{(1)}^{(d)\intercal}\right]\right).\end{aligned}\tag{A-6}$$

A-5 Update Rule $\boldsymbol{\delta}_d$

The update for the other sparsity parameter $\boldsymbol{\delta}_d$ can be found in a similar fashion to $\boldsymbol{\lambda}_d$. From Equation 2-19, we find

$$\begin{aligned}\ln q(\boldsymbol{\Delta}_d)=&\mathbb{E}_{q(\Theta\setminus\boldsymbol{\delta})}[\ln p(\mathbf{y},\Theta)]+C\\=&\mathbb{E}\Bigg[\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d}w_{r_dm_dr_{d-1}}^{(d)}\lambda_{r_d}^d\delta_{m_d}^d\lambda_{r_{d+1}}^d w_{r_dm_dr_{d-1}}^{(d)}+\sum_{m_d=1}^{M_d}\left(\frac{R_dR_{d+1}}{2}+(g_0^{dm_d}-1)\right)\ln\delta_{m_d}^d-\sum_{m_d=1}^{M_d}h_0^{dm_d}\delta_{m_d}^d\Bigg]+C.\end{aligned}$$

Which can be simplified by gathering like terms, summing over M_d and R_d, and distributing the expectation, giving

$$\begin{aligned}=&\mathbb{E}\left[-\frac{1}{2}\sum_{m_d=1}^{M_d}\left(\sum_{r_d=1}^{R_d}w_{r_dm_dr_{d-1}}^{(d)}\lambda_{r_d}^d\lambda_{r_{d+1}}^d w_{r_dm_dr_{d-1}}^{(d)}-h_0^{dm_d}\right)\delta_{m_d}^d+\sum_{m_d=1}^{M_d}\left(\frac{R_dR_{d+1}}{2}+g_0^{dm_d}-1\right)\ln\delta_{m_d}^d\right]+C\\=&\mathbb{E}\left[-\frac{1}{2}\left(\operatorname{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)^\intercal\left(\boldsymbol{\Lambda}_d\otimes\boldsymbol{\Lambda}_{d+1}\right)\operatorname{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)-\mathbf{h}_0^d\right)\boldsymbol{\delta}^d+\left(\frac{R_dR_{d+1}}{2}+\mathbf{G}_0^d-1\right)\ln\boldsymbol{\delta}^d\right]+C\\=&-\frac{1}{2}\left(\mathbb{E}\left[\operatorname{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)^\intercal\left(\boldsymbol{\Lambda}_d\otimes\boldsymbol{\Lambda}_{d+1}\right)\operatorname{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)\right]-\mathbf{h}_0^d\right)\boldsymbol{\delta}^d+\left(\frac{R_dR_{d+1}}{2}+\mathbf{G}_0^d-1\right)\ln\boldsymbol{\delta}^d+C.\end{aligned}$$

From this it can be seen that the posterior is also a Gamma distribution, with updated parameters $\mathbf{g}$ and $\mathbf{h}$.

$$\begin{aligned}\mathbf{g}_N^d&=\mathbf{g}_0^d+\frac{1}{2}(R_dR_{d+1})\\\mathbf{h}_N^d&=\mathbf{h}_0^d+\frac{1}{2}\mathbb{E}_q\left[\mathbf{W}_{(2)}^{(d)\intercal}(\boldsymbol{\Lambda}_{d+1}\otimes\boldsymbol{\Lambda}_d)\mathbf{W}_{(2)}^{(d)}\right].\end{aligned}\tag{A-7}$$

A-6 Update Rule τ

From equation 2-19, we find the optimal variational posterior distribution of the hyperparameter τ as

$$\begin{aligned}\ln q(\tau) &= \mathbb{E}_{q(\Theta\setminus\tau)}[\ln p(\mathbf{y}\Theta)] + C \\ &= \mathbb{E}\left[\frac{\tau}{2}||\mathbf{y}-\mathbf{\Phi w}||_F^2 + \left(\frac{N}{2}+a_0-1\right)\ln\tau - b_0\tau\right] + C.\end{aligned}$$

The term $||\mathbf{y}-\mathbf{\Phi w}||_F^2$, is the only distribution in this equation, it can therefore be rewritten as

$$\ln q(\tau) = \left(\frac{N}{2}-a_0-1\right)\ln\tau - b_0\tau + \mathbb{E}\left[\frac{\tau}{2}||\mathbf{y}-\mathbf{\Phi w}||_F^2\right] + C,$$

Which can be simplified by gathering like terms

$$\ln q(\tau) = \left(\frac{N}{2}-a_0-1\right)\ln\tau + \left(\mathbb{E}\left[\frac{1}{2}||\mathbf{y}-\mathbf{\Phi w}||_F^2\right] - b_0\right)\tau + C. \tag{A-8}$$

From this form, we can see that the variational posterior distribution of hyperparameter is a Gamma distribution, with shape and scale parameters a_N, b_N given as

$$\begin{aligned}a_N &= a_0 + \frac{N}{2} \\ b_N &= b_0 + \frac{1}{2}\mathbb{E}_q\left[||\mathbf{y}-\mathbf{\Phi w}||_F^2\right].\end{aligned} \tag{A-9}$$

A-7 Lower Bound Model Evidence

The Evidence Lower Bound (ELBO) given in equation 2-17 can be expanded as

$$\begin{aligned}\mathcal{L}(q) =& \mathbb{E}_{q(\Theta)}[\ln p(\mathbf{y},\Theta)] + H(q(\Theta)) \\ =& \mathbb{E}_{q(\{\boldsymbol{\mathcal{W}}^{(d)}\},\tau^{-1})}\left[\ln p(\mathbf{y}|\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D},\tau^{-1}\right] + \mathbb{E}_{q(\{\boldsymbol{\mathcal{W}}^{(d)}\},\{\boldsymbol{\lambda}_d\},\{\boldsymbol{\delta}_d\})}\left[\sum_{d=1}^{D}\ln p(\boldsymbol{\mathcal{W}}^{(d)}|\boldsymbol{\lambda}_d,\boldsymbol{\delta}_d,\boldsymbol{\lambda}_{d+1})\right] \\ &+ \mathbb{E}_{q(\{\boldsymbol{\lambda}\})}\left[\sum_{d=1}^{D}\ln p(\boldsymbol{\lambda}_d)\right] + \mathbb{E}_{q(\{\delta\})}\left[\sum_{d=1}^{D}\ln p(\boldsymbol{\delta}_d)\right] + \mathbb{E}_{q(\tau)}[p(\tau)] - \mathbb{E}_{q(\{\boldsymbol{\mathcal{W}}^{(d)}\})}\left[\sum_{d=1}^{D}\ln q(\boldsymbol{\mathcal{W}}^{(d)})\right] \\ &- \mathbb{E}_{q(\{\boldsymbol{\lambda}_d\})}\left[\sum_{d=1}^{D}\ln q(\boldsymbol{\lambda}_d)\right] - \mathbb{E}_{q(\{\boldsymbol{\delta}_d\})}\left[\sum_{d=1}^{D}\ln q(\boldsymbol{\delta}_d)\right] - \mathbb{E}_{q(\tau)}[\ln q(\tau)]\end{aligned} \tag{A-10}$$

Where all the expectations are taken with respect to the posterior distribution q. The first term represents the expected log-likelihood. The next four terms correspond to the expected log-priors over the set of parameters Θ. The final four terms represent the entropies of the posterior distributions of all parameters. Each term can be computed separately before combining them all.

A-7-1 Expected Log-Likelihood

The form of the expected log-likelihood follows from the form of the Gaussian and Gamma distributions of the tensor cores $\{\boldsymbol{\mathcal{W}}^{(d)}\}_{d=1}^{D}$ and

$$\begin{aligned}\mathbb{E}_{q(\{\boldsymbol{\mathcal{W}}^{(d)}\},\tau^{-1})}\left[\ln p(\mathbf{y}|\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D},\tau^{-1}\right] &= -\frac{N}{2}\ln(2\pi)+\frac{N}{2}\mathbb{E}_q[\ln\tau]-\frac{1}{2}\mathbb{E}\left[||\mathbf{y}-\boldsymbol{\Phi}\mathbf{w}||_F^2\right]\\ &= -\frac{N}{2}\ln(2\pi)+\frac{N}{2}\left(\psi(a_N)-\ln b_N\right)-\frac{a_N}{2b_N}-\frac{1}{2}\mathbb{E}[||\mathbf{y}-\boldsymbol{\Phi}\mathbf{w}||_F^2]\end{aligned} \tag{A-11}$$

Where N is the number of training points and $\psi(\cdot)$ denotes the digamma function $\psi(z)=\frac{d}{dz}\ln\Gamma(z)=\frac{\Gamma'(z)}{\Gamma(z)}$. The residual error $\mathbb{E}[||\mathbf{y}-\boldsymbol{\Phi}\mathbf{w}||_F^2]$ can be found using Equation (2-40).

A-7-2 Expected Log-Prior over $\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D}$

From the Gaussian distribution of the prior $p(\boldsymbol{\mathcal{W}}^{(d)}|\boldsymbol{\lambda}_d,\boldsymbol{\delta}_d,\boldsymbol{\lambda}_{d+1})$, we find the expected log-prior as

$$\begin{aligned}&\mathbb{E}_{q(\{\boldsymbol{\mathcal{W}}^{(d)}\},\{\boldsymbol{\lambda}_d\},\{\boldsymbol{\delta}_d\})}\left[\sum_{d=1}^{D}\ln p(\boldsymbol{\mathcal{W}}^{(d)}|\boldsymbol{\lambda}_d,\boldsymbol{\delta}_d,\boldsymbol{\lambda}_{d+1})\right]\\ &=\mathbb{E}_q\left[\sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d}\sum_{r_{d+1}}^{R_{d+1}}\left(-\frac{\ln 2\pi}{2}+\frac{1}{2}\left(\ln|\lambda_{r_d}^d\delta_{r_d}^d\lambda_{r_{d+1}}^{d+1}|\right)-\frac{1}{2}w_{r_d m_d r_{d+1}}^{(d)}\lambda_{r_d}^d\delta_{r_d}^d\lambda_{r_{d+1}}^d w_{r_d m_d r_{d+1}}^{(d)}\right)\right]\\ &=\mathbb{E}_q\Bigg[\sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d}\sum_{r_{d+1}}^{R_{d+1}}\Bigg(-\frac{\ln 2\pi}{2}+\frac{1}{2}\left(\ln|\lambda_{r_d}^d|+\ln|\delta_{r_d}^d|+\ln|\lambda_{r_{d+1}}^{d+1}|\right)\\ &\quad-\frac{1}{2}w_{r_d m_d r_{d+1}}^{(d)}\lambda_{r_d}^d\delta_{r_d}^d\lambda_{r_{d+1}}^d w_{r_d m_d r_{d+1}}^{(d)}\Bigg)\Bigg]\end{aligned}$$

Distributing the summations, and summing the final term over R_d, M_d and R_{d+1} we find

$$\begin{aligned}&=\sum_{d=1}^{D}\Bigg\{-\frac{R_d M_d R_{d+1}}{2}\ln(2\pi)+\frac{M_d R_{d+1}}{2}\sum_{r_d=1}^{R_d}\mathbb{E}_q\left[\ln\lambda_{r_d}^d\right]+\sum_{m_d=1}^{M_d}\frac{R_d R_{d+1}}{2}\mathbb{E}\left[\ln\delta_{m_d}^d\right]\\ &\quad+\frac{R_d M_d}{2}\sum_{r_{d+1}}^{R_{d+1}}\mathbb{E}_q\left[\ln\lambda_{r_{d+1}}^{d+1}\right]-\frac{1}{2}\mathbb{E}_q\left[\mathrm{vec}(\boldsymbol{\mathcal{W}}^{(d)})^{\intercal}(\boldsymbol{\Lambda}_{d+1}\otimes\boldsymbol{\Delta}_d\otimes\boldsymbol{\Lambda}_d)\mathrm{vec}(\boldsymbol{\mathcal{W}}^{(d)})\right]\Bigg\}\end{aligned}$$

The last term can then be expanded as

$$
\begin{aligned}
&= -\frac{\sum_{d=1}^{D} R_d M_d R_{d+1}}{2}\ln(2\pi) + \frac{\sum_{d=1}^{D} M_d R_{d+1}}{2}\sum_{r_d=1}^{R_d}\mathbb{E}_q\left[\ln\lambda_{r_d}^{d}\right] + \sum_{m_d=1}^{M_d}\frac{\sum_{d=1}^{D} R_d R_{d+1}}{2}\mathbb{E}\left[\ln\delta_{m_d}^{d}\right] \\
&\quad + \frac{\sum_{d=1}^{D} R_d M_d}{2}\sum_{r_{d+1}}^{R_{d+1}}\mathbb{E}_q\left[\ln\lambda_{r_{d+1}}^{d+1}\right] - \frac{1}{2}\sum_{d=1}^{D}\Bigg\{\mathrm{Tr}\left(\left(\tilde{\boldsymbol{\Lambda}}_{d+1}\otimes\tilde{\boldsymbol{\Delta}}_d\otimes\tilde{\boldsymbol{\Lambda}}_d\right)\mathrm{Var}\left(\mathrm{vec}(\boldsymbol{\mathcal{W}}^{(d)})\right)\right) \\
&\quad + \mathbb{E}_q\left[\mathrm{vec}(\boldsymbol{\mathcal{W}}^{(d)})^{\intercal}\right]\left(\mathbb{E}_q[\boldsymbol{\Lambda}_{d+1}]\otimes\mathbb{E}_q[\boldsymbol{\Delta}_d]\otimes\mathbb{E}_q[\boldsymbol{\Lambda}_d]\right)\mathbb{E}_q\left[\mathrm{vec}(\boldsymbol{\mathcal{W}}^{(d)})\right]\Bigg\} \\
&= -\frac{\sum_{d=1}^{D} R_d M_d R_{d+1}}{2}\ln(2\pi) + \frac{\sum_{d=1}^{D} M_d R_{d+1}}{2}\sum_{r_d=1}^{R_d}\mathbb{E}_q\left[\ln\lambda_{r_d}^{d}\right] + \sum_{m_d=1}^{M_d}\frac{\sum_{d=1}^{D} R_d R_{d+1}}{2}\mathbb{E}\left[\ln\delta_{m_d}^{d}\right] \\
&\quad + \frac{\sum_{d=1}^{D} R_d M_d}{2}\sum_{r_{d+1}}^{R_{d+1}}\mathbb{E}_q\left[\ln\lambda_{r_{d+1}}^{d+1}\right] - \frac{1}{2}\sum_{d=1}^{D}\left\{\mathrm{Tr}\left(\left(\tilde{\boldsymbol{\Lambda}}_{d+1}\otimes\tilde{\boldsymbol{\Delta}}_d\otimes\tilde{\boldsymbol{\Lambda}}_d\right)\mathbf{V}^{(d)}\right)\right\} \\
&\quad - \frac{1}{2}\sum_{d=1}^{D}\left\{\mathrm{Tr}\left(\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)^{\intercal}\left(\tilde{\boldsymbol{\Lambda}}_{d+1}\otimes\tilde{\boldsymbol{\Delta}}_d\otimes\tilde{\boldsymbol{\Lambda}}_d\right)\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)\right)\right\} \\
&= -\frac{\sum_{d=1}^{D} R_d M_d R_{d+1}}{2}\ln(2\pi) + \frac{\sum_{d=1}^{D} M_d R_{d+1}}{2}\sum_{r_d=1}^{R_d}\mathbb{E}_q\left[\ln\lambda_{r_d}^{d}\right] + \sum_{m_d=1}^{M_d}\frac{\sum_{d=1}^{D} R_d R_{d+1}}{2}\mathbb{E}\left[\ln\delta_{m_d}^{d}\right] \\
&\quad - \frac{1}{2}\mathrm{Tr}\left\{\sum_{d=1}^{D}\left(\tilde{\boldsymbol{\Lambda}}_{d+1}\otimes\tilde{\boldsymbol{\Delta}}_d\otimes\tilde{\boldsymbol{\Lambda}}_d\right)\left(\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)^{\intercal} + \mathbf{V}^{(d)}\right)\right\}
\end{aligned}
$$

The expectations of the hyperpriors λ, δ can then be expanded further as.

$$
\begin{aligned}
&= -\frac{\sum_{d=1}^{D} R_d M_d R_{d+1}}{2}\ln(2\pi) + \frac{\sum_{d=1}^{D} R_{d+1} M_d}{2}\sum_{r_d=1}^{R_d}(\psi(c_N^{r_d}) - \ln d_N^{r_d}) \\
&\quad + \frac{\sum_{d=1}^{D} R_{d+1} R_d}{2}\sum_{m_d=1}^{M_d}(\psi(g_N^{r_d}) - \ln h_N^{r_d}) + \frac{\sum_{d=1}^{D} M_d R_d}{2}\sum_{r_{d+1}=1}^{R_{d+1}}\left(\psi(c_N^{r_{d+1}}) - \ln d_N^{r_{d+1}}\right) \quad \text{(A-12)} \\
&\quad - \frac{1}{2}\mathrm{Tr}\left\{\sum_{d=1}^{D}\left(\tilde{\boldsymbol{\Lambda}}_{d+1}\otimes\tilde{\boldsymbol{\Delta}}_d\otimes\tilde{\boldsymbol{\Lambda}}_d\right)\left(\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)^{\intercal} + \mathbf{V}^{(d)}\right)\right\}.
\end{aligned}
$$

Due to the boundary conditions that $R_1 = R_{D+1} = 1$, the terms corresponding to r_{d+1} and r_d can be combined, resulting in

$$
\begin{aligned}
&= -\frac{\sum_{d=1}^{D} R_d M_d R_{d+1}}{2}\ln(2\pi) + \sum_{d=1}^{D} R_{d+1} M_d\sum_{r_d=1}^{R_d}(\psi(c_N^{r_d}) - \ln d_N^{r_d}) \\
&\quad + \frac{\sum_{d=1}^{D} R_{d+1} R_d}{2}\sum_{m_d=1}^{M_d}(\psi(g_N^{r_d}) - \ln h_N^{r_d}) \quad \text{(A-13)} \\
&\quad - \frac{1}{2}\mathrm{Tr}\left\{\sum_{d=1}^{D}\left(\tilde{\boldsymbol{\Lambda}}_{d+1}\otimes\tilde{\boldsymbol{\Delta}}_d\otimes\tilde{\boldsymbol{\Lambda}}_d\right)\left(\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)^{\intercal} + \mathbf{V}^{(d)}\right)\right\}.
\end{aligned}
$$

With $\mathbf{V}^{(d)} = \mathrm{Var}\left(\mathrm{vec}(\boldsymbol{\mathcal{W}}^{(d)})\right)$

A-7-3 Expected Log-Prior over $\{\boldsymbol{\lambda}_{r_d}\}_{d=1}^{D}$

From the Gamma form of the prior distribution $\boldsymbol{\lambda}$ it follows that

$$\begin{aligned}\mathbb{E}_q[\ln p(\boldsymbol{\lambda}_{r_d})] &= \sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\left\{\ln\Gamma(c_0^{r_d}) + c_0^{r_d}\ln d_0^{r_d} + (c_0^{r_d}-1)\mathbb{E}_q[\ln\lambda_{r_d}^d] - d_0^{r_d}\mathbb{E}_q[\lambda_{r_d}^d]\right\} \\ &= \sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\left\{\ln\Gamma(c_0^{r_d}) + c_0^{r_d}\ln d_0^{r_d} + (c_0^{r_d}-1)(\psi(c_N^{r_d}) - \ln d_N^{r_d}) - d_0^{r_d}\frac{c_N^{r_d}}{d_N^{r_d}}\right\}\end{aligned} \tag{A-14}$$

A-7-4 Expected Log-Prior over $\{\boldsymbol{\delta}_{r_d}\}_{d=1}^{D}$

From the Gamma form of the prior distribution $\boldsymbol{\delta}$ it follows that

$$\begin{aligned}\mathbb{E}_q[\ln p(\boldsymbol{\delta}_{m_d})] &= \sum_{d=1}^{D}\sum_{m_d=1}^{M_d}\left\{\ln\Gamma(g_0^{m_d}) + g_0^{m_d}\ln h_0^{m_d} + (g_0^{m_d}-1)\mathbb{E}_q[\ln\delta_{m_d}^d] - h_0^{m_d}\mathbb{E}_q[\delta_{m_d}^d]\right\} \\ &= \sum_{d=1}^{D}\sum_{m_d=1}^{M_d}\left\{\ln\Gamma(g_0^{m_d}) + g_0^{m_d}\ln h_0^{m_d} + (g_0^{m_d}-1)(\psi(g_N^{m_d}) - \ln h_N^{m_d}) - h_0^{m_d}\frac{g_N^{m_d}}{h_N^{m_d}}\right\}\end{aligned} \tag{A-15}$$

A-7-5 Expected Log-Prior over τ

From the Gamma form of the prior distribution τ it follows that

$$\begin{aligned}\mathbb{E}_q[\ln p(\tau)] &= -\ln\Gamma(a_0) + a_0\ln b_0 + (a_0-1)\mathbb{E}_q[\ln\tau] - b_0\mathbb{E}_1[\tau] \\ &= -\ln\Gamma(a_0) + a_0\ln b_0 + (a_0-1)(\psi(a_N) - \ln b_N) - b_0\frac{a_N}{b_N}\end{aligned} \tag{A-16}$$

A-7-6 Entropy of posterior distributions

From the Gaussian form of the variational posterior distribution over the tensor cores $q(\boldsymbol{\mathcal{W}}_{(d)})$

$$\begin{aligned}-\mathbb{E}\left[\sum_{d=1}^{D}\ln q(\boldsymbol{\mathcal{W}}^{(d)})\right] &= \sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\sum_{m_d=1}^{M_d}\sum_{r_{d+1}=1}^{D}\mathbb{E}_q\left[\ln q(w_{r_d m_d r_{d+1}}^{(d)})\right] \\ &= \sum_{d=1}^{D}\left\{\frac{1}{2}\ln|\mathbf{V}^{(d)}|\right\} + \frac{\sum_{d=1}^{D}R_{d+1}M_dR_d}{2}[1+\ln(2\pi)].\end{aligned} \tag{A-17}$$

While from the Gamma form of the variational posterior distributions $q(\boldsymbol{\lambda}_d)$, $q(\boldsymbol{\delta}_d)$, $q(\tau)$ we find

$$\begin{aligned}-\mathbb{E}_q[\ln q(\boldsymbol{\lambda}_d)] &= \sum_{d=1}^{D}\sum_{r_d=1}^{R_d}\{\ln\Gamma(c_N^{r_d}) - (c_N^{r_d}-1)\psi(c_N^{r_d}) - \ln d_N^{r_d} + c_N^{r_d}\} \\ -\mathbb{E}_q[\ln q(\boldsymbol{\delta}_d)] &= \sum_{d=1}^{D}\sum_{m_d=1}^{M_d}\{\ln\Gamma(g_N^{m_d}) - (g_N^{m_d}-1)\psi(g_N^{m_d}) - \ln h_N^{m_d} + g_N^{r_d}\} \\ -\mathbb{E}_q[\ln q(\tau)] &= \ln\Gamma(a_N) - (a_N-1)\psi(a_N) - \ln b_N + a_N\end{aligned} \tag{A-18}$$

The terms in Equations (A-11) to (A-18) can then all be combined into the evidence lower bound. Excluding all constants, we find

$$\begin{aligned}
\mathcal{L}(q) =& \frac{N}{2}\left(\psi(a_N) - \ln b_N\right) - \frac{a_N}{2b_N} - \frac{1}{2}\mathbb{E}[||\mathbf{y} - \mathbf{\Phi w}||_F^2] + \sum_{d=1}^{D} R_{d+1} M_d \sum_{r_d=1}^{R_d} \left(\psi(c_N^{r_d}) - \ln d_N^{r_d}\right) \\
&+ \frac{\sum_{d=1}^{D} R_{d+1} R_d}{2} \sum_{m_d=1}^{M_d} \left(\psi(g_N^{r_d}) - \ln h_N^{r_d}\right) + \sum_{d=1}^{D} \sum_{r_d=1}^{R_d} \sum_{m_d=1}^{M_d} \sum_{r_{d+1}=1}^{D} \mathbb{E}_q\left[\ln q(w^{(d)}_{r_d m_d r_{d+1}})\right] \\
&- \frac{1}{2}\mathrm{Tr}\left\{ \sum_{d=1}^{D} \left(\tilde{\mathbf{\Lambda}}_{d+1} \otimes \tilde{\mathbf{\Delta}}_d \otimes \tilde{\mathbf{\Lambda}}_d\right) \left(\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right) \mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)^{\mathsf{T}} + \mathbf{V}^{(d)}\right) \right\} \\
&+ \sum_{d=1}^{D} \sum_{r_d=1}^{R_d} \left\{ \ln\Gamma(c_0^{r_d}) + c_0^{r_d} \ln d_0^{r_d} + (c_0^{r_d} - 1)(\psi(c_N^{r_d}) - \ln d_N^{r_d}) - d_0^{r_d} \frac{c_N^{r_d}}{d_N^{r_d}} \right\} \\
&+ \sum_{d=1}^{D} \sum_{m_d=1}^{M_d} \left\{ \ln\Gamma(g_0^{m_d}) + g_0^{m_d} \ln h_0^{m_d} + (g_0^{m_d} - 1)(\psi(g_N^{m_d}) - \ln h_N^{m_d}) - h_0^{m_d} \frac{g_N^{m_d}}{h_N^{m_d}} \right\} \\
&+ \sum_{d=1}^{D} \sum_{r_d=1}^{R_d} \{\ln\Gamma(c_N^{r_d}) - (c_N^{r_d} - 1)\psi(c_N^{r_d}) - \ln d_N^{r_d} + c_N^{r_d}\} \\
&+ \sum_{d=1}^{D} \sum_{m_d=1}^{M_d} \{\ln\Gamma(g_N^{m_d}) - (g_N^{m_d} - 1)\psi(g_N^{m_d}) - \ln h_N^{m_d} + g_N^{r_d}\} + \ln\Gamma(a_N) - (a_N - 1)\psi(a_N) \\
&- \ln b_N + a_N - \ln\Gamma(a_0) + a_0 \ln b_0 + (a_0 - 1)(\psi(a_N) - \ln b_N) - b_0 \frac{a_N}{b_N}
\end{aligned}$$

Gathering like terms we find

$$\begin{aligned}
\mathcal{L}(q) =& \left(\frac{N}{2} + a_0 - a_N\right)\psi(a_N) - \left(\frac{N}{2} + a_0\right)\ln b_N - \frac{a_N}{2b_N} - \frac{1}{2}\mathbb{E}\left[||\mathbf{y} - \mathbf{\Phi w}||_F^2\right] \\
&- \frac{1}{2}\mathrm{Tr}\left\{ \sum_{d=1}^{D} \left(\tilde{\mathbf{\Lambda}}_{d+1} \otimes \tilde{\mathbf{\Delta}}_d \otimes \tilde{\mathbf{\Lambda}}_d\right) \left(\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right) \mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)^{\mathsf{T}} + \mathbf{V}^{(d)}\right) \right\} + \sum_{d=1}^{D} \left\{ \frac{1}{2} \ln |\mathbf{V}^{(d)}| \right\} \\
&+ \sum_{d=1}^{D} \left\{ \sum_{r_d=1}^{R_d} \left\{ (R_{d+1} M_d + c_0^{r_d} - c_N^{r_d})\, \psi(c_N^{r_d}) - (R_{d+1} M_d + c_0^{r_d}) \ln d_N^{r_d} \right\} \right. \\
&+ \sum_{m_d=1}^{M_d} \left\{ \left(\frac{R_{d+1} R_d}{2} + g_0^{m_d} - g_N^{r_d}\right) \psi(g_N^{r_d}) - \left(\frac{R_{d+1} R_d}{2} + g_0^{m_d}\right) \ln h_N^{m_d} \right\} \\
&+ \sum_{r_d=1}^{R_d} \left\{ \ln\Gamma(c_N^{r_d} + c_N^{r_d} - d_0^{r_d} \frac{c_N^{r_d}}{d_N^{r_d}} \right\} + \left. \sum_{m_d=1}^{M_d} \left\{ \ln\Gamma(g_N^{m_d} + g_N^{m_d} - h_o^{m_d} \frac{g_N^{m_d}}{h_N^{m_d}} \right\} \right\} \\
&- b_0 \frac{a_N}{b_N} + \ln\Gamma(a_n) + a_N + C.
\end{aligned}$$

Using the relationship that $\mathbb{E}[\ln q] = \psi(a) - \ln b$, where q is a digamma function with shape

and scale parameters a and b we find the distribution

$$\begin{aligned}\mathcal{L}(q) =&\mathbb{E}\left[||\mathbf{y}-\mathbf{\Phi w}||_F^2\right] - -\frac{1}{2}\mathrm{Tr}\left\{\sum_{d=1}^{D}\left(\tilde{\mathbf{\Lambda}}_{d+1}\otimes\tilde{\mathbf{\Delta}}_d\otimes\tilde{\mathbf{\Lambda}}_d\right)\left(\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right)^{\intercal}+\mathbf{V}^{(d)}\right)\right\}\\ &+\sum_{d=1}^{D}\left\{\sum_{r_d=1}^{R_d}\left\{\ln\Gamma(c_N^{r_d})+c_N^{r_d}\left(1-\ln d_N^{r_d}-\frac{d_0^{r_d}}{d_N^{r_d}}\right)\right\}+\sum_{m_d=1}^{M_d}\left\{\ln\Gamma(g_N^{m_d})+g_N^{m_d}\left(1-\ln h_N^{m_d}-\frac{h_0^{m_d}}{h_N^{m_d}}\right)\right\}\right\}\\ &+\sum_{d=1}^{D}\left\{\frac{1}{2}\ln|\mathbf{V}^{(d)}|\right\}+\ln\Gamma(a_N)+a_N\left(1-\ln b_N-\frac{b_0}{b_N}\right)+C.\end{aligned} \tag{A-19}$$

A-8 Derivation of Predictive Distribution

The predictive distribution over unseen datapoints

$$p(y_i^*|\mathbf{y}) = \int p(y^*|\Theta)p(\Theta|\mathbf{y})d\Theta,$$

can be approximated by using the variational distributions $q(\theta_j)$, $\forall\theta_j\in\Theta$ as

$$\begin{aligned}p(y_i^*|\mathbf{y}) &\simeq \int\int p\left(y_i^*\middle|\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D},\tau^{-1}\right)q\left(\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d=1}^{D}\right)q(\tau)d\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}d\tau\\ &= \int\int\mathcal{N}\left(y_i^*\middle|\mathbf{\Phi w},\tau^{-1}\right)\prod_{d=1}^{D}\mathcal{N}\left(\mathrm{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)\middle|\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right),\mathbf{\Sigma}^{(d)}\right)q(\tau)d\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}d\tau.\end{aligned}$$

As the variational distributions are independent, they can each be integrated over individually as

$$\begin{aligned}p(y_i^*|\mathbf{y}) = \int\int\mathcal{N}\left(y_i^*\middle|\mathbf{\Phi w},\tau^{-1}\right)\mathcal{N}\left(\mathrm{vec}\left(\boldsymbol{\mathcal{W}}^{(1)}\right)\middle|\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(1)}\right),\mathbf{\Sigma}^{(1)}\right)d\mathrm{vec}\left(\boldsymbol{\mathcal{W}}^{(1)}\right)\dots\\ \cdots\prod_{d\neq1}\mathcal{N}\left(\mathrm{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)\middle|\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right),\mathbf{\Sigma}^{(d)}\right)q(\tau)d\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d\neq1}d\tau\end{aligned}$$

Now, integrating over each core $\{\boldsymbol{\mathcal{W}}^{(d)}\}_{d=1}^{D}$, we can incorporate their posterior distribution $q(\boldsymbol{\mathcal{W}}^{(d)})$ into the distribution of the unseen data as

$$\begin{aligned}p(y_i^*|\mathbf{y}) &= \int\int\mathcal{N}\left(y_i^*\middle|\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(1)}\right)^{\intercal}\mathbf{G}_{\backslash1}(x_i),\tau^{-1}+\mathbf{G}_{\backslash1}^{\intercal}(x_i)\mathbf{V}^{(1)}\mathbf{G}_{\backslash1}\right)\dots\\ &\qquad\cdots\prod_{d\neq1}\left(\mathrm{vec}\left(\boldsymbol{\mathcal{W}}^{(d)}\right)\middle|\mathrm{vec}\left(\tilde{\boldsymbol{\mathcal{W}}}^{(d)}\right),\mathbf{\Sigma}^{(d)}\right)q(\tau)d\left\{\boldsymbol{\mathcal{W}}^{(d)}\right\}_{d\neq1}d\tau\\ &\qquad\qquad\vdots\\ &= \int\mathcal{N}\left(y_i^*\middle|\mathbf{\Phi w},\tau^{-1}+\sum_{d=1}^{D}\mathbf{G}_{\backslash d}(x_i)^{\intercal}\mathbf{\Sigma}^{(d)}\mathbf{G}_{\backslash d}(x_i)\right)q(\tau)d\tau\end{aligned}$$

Now substituting the posterior distribution of τ, we find

$$p(y_i^*|\mathbf{y}) = \int\mathcal{N}\left(y_i^*\middle|\mathbf{\Phi w},\tau^{-1}+\sum_{d=1}^{D}\mathbf{G}_{\backslash d}(x_i)^{\intercal}\mathbf{\Sigma}^{(d)}\mathbf{G}_{\backslash d}(x_i)\right)\mathrm{Ga}(\tau|a_N,b_N)d\tau,$$

yielding a Student's t-distribution

$$p(y_i^*|\mathbf{y}) \simeq \mathcal{T}\left(y_i^*\Big|\mathbf{\Phi w}, \left\{\frac{b_N}{a_N} + \sum_{d=1}^{D} \mathbf{G}_{\setminus d}(x_i)^\intercal \mathbf{\Sigma}^{(d)} \mathbf{G}_{\setminus d}(x_i)\right\}^{-1}, 2a_N\right). \tag{A-20}$$

That is to say, it is a Student's t-distribution $\mathcal{T}(y_i^*, \mathcal{S}_i, \nu_y)$, with parameters

$$\begin{aligned} y_i^* &= \mathbf{\Phi w}, & \nu_y &= 2a_N \\ \mathcal{S}_i &= \left\{\frac{a_N}{b_N} + \sum_{d=1}^{D} \mathbf{G}_{\setminus d}(x_i)^\intercal \mathbf{\Sigma}^{(d)} \mathbf{G}_{\setminus d}(x_i)\right\}^{-1} \end{aligned} \tag{A-21}$$

With its predictive variance $\mathrm{Var}(y_i) = \frac{\nu_y}{\nu_y - 2}\mathcal{S}_i$.

A-9 Upper and Lower Bounds on Core Updates

A-9-1 Matrix G

The column m, and with it the elements of the matrix-matrix product $\mathbf{W}^{(k)\intercal}\mathbf{\Phi}^{(k)}$ are bounded by

$$\sigma_{\min}(\mathbf{W}^{(k)\intercal})||\mathbf{\Phi}^{(k)}_{:,m}|| \leq ||(\mathbf{W}^{(k)\intercal}\mathbf{\Phi}^{(k)})_{:,m}|| \leq \sigma_{\max}(\mathbf{W}^{(k)\intercal})||\mathbf{\Phi}^{(k)}_{:,m}||$$

. Repeating this over $D-1$ Hadamard products, the bounds for the elements of $\mathbf{G}^{(d)}$ are given by

$$\sigma_{\min}(\mathbf{W}^\intercal)^{D-1}||\mathbf{\Phi}_{:,m}||^{D-1} \leq ||\mathbf{G}_{:,m}|| \leq \sigma_{\max}(\mathbf{W}^\intercal)^{D-1}||\mathbf{\Phi}_{:,m}||^{D-1},$$

With unit norm columns of $\mathbf{\Phi}^{(k)}$, we find

$$\sigma_{\min}(\mathbf{W}^\intercal) \leq ||\mathbf{G}_{:,m}|| \leq \sigma_{\max}(\mathbf{W}^\intercal).$$

For the TT-formulation of $\mathbf{G}_{\setminus d}$, we can see that the bounds for the forward and backward accumulators are identical. The magnitude of the columns over a row-wise Khatri-Rao product, will be the multiplication of the columns of both matrices. Starting from $\mathbf{G}_{<2}$, we can find it's columns bounded by

$$||\mathbf{\Phi}^{(1)}_m||\sigma_{\min}(\mathbf{W}^{(1)}_{(3)}) \leq ||\mathbf{G}_{<2:,m}|| \leq ||\mathbf{\Phi}^{(1)}_m||\sigma_{\min}(\mathbf{W}^{(1)}_{(3)}).$$

Under the assumption of unit norm columns of $||\mathbf{\Phi}^{(k)}$ and identical singular values for all $\boldsymbol{\mathcal{W}}$, this simplifies to

$$\sigma_{\min}(\mathbf{W}) \leq ||\mathbf{G}_{<2:,m}|| \leq \sigma_{\min}(\mathbf{W}).$$

Continuing to the second core we find

$$\sigma_{\min}(\mathbf{W})^2 \leq ||\mathbf{G}_{<2:,m}|| \leq \sigma_{\min}(\mathbf{W})^2.$$

And generalizing to all cores we find

$$\sigma_{\min}(\mathbf{W})^{D-1} \leq ||\mathbf{G}_{<2:,m}|| \leq \sigma_{\min}(\mathbf{W})^{D-1}.$$

A-9-2 matrix $\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus d}^{\intercal}$

From the CPD formulation given in (2-47), we can see for one step, the magnitude of the product is bounded by

$$\sigma_{\min}(\mathbf{W})^2 \leq \mathbb{E}_q[\mathbf{G}_{<1}\mathbf{G}_{<1}^{\intercal}] \leq \sigma_{\max}(\mathbf{W})^2. \tag{A-22}$$

Repeating this over $D-1$ hadamard products, we find the bounds on $E_q[\mathbf{G}\mathbf{G}^{\intercal}]$ as

$$\sigma_{\min}(\mathbf{W})^{2(D-1)} \leq \mathbb{E}_q[\mathbf{G}_{\setminus d}\mathbf{G}_{\setminus}^{\intercal}] \leq \sigma_{\max}(\mathbf{W})^{2(D-1)} \tag{A-23}$$

in the CPD formulation.

For the TT formulation, starting at $k = 2$, we find the bounds on the first step of the accumulator $\mathbf{H}_{<d}$ as

$$\sigma_{\min}(\mathbf{W})^2 \leq \mathbf{H}_{<1} \leq \sigma_{\max}(\mathbf{W})^2, \tag{A-24}$$

Incorporating the assumptions made previously. Repeating this for $D-1$ matrix multiplications, we find the same bounds as the CPD case

$$\sigma_{\min}(\mathbf{W})^{2(D-1)} \leq \mathbf{H}_{<D-1} \leq \sigma_{\max}(\mathbf{W})^{2(D-1)} \tag{A-25}$$

Bibliography

[1] Suzanne Aigrain and Daniel Foreman-Mackey. Gaussian Process Regression for Astronomical Time Series. *Annual Review of Astronomy and Astrophysics Downloaded from www.annualreviews.org. Guest*, 13:57, 2026.

[2] Ahmet Alkan and Mücahid Günay. Identification of EMG signals using discriminant analysis and SVM classifier. *Expert Systems with Applications*, 39(1):44–47, 1 2012.

[3] Barry Becker and Ronny Kohavi. Adult, 1996.

[4] Alain Berlinet and Christine Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer US, Boston, MA, 2004.

[5] Thomas Brooks, D. Pope, and Michael Marcolini. Airfoil Self-Noise, 1989.

[6] J Douglas Carroll and Jih-Jie Chang. Analysis of Individual Differences In Multidimensional Scaling via an N-Way Generalization of "Eckart-Young" Decomposition. *Psychometrika*, 35(3), 9 1970.

[7] Andrzej Cichocki, Danilo Mandic, Lieven De Lathauwer, Guoxu Zhou, Qibin Zhao, Cesar Caiafa, and Huy Anh Phan. Tensor decompositions for signal processing applications: From two-way to multiway component analysis, 3 2015.

[8] Corinna Cortes, Vladimir Vapnik, and Lorenza Saitta. Support-Vector Networs. *Machine Leaming*, 20(3):273–297, 1995.

[9] Tri Dao, Christopher De Sa, and Christopher Ré. Gaussian Quadrature for Kernel Features. Technical report, Stanford University, 1 2018.

[10] Xi Deng, Shun Peng Zhu, Lanyi Wang, Changqi Luo, Sicheng Fu, and Qingyuan Wang. Probabilistic framework for strain-based fatigue life prediction and uncertainty quantification using interpretable machine learning. *International Journal of Fatigue*, 190, 1 2025.

[11] Cesare Donati, Martina Mammarella, Giuseppe C. Calafiore, Fabrizio Dabbene, Constantino Lagoa, and Carlo Novara. A kernel-based approach to physics-informed nonlinear system identification. *IEEE Transactions on Automatic Control*, 2 2026.

[12] Petros Drineas and Michael W Mahoney. On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.

[13] Ke Lin Du, Bingchun Jiang, Jiabin Lu, Jingyu Hua, and M. N.S. Swamy. Exploring Kernel Machines and Support Vector Machines: Principles, Techniques, and Future Directions. *Mathematics*, 12(24), 12 2024.

[14] Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep Convolutional Networks as shallow Gaussian Processes. In *International Conference on Learning Representations*. ICLR, 5 2019.

[15] J. Gerritsma, R. Onnink, and A. Versluis. Yacht Hydrodynamics, 1981.

[16] Mohamed Goudjil, Mouloud Koudil, Mouldi Bedda, and Noureddine Ghoggali. A Novel Active Learning Method Using SVM for Text Classification. *International Journal of Automation and Computing*, 15(3):290–298, 6 2018.

[17] Barbara Hammer and Kai Gersmann. A Note on the Universal Approximation Capability of Support Vector Machines. *NeuralProcessing Letters*, 17, 2003.

[18] Zhifeng Hao, Lifang He, Bingqian Chen, and Xiaowei Yang. A linear support higher-order tensor machine for classification. *IEEE Transactions on Image Processing*, 22(7):2911–2920, 2013.

[19] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics*, 10(16):1–84, 1970.

[20] Lifang He, Chun-Ta Lu, Guixiang Ma, Shen Wang, Linlin Shen, Philip S Yu, and Ann B Ragin. Kernelized Support Tensor Machines. *International conference on machine learning*, 2017.

[21] James Hensman and Nicolas Durrande. Variational Fourier Features for Gaussian Processes. *Journal of Machine Learning Research*, 18:1–52, 2018.

[22] Jesper L. Hinrich and Morten Mørup. *27th EUSIPCO 2019 : European Signal Processing Conference : A Coruna, Spain, September 2-6, 2019*. IEEE, 2019.

[23] Arthur E Hoerl and Robert W Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *TECHNOMETRICS*, 12(1), 2 1970.

[24] Peter D. Hoff. Multilinear tensor regression for longitudinal relational data. *Annals of Applied Statistics*, 9(3):1169–1193, 9 2015.

[25] Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermondt. Spambase, 1999.

[26] C G Khatri and C Radhakrishna Rao. Solutions to Some Functional Equations and Their Applications to Characterization of Probability Distributions. *The Indian Journal of Statistics*, 30(2):167–180, 6 1968.

[27] Afra Kilic and Kim Batselier. Interpretable Bayesian Tensor Network Kernel Machines with Automatic Rank and Feature Selection. 7 2025.

[28] Tamara G Kolda. SANDIA REPORT Multilinear operators for higher-order decompositions. Technical report, Sandia, Livermore, 2006.

[29] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications, 2009.

[30] Kriton Konstantinidis, Yao Lei Xu, Qibin Zhao, and Danilo P. Mandic. Variational Bayesian Tensor Networks with Structured Posteriors. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, volume 2022-May, pages 3638–3642. Institute of Electrical and Electronics Engineers Inc., 2022.

[31] Solomon Kullback. *Information theory and statistics*. Dover, 1978.

[32] Steve Lawrence, C Lee Giles, and Ah Chung Tsoi. Lessons in Neural Network Training: Overfitting May be Harder than Expected. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence,*, pages 540–545. AAAI Press, 1997.

[33] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep Neural Networks as Gaussian Processes. In *ICLR*, 3 2018.

[34] Li Liu, Haocheng Sun, and Fanzhang Li. A Lie group kernel learning method for medical image classification. *Pattern Recognition*, 142:109735, 10 2023.

[35] David J. C. MacKay. Bayesian Methods for Backpropagation Networks. In *Models of Neural Networks III*, pages 211–254. Springer, 1996.

[36] Clara Menzen, Eva Memmel, Kim Batselier, and Manon Kok. Projecting basis functions with tensor networks for Gaussian process regression. Technical report, Delft Center for Systems and Control, Delft, 10 2023.

[37] Mojmír Mutný and Andreas Krause. Efficient High Dimensional Bayesian Optimization with Additivity and Quadrature Fourier Features. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems*, 2018.

[38] Radford M. Neal. *Bayesian Learning for Neural Networks*, volume 118. Springer New York, New York, NY, 1996.

[39] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian Deep Convolutional Networks with Many Channels are Gaussian Processes. In *ICLR*, 8 2020.

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

[41] Ali Rahimi and Ben Recht. Random Features for Large-Scale Kernel Machines. In J Platt, D Koller, Y Singer, and S Roweis, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2007.

[42] Piyush Rai, Yingjian Wang, and Lawrence Carin. Leveraging Features and Networks for Probabilistic Tensor Decomposition. In *AAAI Conference on Artificial Intelligence*, Austin, Texas, 1 2023.

[43] Carl Edward. Rasmussen and Christopher K. I.. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.

[44] Richik Sengupta, Soumik Adhikary, Ivan Oseledets, and Jacob Biamonte. Tensor networks in machine learning. *European Mathematical Society Magazine*, 126:4–12, 10 2022.

[45] Arno Solin and Simo Särkkä. Hilbert Space Methods for Reduced-Rank Gaussian Process Regression. *Statistics and Computing*, 2, 8 2019.

[46] Johan A. K.. Suykens. *Least squares support vector machines*. World Scientific, 2005.

[47] Dacheng Tao, Xuelong Li, Xindong Wu, Weiming Hu, and Stephen J. Maybank. Supervised tensor learning. *Knowledge and Information Systems*, 13(1):1–42, 2007.

[48] Ye Min Thant, Taishiro Wakamiya, Methawee Nukunudompanich, Keisuke Kameda, Manabu Ihara, and Sergei Manzhos. Kernel regression methods for prediction of materials properties: Recent developments. *Chemical Physics Reviews*, 6(1), 3 2025.

[49] Frederiek Wesel and Kim Batselier. Large-Scale Learning with Fourier Features and Tensor Decompositions. *Advances in Neural Information Processing Systems*, 34, 10 2021.

[50] Frederiek Wesel and Kim Batselier. Quantized Fourier and Polynomial Features for more Expressive Tensor Network Models. In *International Conference on Artificial Intelligence and Statistics*, pages 1261–1269. PMLR, 3 2024.

[51] Christopher K. I. Williams and Matthias Seeger. Using the Nystrom Method to Speed Up Kernel Machines. *Advances in Neural Information Processing Systems*, 13, 2000.

[52] John Winn and Christopher M Bishop. Variational Message Passing. *Journal of Machine Learning Research*, 5, 2004.

[53] Angeliki Xifara and Athanasios Tsanas. Energy Efficiency, 2012.

[54] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. Temporal Collaborative Filtering with Bayesian Probabilistic Tensor Factorization. In *SIAM International Conference on Data Mining*, page 211222, Columbus, Ohio, 12 2010. SIAM.

[55] Le Xu, Lei Cheng, Ngai Wong, and Yik Chung Wu. Probabilistic Tensor Train Decomposition with Automatic Rank Determination from Noisy Data. In *IEEE Workshop on Statistical Signal Processing Proceedings*, volume 2021-July, pages 461–465, Rio de Janeiro, Brazil, 7 2021. IEEE Computer Society.

[56] I-Cheng Yeh. Concrete Compressive Strength, 1998.

[57] Qibin Zhao, Liqing Zhang, and Andrzej Cichocki. Bayesian CP Factorization of Incomplete Tensors with Automatic Rank Determination. *IEEE*, 37(37), 10 2015.

[58] Qibin Zhao, Guoxu Zhou, Tulay Adali, Liqing Zhang, and Andrzej Cichocki. Kernel-Based Tensor Partial Least Squares for Reconstruction of Limb Movements. In *IEEE international conference on acoustics, speech and signal processing*, pages 3577–3581. IEEE, 2013.

Glossary

List of Acronyms

AI	Artificial Intelligence
ALS	Alternating Least Squares
ARD	Automatic Relevance Detection
BTN	Bayesian Tensor Network
BTTKM	Bayesian Tensor Train Kernel Machine
CPD	Canonical Polyadic Decomposition
ELBO	Evidence Lower Bound
GP	Gaussian Process
KL	Kullback-Leibler
KRR	Kernel Ridge Regression
NLL	Negative Log-Likelihood
RMSE	Root Mean Square Error
RKHS	Reproducing Kernel Hilbert Space
SP-BTN	Structured Posterior Bayesian Tensor Network
SVM	Support Vector Machine
T-KRR	Tensor Kernel Ridge Regression
TN	Tensor Network
TNKM	Tensor Network Kernel Machine
TT	Tensor Train

List of Symbols

$\boldsymbol{\delta}_d$	Feature dimension precision parameter
$\boldsymbol{\lambda}_d$	Tensor rank precision parameter
$\boldsymbol{\Sigma}^{(d)}$	Covariance matrix corresponding to d-th core
$\boldsymbol{\Phi}$	Feature map
$\boldsymbol{\Phi}^{(d)}$	Component matrix
$\Gamma(a)$	Gamma function
τ	Noise precision
Θ	Set of latent variables and hyperparameters
$\varphi(\cdot)$	Feature map
$\mathbf{G}_{<d}$	Forward recursive accumulator
$\mathbf{G}_{>d}$	Backward recursive accumulator
$\mathbf{e}$	Noise vector
$\mathbf{H}_{<d}$	Forward recursive accumulator expected Gram matrix
$\mathbf{H}_{>d}$	Backward recursive accumulator expected Gram matrix
$\mathbf{w}$	Weights vector
$\mathbf{W}^{(d)}$	Factor matrix
$\mathbf{y}$	Observations
$\mathcal{L}(q)$	Evidence Lower Bound
$\boldsymbol{\mathcal{V}}^{(d)}$	Variance tensor
$\boldsymbol{\mathcal{W}}$	Tensor
$\boldsymbol{\mathcal{W}}^{(d)}$	TT-core
$\tilde{\boldsymbol{\mathcal{W}}}^{(d)}$	Mean estimation of d-th core
D	Data dimension
M	Feature dimension
N	Data size
R	TT-Rank
y_i^*	Unseen data
$_{(n)}$	Mode-n unfolding
$\circ$	Outer product
$\circledast$	Hadamard product
$\mathcal{R}\{\cdot\}_{I\times J}$	Reshape operator
$\odot$	(row-wise) Khatri-Rao product
$\otimes$	Kronecker product
$\text{diag}(\cdot)$	Diagonal operator
$\text{vec}(\cdot)$	Vectorization operator
$l(\Theta)$	Log-joint distribution