Batch Bayesian Learning of Large-Scale LS-SVMs Based on Low-rank Tensor Networks

Chenxu Wang

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Batch Bayesian Learning of Large-Scale LS-SVMs Based on Low-rank Tensor Networks

Master of Science Thesis

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

Chenxu Wang

June 23, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Least Squares Support Vector Machines (LS-SVMs) are state-of-the-art learning algorithms that have been widely used for pattern recognition. The solution for an LS-SVM is found by solving a system of linear equations, which involves the computational complexity of $\mathcal{O}(N^3)$. When datasets get larger, solving LS-SVM problems with standard methods becomes burdensome or even unfeasible. The Tensor Train (TT) decomposition provides an approach to representing data in highly compressed formats without loss of accuracy. By converting vectors and matrices in the TT format, the storage and computational requirements can be greatly reduced. In this thesis, we develop a Bayesian learning method in the TT format to solve large-scale LS-SVM problems, which involves the computation of a matrix inverse. This method allows us to include the information we know about the model parameters in the prior distribution. As a result, we are able to obtain a probability distribution of the parameters, which enables us to construct confidence levels of the predictions. In the numerical experiment we show that the developed method performs competitively with the current methods.

Table of Contents

List of Figures

List of Tables

Acknowledgements

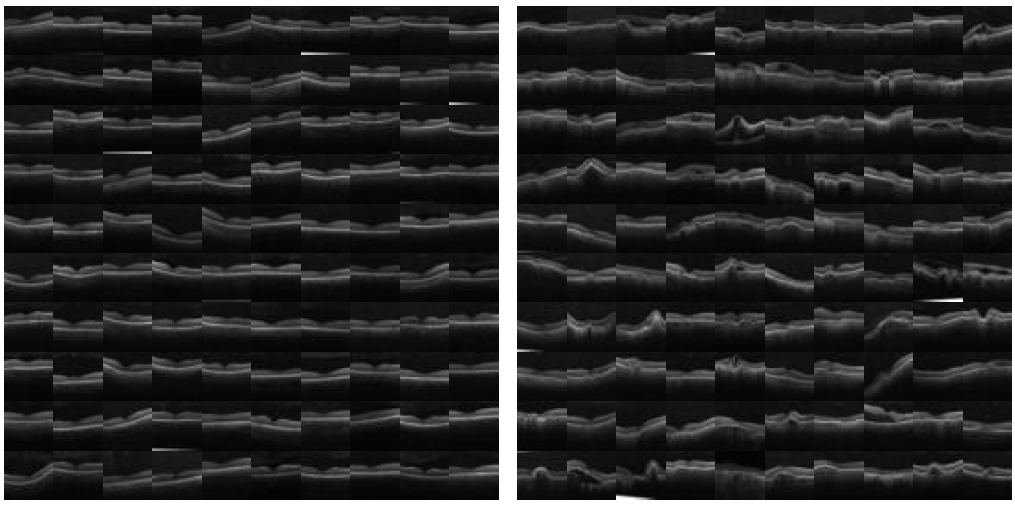
First of all, I would like to thank my supervisor, dr.ir. Kim Batselier, for introducing this new world of tensor networks to me. It was his guidance and patience that helped me complete this thesis project. Next, I wish to thank my parents for offering me this opportunity to study abroad. Their support is my biggest motivation. Finally, I would like to express my gratitude to my girlfriend, Hongxuan, for her encouragement and company in this special period due to the coronavirus pandemic. I am proud of myself in this two-year journey of studying in the TU Delft, and I am ready to write new chapters in my life.

Delft, University of Technology Chenxu Wang
June 23, 2021

Chapter 1

Introduction

Artificial intelligence (AI) has the potential to accelerate medical diagnosis and management by rapidly reviewing immense amounts of images and performing classification for human experts. For those cases that are easy to diagnose, they can be processed automatically by the machine. When the machine fails to give a definite result, the expert can take over. In this thesis, we will complete a classification task on a medical imaging dataset. We give images to the machine, and let it tell whether the corresponding patients are diseased or not. Figure 1-1 shows some of the examples of the dataset.



(a) Normal (b) Diseased

Figure 1-1: Samples of the medical imaging dataset.

The AI models we choose to solve the problem are Least Squares Support Vector Machines (LS-SVMs) [2], the least-squares versions of Support Vector Machines (SVMs) [3], which are state-of-the-art learning algorithms widely used for classification and regression analysis. For many algorithms, one has to explicitly transform the raw data into feature

vector representations via a user-specified feature map. However, as kernel-based methods, LS-SVMs enable us to operate in a high-dimensional, implicit feature space without ever computing the exact coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This approach is called the kernel trick. To take advantage of it, the solution has to be found in dual space, leading to solving a system of linear equations for LS-SVMs. This linear system can be further reduced to two symmetric positive definite (SPD) linear systems of size $N \times N$ for a training set of N data points, which may be solved with the computational complexity of $\mathcal{O}(N^3)$.

When datasets get larger and larger, solving LS-SVM problems with standard methods becomes burdensome or even unfeasible. To circumvent this bottleneck, some low-rank approximation methods are considered. The Nyström method [4] uses a random subset of the training set to find an approximation to the kernel matrix. Then the linear system is solved by applying the Sherman-Morrison-Woodbury formula [5], with the computational complexity of $\mathcal{O}(M^2N)$ for a subset containing M data points. The Nyström method also provides an explicit estimate of the nonlinear feature map, which makes it possible to solve LS-SVM problems in primal space. This approach is referred to as the Fixed Size LS-SVM (FS-LSSVM) [2]. Since the size of the vector of unknowns in primal space is proportional to the dimension of feature vectors, the computational requirements do not grow with the size of dataset.

The existing low-rank approximation methods only consider a subset of dataset, which may lose some key information. Tensor Networks (or tensor decompositions) provide an approach to representing data in highly compressed formats without loss of accuracy. Amongst them, the Tensor Train (TT) decomposition [6] is numerically stable and easy to compute. Moreover, structured vectors and matrices can be well represented in low-rank TT formats [7], and basic operations can be efficiently implemented. By converting the kernel matrix into the TT format, the storage and computational requirements can be significantly reduced.

In addition to the large-scale problem, there is also one drawback associated with LS-SVMs. Take our medical classification problem for example. Basic LS-SVM classifiers can only give the label of a given instance, i.e., if a patient is ill. However, when the machine is actually put into use, we need to know how confident they are to make a diagnosis, so that we can decide whether a human expert is required. To tackle this problem, we develop a Bayesian learning method for solving the LS-SVM problems. This approach allows us to include the information we have about the model parameters in the prior distribution. As a result, we are able to obtain a probability distribution of the parameters, which enables us to construct confidence levels of the predictions. There exists the batch solution to the posterior distribution, which involves computing a matrix inverse.

In this thesis, we will investigate whether the Batch Bayesian learning approach can be well applied to solving (large-scale) LS-SVM problems in the TT format. Specifically, the research questions to be answered are listed below.

- Q1. By converting the kernel matrix in the TT format, can we find a low-rank structure to efficiently store the data without losing its accuracy?
- Q2. Can the matrix inversion be reliably computed in the TT format, i.e., is the resulting covariance matrix guaranteed to be SPD?
- Q3. What are the advantages and disadvantages of our method compared with the state of the art, and what would be the suitable cases to implement our model?

Q4. In which aspects can the method be further improved?

The remainder of the thesis is organized as follows. Chapter 2 introduces basic theories of LS-SVMs and several methods for solving large-scale LS-SVM problems. Chapter 3 provide all the information needed about tensor networks, including the definition of the TT decomposition, basic operations, solving linear systems, and matrix inversion in the TT format. Chapter 4 illustrates the practical details about how to implement the Batch Bayesian learning approach in the TT context, and provides the numerical results of the model performance on the medical imaging dataset. Lastly, conclusion and some recommendations for future work are given in Chapter 5.

The notations used in this report are listed in Table 1-1.

Table 1-1: Basic notations and products.

y, γ, M	scalars
$\mathbf{x}, \boldsymbol{\alpha}$	vectors
$\mathbf{A}, \boldsymbol{\Omega}$	matrices
$\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{X}}$	tensors
$\mathbf{A}^\mathsf{T}, \mathbf{A}^{-1}, \|\mathbf{A}\|_F$	transpose, inverse and Frobenius norm of matrix $\mathbf{A}$
$\mathbf{A}^{(k)}, \boldsymbol{\mathcal{A}}^{(k)}$	the k-th matrix or tensor in a sequence
$\mathbf{I}_N$	identity matrix of size $N \times N$
$\mathbf{1}_N, \mathbf{0}_N$	vectors of all ones or zeros of length N
$\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$	d-way tensor of size $I_1 \times I_2 \times \cdots \times I_d$
$a_{i_1, i_2, \ldots, i_d}$	the $(i_1, i_2, \ldots, i_d)$-th entry of tensor $\boldsymbol{\mathcal{A}}$
$\langle \mathbf{a}, \mathbf{b} \rangle$	inner product of vectors $\mathbf{a}$ and $\mathbf{b}$
$\mathbf{A} \odot \mathbf{B}$	Hadamard (element-wise) product of matrices $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product of matrices $\mathbf{A}$ and $\mathbf{B}$

Chapter 2

LS-SVMs for Classification and Large-Scale Problems

In this Chapter, we introduce the basic theory of Least Squares Support Vector Machines (LS-SVMs), including the existing methods for solving large-scale problems. To have a good understanding, a brief overview on the formulation of Support Vector Machines (SVMs) is first given in Section 2-1, in which we show that one solves a convex Quadratic Programming (QP) problem in both primal and dual space for a linear classifier. The extension to the nonlinear case is achieved by the so-called kernel trick. Thereafter, Section 2-2 discusses the LS-SVM theory and basic methods for solving small- and medium-scale LS-SVM problems. With small modifications to the standard SVM, one solves a system of linear equations instead of a QP problem in the LS-SVM context. For problems with large-scale datasets (typically $N \geq 15000$), the computation of the system becomes burdensome. This is where the low-rank approximation methods come in. Section 2-3 introduces two methods suited for solving large-scale LS-SVM problems.

For the sake of simplicity, the discussion will only focus on the *binary classification* case. The formulations can be easily adapted for regression tasks [3, 2]. To perform the multiclass classification one can train multiple binary classifiers, known as the one-versus-one strategy [2].

2-1 Support Vector Machines

2-1-1 Linear classifiers

Given a training set for a binary classification problem $\{\mathbf{x}_k, y_k\}_{k=1}^N$, with input data $\mathbf{x}_k \in \mathbb{R}^n$ and class labels $y_k \in \{-1, +1\}$, a linear SVM classifier predicts the class $\hat{y}$ of a new instance $\mathbf{x} \in \mathbb{R}^n$ based on the sign of the weighted sum of all elements:

$$\hat{y}(\mathbf{x}) = \text{sign}\left(\mathbf{w}^\mathsf{T}\mathbf{x} + b\right), \tag{2-1}$$

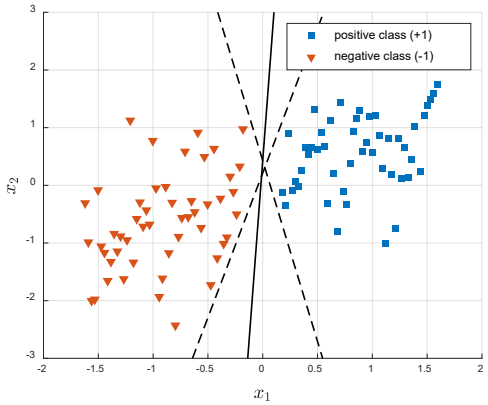


Figure 2-1: A separable dataset: several working hyperplanes.

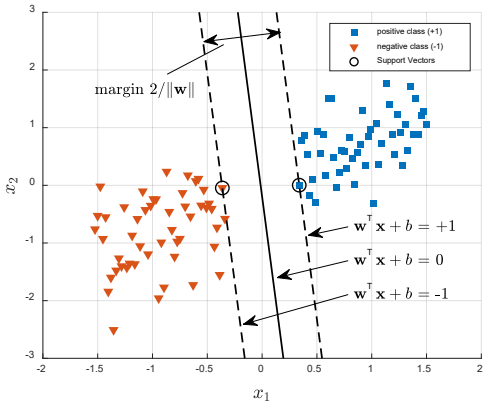


Figure 2-2: The hyperplane given by the SVM that maximizes the margin.

where the weight vector $\mathbf{w}$ and the bias term b together define a hyperplane (i.e., a decision boundary) to separate the given data points.

Hard margin classification

Figure 2-1 depicts a linearly separable dataset in a two-dimensional space. It is easy to show that there exist infinitely many hyperplanes which are able to separate the data correctly. The decision boundaries represented by the dashed lines will not generalize well on new instances, since they are so close to the data points. In contrast, the solid line represents the decision boundary of a linear SVM classifier, which not only correctly separates the two classes but also stays as far away from instances as possible. A natural question to be asked is: how can we measure this distance?

To do that, let the points closest to the decision boundary satisfy $|\mathbf{w}^\mathsf{T}\mathbf{x}_k + b| = 1$. As shown in Figure 2-2, two parallel dashed lines that are at equal distance to the decision boundary form a *margin* around it. In this case the margin equals $2/\|\mathbf{w}\|$. It is a desirable property to maximize this margin, which corresponds to minimizing $\|\mathbf{w}\|$.

For a separable case if we want all instances to be correctly classified, called the *hard margin classification*, we need

$$\begin{cases} \mathbf{w}^\mathsf{T}\mathbf{x}_k + b \geq +1, & \text{if} \quad y_k = +1 \\ \mathbf{w}^\mathsf{T}\mathbf{x}_k + b \leq -1, & \text{if} \quad y_k = -1 \end{cases} \quad \Longleftrightarrow \quad y_k \left(\mathbf{w}^\mathsf{T}\mathbf{x}_k + b\right) \geq 1, \quad k = 1, \ldots, N.$$

We can formulate our search for the optimal hyperplane as a constrained optimization problem. The objective is to maximize the margin under the constraints that all data points must lie on the correct side of the hyperplane:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} \\ \text{s.t.} \quad & y_k \left(\mathbf{w}^\mathsf{T}\mathbf{x}_k + b\right) \geq 1, \quad k = 1, \ldots, N. \end{aligned} \tag{2-2}$$

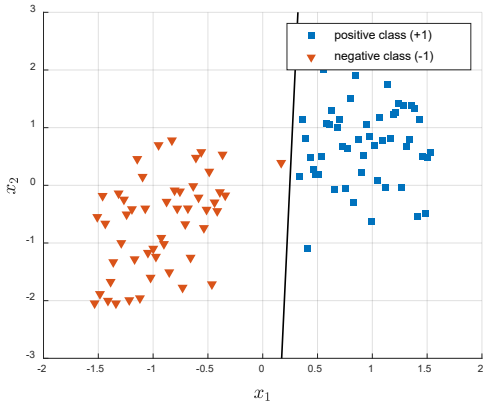


Figure 2-3: The hard margin classification is sensitive to outliers.

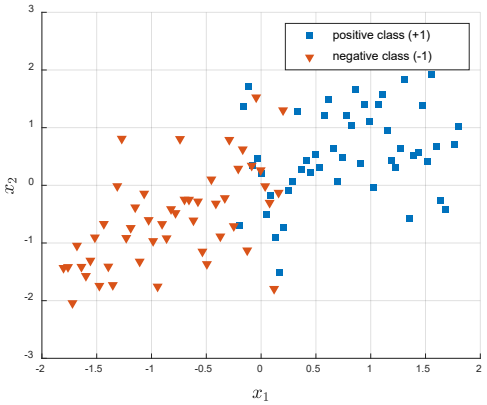


Figure 2-4: Most real-life classification problems are non-separable.

When we obtain the optimal pair $(\mathbf{w}^*, b^*)$, it can be shown that some instances have tight constraints, i.e., $y_k(\mathbf{w}^{*\mathsf{T}}\mathbf{x}_k + b^*) = 1$. These points are called the *support vectors* (circled in Figure 2-2), which determine the shape of the hyperplane – if we move one of them and retrain an SVM classifier, the resulting hyperplane would change.

Soft margin classification

There are two main issues with the hard margin classification, i.e., enforcing all data points to be correctly classified: (1) it is sensitive to outliers. Figure 2-3 shows the resulting decision boundary after adding just one outlier; (2) it only works for separable problems. However, most real-life datasets are noisy and non-separable, as depicted in Figure 2-4. As a consequence, we need to tolerate misclassified data points and at the same time limit their influence on the result. This strategy is known as the *soft margin classification*.

To achieve the soft margin objective, we introduce the slack variables $\xi_k \geq 0$ for all instances [8]: ξ_k measures how much the k-th instance is allowed to violate the margin. The optimization problem now becomes

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{k=1}^{N}\xi_k \\ \text{s.t.} \quad & y_k\left(\mathbf{w}^\mathsf{T}\mathbf{x}_k + b\right) \geq 1 - \xi_k, \quad k = 1, \ldots, N \\ & \xi_k \geq 0, \quad k = 1, \ldots, N, \end{aligned} \tag{2-3}$$

where $C > 0$ is a hyperparameter that allows us to define the trade-off between reducing the margin violations and increasing the margin [9].

According to the duality theory [10], given a constrained optimization problem, known as the *primal problem*, we can form the *Lagrangian* by introducing the Lagrange multipliers, take the conditions for optimality, and finally solve the *dual problem*.

The Lagrangian for problem (2-3) is

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\nu}) = \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C\sum_{k=1}^{N}\xi_k - \sum_{k=1}^{N}\alpha_k\left[y_k\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_k + b\right) - 1 + \xi_k\right] - \sum_{k=1}^{N}\nu_k\xi_k, \quad (2\text{-}4)$$

with Lagrange multipliers $\alpha_k \geq 0, \nu_k \geq 0$ for $k = 1, \ldots, N$. The solution is given by the saddle point of (2-4) [10]:

$$\max_{\boldsymbol{\alpha},\boldsymbol{\nu}} \min_{\mathbf{w},b,\boldsymbol{\xi}} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\nu}). \quad (2\text{-}5)$$

The Karush-Kuhn-Tucker (KKT) conditions of optimality [10] yields

$$\begin{cases} \dfrac{\partial L}{\partial \mathbf{w}} = 0 \quad \rightarrow \quad \mathbf{w} = \displaystyle\sum_{k=1}^{N}\alpha_k y_k \mathbf{x}_k, \\ \dfrac{\partial L}{\partial b} = 0 \quad \rightarrow \quad \displaystyle\sum_{k=1}^{N}\alpha_k y_k = 0, \\ \dfrac{\partial L}{\partial \xi_k} = 0 \quad \rightarrow \quad C - \alpha_k - \nu_k = 0. \end{cases} \quad (2\text{-}6)$$

By combining (2-4) - (2-6), we obtain the following QP problem as the dual problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\sum_{k=1}^{N}\sum_{l=1}^{N}\alpha_k\alpha_l y_k y_l \mathbf{x}_k^{\mathsf{T}}\mathbf{x}_l + \sum_{k=1}^{N}\alpha_k \\ \text{s.t.} \quad & \sum_{k=1}^{N}\alpha_k y_k = 0 \\ & 0 \leq \alpha_k \leq C, \quad k = 1, \ldots, N. \end{aligned} \quad (2\text{-}7)$$

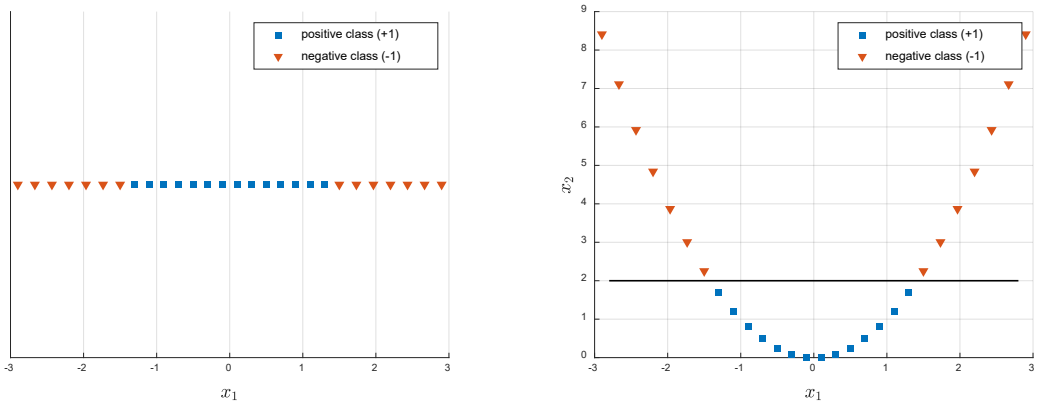
Once we find the optimal dual vector $\boldsymbol{\alpha}^*$, we can compute the pair $(\mathbf{w}^*, b^*)$ that minimizes the primal problem (2-3) as follows [11]:

$$\mathbf{w}^* = \sum_{k=1}^{N}\alpha_k^* y_k \mathbf{x}_k, \quad b^* = \frac{1}{n_s}\sum_{\substack{k=1 \\ \alpha_k^* > 0}}^{N}\left(y_k - \mathbf{x}_k^{\mathsf{T}}\mathbf{w}^*\right), \quad (2\text{-}8)$$

where n_s denotes the number of support vectors, which corresponds to the data points with non-zero α_k^* values. The linear SVM classifier takes the form

$$\hat{y}(\mathbf{x}) = \operatorname{sign}\left(\sum_{k=1}^{N}\alpha_k^* y_k \mathbf{x}_k^{\mathsf{T}}\mathbf{x} + b^*\right). \quad (2\text{-}9)$$

Note that the dual optimization problem (2-7) and the decision function (2-9) only depend on the inner product of instances. This provides an approach to extending linear SVMs to nonlinear cases [3].



(a) Nonseparable in a 1-D input space (b) Linearly separable in a 2-D feature space

Figure 2-5: Mapping data into a higher dimensional space makes a dataset linearly separable.

2-1-2 Nonlinear classifiers: Kernel trick

The basic idea of performing nonlinear classifications is to map the input data into a higher (or even infinite) dimensional feature space, in which the linear separating hyperplane is found. Figure 2-5(a) shows a one-dimensional dataset that is not linearly separable. By applying the mapping $\varphi(x) = [x, x^2]^\mathsf{T}$, the dataset is linearly separable as depicted in Figure 2-5(b).

It could be burdensome, however, to work in a huge dimensional feature space. Fortunately, the kernel functions enable us to do that without actually doing the computations in that space. Furthermore, explicit definition of the nonlinear map is not needed. This is known as the *kernel trick* [3]. Given a nonlinear map $\varphi(\cdot) : \mathbb{R}^n \to \mathcal{H}$, if for all $\mathbf{x}$, $\mathbf{x}' \in \mathbb{R}^n$:

$$K(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x})^\mathsf{T} \varphi(\mathbf{x}'), \tag{2-10}$$

then $K(\mathbf{x}, \mathbf{x}')$ is called a kernel function. To become a valid kernel, the continuous function $K(\mathbf{x}, \mathbf{x}')$ should be symmetric and satisfies Mercer's condition [12]. The commonly used kernels are listed as follows:

$$\begin{aligned} \text{Linear:} \quad & K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\mathsf{T} \mathbf{x}', \\ \text{Polynomial:} \quad & K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T} \mathbf{x}' + 1)^d, \\ \text{RBF:} \quad & K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma_{\text{rbf}}^2}\right). \end{aligned} \tag{2-11}$$

The following example illustrates how the kernel trick works. Suppose that we want to apply a second-degree polynomial transformation to a two-dimensional training set:

$$\varphi(\mathbf{x}) = \varphi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{bmatrix},$$

and train a linear SVM classifier on the transformed dataset. The inner product of the transformed vector is

$$\varphi(\mathbf{x})^\mathsf{T} \varphi(\mathbf{x}') = (x_1 x_2 + x_1' x_2')^2 = (\mathbf{x}^\mathsf{T} \mathbf{x}')^2.$$

Thus, there is no need to actually compute the transformation; just replace the inner product by its square in (2-7). With the help of the kernel trick, the extension from linear SVM classifiers to nonlinear cases is straightforward: we can replace $\mathbf{x}$ by $\varphi(\mathbf{x})$ and apply the kernel trick where possible. The primal problem now becomes

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{k=1}^{N}\xi_k \\ \text{s.t.} \quad & y_k\left(\mathbf{w}^\mathsf{T}\varphi(\mathbf{x}_k)+b\right) \geq 1-\xi_k, \quad k=1,\ldots,N \\ & \xi_k \geq 0, \quad k=1,\ldots,N. \end{aligned} \tag{2-12}$$

Note that $\varphi(\mathbf{x})$ is usually not well-defined and can even be infinite dimensional (e.g. when using the RBF kernel). Hence, it could be difficult or even impossible to compute the optimal vector $\mathbf{w}^*$ directly from the primal problem. On the other hand, the dual QP problem for the nonlinear case is written as

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\sum_{k=1}^{N}\sum_{l=1}^{N}\alpha_k\alpha_l y_k y_l K(\mathbf{x}_k,\mathbf{x}_l) + \sum_{k=1}^{N}\alpha_k \\ \text{s.t.} \quad & \sum_{k=1}^{N}\alpha_k y_k = 0 \\ & 0 \leq \alpha_k \leq C, \ k=1,\ldots,N, \end{aligned} \tag{2-13}$$

where $K(\mathbf{x}_k,\mathbf{x}_l) = \varphi(\mathbf{x}_k)^\mathsf{T}\varphi(\mathbf{x}_l)$ is given by the kernel trick. Surprisingly, predictions can be made without knowing $\mathbf{w}^*$:

$$\hat{y}(\mathbf{x}) = \operatorname{sign}\left(\sum_{k=1}^{N}\alpha_k^* y_k K(\mathbf{x}_k,\mathbf{x}) + b^*\right), \tag{2-14}$$

where $\boldsymbol{\alpha}^*$ is the solution to the QP problem (2-13) and b^* is computed by averaging over all the support vectors:

$$b^* = \frac{1}{n_s}\sum_{\substack{k=1 \\ \alpha_k^*>0}}^{N}\left(y_k - \sum_{\substack{l=1 \\ \alpha_l^*>0}}^{N}\alpha_l^* y_l K(\mathbf{x}_k,\mathbf{x}_l)\right).$$

Training of SVMs can be done using a QP solver. There also exist some fast algorithms, such as the Sequential Minimal Optimization (SMO) [13]. Some properties of SVM classifiers are summarized here.

- Choosing a positive definite kernel guarantees that we can obtain a global and unique solution $\boldsymbol{\alpha}^*$ to the QP problem (2-13).
- Most of the elements in $\boldsymbol{\alpha}^*$ are equal to zero (known as the sparseness property). Those non-zero entries correspond to the support vectors.

2-2 Least Squares Support Vector Machines

2-2-1 LS-SVM classifiers

LS-SVMs, proposed in [14], aim to simplify the formulation of standard SVMs while maintaining their advantages. In the LS-SVM context, the primal problem (2-12) is modified as follows [2]:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + \gamma\frac{1}{2}\sum_{k=1}^{N}\xi_k^2 \\ \text{s.t.} \quad & y_k\left(\mathbf{w}^\mathsf{T}\varphi(\mathbf{x}_k)+b\right) = 1-\xi_k, \quad k=1,\ldots,N. \end{aligned} \tag{2-15}$$

On the one hand, a sum of squared errors is used in cost function. On the other hand, the inequality constraints are replaced by equality constraints. As discussed before, $\mathbf{w}$ might be infinite dimensional for a nonlinear classifier. So the problem is better solved in dual space. The Lagrangian for (2-15) is given by

$$L(\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + \gamma\frac{1}{2}\sum_{k=1}^{N}\xi_k^2 - \sum_{k=1}^{N}\alpha_k\left[y_k\left(\mathbf{w}^\mathsf{T}\varphi(\mathbf{x}_k)+b\right)-1+\xi_k\right], \tag{2-16}$$

where the Lagrange multipliers α_k can be positive or negative due to the equality constraints. The conditions for optimality yield

$$\begin{cases} \dfrac{\partial L}{\partial \mathbf{w}} = 0 \quad \rightarrow \quad \mathbf{w} = \displaystyle\sum_{k=1}^{N}\alpha_k y_k \varphi(\mathbf{x}_k), \\ \dfrac{\partial L}{\partial b} = 0 \quad \rightarrow \quad \displaystyle\sum_{k=1}^{N}\alpha_k y_k = 0, \\ \dfrac{\partial L}{\partial \xi_k} = 0 \quad \rightarrow \quad \alpha_k = \gamma\xi_k, \quad k=1,\ldots,N, \\ \dfrac{\partial L}{\partial \alpha_k} = 0 \quad \rightarrow \quad y_k\left(\mathbf{w}^\mathsf{T}\varphi(\mathbf{x}_k)+b\right)-1+\xi_k = 0, \quad k=1,\ldots,N. \end{cases} \tag{2-17}$$

By eliminating $\mathbf{w}$ and $\boldsymbol{\xi}$, we obtain the following linear KKT system [2] as the dual problem:

$$\begin{bmatrix} 0 & \mathbf{y}^\mathsf{T} \\ \mathbf{y} & \boldsymbol{\Omega} + \mathbf{I}_N/\gamma \end{bmatrix}\begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{1}_N \end{bmatrix}. \tag{2-18}$$

Kernels listed in (2-11) can be well applied to LS-SVMs [2]. The kernel trick is applied within the matrix $\boldsymbol{\Omega} \in \mathbb{R}^{N\times N}$, defined element-wise as

$$\Omega_{k,l} = y_k y_l \varphi(\mathbf{x}_k)^\mathsf{T}\varphi(\mathbf{x}_l) = y_k y_l K(\mathbf{x}_k,\mathbf{x}_l), \quad k,l=1,\ldots,N.$$

Once obtained the solution $(b^*,\boldsymbol{\alpha}^*)$ to the linear system (2-18), the resulting classifier takes the form

$$\hat{y}(\mathbf{x}) = \operatorname{sign}\left(\sum_{k=1}^{N}\alpha_k^* y_k K(\mathbf{x}_k,\mathbf{x}) + b^*\right). \tag{2-19}$$

Finally, some important properties of LS-SVMs are listed here.

- The dual problem for linear and non-linear LS-SVMs corresponds to solving a linear KKT system instead of a QP problem. A unique solution is guaranteed given a full-rank system matrix.
- The size of the KKT system is not influenced by the dimension of the input space n, but is only determined by the number of data points N.
- The regularization parameter γ in (2-18) allows us to determine the complexity of the model, which can be tuned by the cross-validated grid-search.
- In the LS-SVM context, no α_k^* values will be exactly equal to zero. The points located close and far from the decision boundary contribute more to the model. There are, however, many approaches to obtaining a sparse model. For instance, one can apply a simple pruning algorithm [2]: train an LS-SVM in several steps where in each step a small amount of instances (e.g. 5% of the data) with smallest support values $|\alpha_k|$ are omitted, and re-train the model on the reduced training set.

2-2-2 Solving the LS-SVM KKT system

There are various methods for solving linear systems. Most of them, however, are suited for the system with symmetric positive definite (SPD) matrix. The linear KKT system (2-18) is not SPD due to the existence of the top-left zero. Fortunately, we can transform the system according to [15] as follows

$$\begin{bmatrix} s & \mathbf{0}_N^\mathsf{T} \\ \mathbf{0}_N & \mathbf{A} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} + b\mathbf{A}^{-1}\mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{y}^\mathsf{T}\mathbf{A}^{-1}\mathbf{1}_N \\ \mathbf{1}_N \end{bmatrix}, \tag{2-20}$$

where $\mathbf{A} = \boldsymbol{\Omega} + \mathbf{I}_N/\gamma$ and $s = \mathbf{y}^\mathsf{T}\mathbf{A}^{-1}\mathbf{y} > 0$. Since $\mathbf{A}$ is SPD, the overall matrix is SPD. The solution pair $(b, \boldsymbol{\alpha})$ can then be found in the following steps [2]:

(1) Solve $\mathbf{z}_2, \mathbf{z}_1$ from $\mathbf{A}\mathbf{z}_2 = \mathbf{y}$ and $\mathbf{A}\mathbf{z}_1 = \mathbf{1}_N$.
(2) Compute $s = \mathbf{y}^\mathsf{T}\mathbf{z}_2$.
(3) Find solution $b = \mathbf{z}_2^\mathsf{T}\mathbf{1}_N/s$ and $\boldsymbol{\alpha} = \mathbf{z}_1 - \mathbf{z}_2 b$.

Now the problem reduces to solving two linear systems of the form $\mathbf{A}\mathbf{z} = \mathbf{d}$ with $\mathbf{A} = \mathbf{A}^\mathsf{T} > 0$.

Direct methods

Direct methods are well suited for solving small-sized systems ($N \leq 3000$) [2]. One of the most efficient ways is to apply the *Cholesky factorization* [5]. Given an SPD matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, there exists a unique lower triangular matrix $\mathbf{G} \in \mathbb{R}^{N\times N}$ with positive diagonal entries such that $\mathbf{A} = \mathbf{G}\mathbf{G}^\mathsf{T}$, where $\mathbf{G}$ is called the Cholesky factor. The linear system $\mathbf{A}\mathbf{z} = \mathbf{d}$ can be decomposed into two triangular systems

$$\begin{cases} \mathbf{A}\mathbf{z} = \mathbf{d} \\ \mathbf{A} = \mathbf{G}\mathbf{G}^\mathsf{T} \end{cases} \iff \begin{cases} \mathbf{G}\mathbf{z}' = \mathbf{d}, \\ \mathbf{G}^\mathsf{T}\mathbf{z} = \mathbf{z}', \end{cases}$$

which are easily solved using forward and backward substitution.

Other direct methods include LU factorization and Gaussian elimination. Direct methods would in general involve a computational complexity of $\mathcal{O}(N^3)$ and a memory requirements of $\mathcal{O}(N^2)$ [5] assuming that the matrix is completely stored in memory. Therefore, these methods become unfeasible as the size of matrix grows. For larger datasets the use of iterative methods is recommended.

Iterative methods

Iterative methods are appropriate for medium-scale problems ($3000 \leq N \leq 15000$) [16]. Given a linear system $\mathbf{Az} = \mathbf{d}$ with $\mathbf{A} = \mathbf{A}^\mathsf{T} > 0$, the iterative methods start with an initial guess $\mathbf{z}^{(0)}$. At each step i, the vector $\mathbf{z}^{(i)}$ is updated based on some algorithms. The sequence $\{\mathbf{z}^{(0)}, \mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(i)}, \ldots\}$ will converge to the optimal solution $\mathbf{z}^*$.

There are various methods that are widely used [5], including Gauss-Seidel, Successive Over-relaxation (SOR), Conjugate Gradient (CG), Generalized Minimal Residual (GMRES), etc. We introduce the CG methods because of their efficiency. For further reading about other methods, [5, 17, 18, 16, 19] are proven to be useful.

Consider the cost function of the quadratic form

$$V(\mathbf{z}) = \frac{1}{2}\langle \mathbf{z}, \mathbf{Az} \rangle - \langle \mathbf{z}, \mathbf{d} \rangle \tag{2-21}$$

the solution to the linear system is found as $\arg\min_{\mathbf{z}} V(\mathbf{z})$. For the Hestenes-Stiefel CG algorithm [5], shown in Algorithm 1, $V(\mathbf{z}^{(i)})$ is guaranteed to decrease in each iteration step i [20]. The $\mathbf{A}$-norm of a vector $\boldsymbol{\rho}$ is defined as

$$\|\boldsymbol{\rho}\|_{\mathbf{A}} = \left(\boldsymbol{\rho}^\mathsf{T}\mathbf{A}\boldsymbol{\rho}\right)^{\frac{1}{2}}.$$

The algorithm stops when one of the three stopping criteria is satisfied. While the first one is related to the maximal number of iterations $i_{\max}$, the second one is based on the norm of the residuals, and the third one is based on the evolution of the cost function (2-21). The convergence rate depends not only on the requested accuracy ϵ, but also on the condition number $\kappa = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$ [2, 5].

When the matrix $\mathbf{A}$ is too large for the memory requirements, one can recompute individual row of matrix in each iteration step [20], which costs $\mathcal{O}(N^2)$ operations per step but also reduces the memory requirements to $\mathcal{O}(N)$. In general, the total computational complexity of the CG methods is $\mathcal{O}(iN^2)$ [16], where i denotes the number of iterations before convergence. The basic CG algorithm might be further improved by using preconditioners [2].

2-3 Large-Scale Problems

Solving the LS-SVM dual problem corresponds to solving two SPD systems of size $N \times N$, which involve the computational requirements of $\mathcal{O}(N^3)$. Work on large-scale methods proposes solutions to circumvent this bottleneck.

Algorithm 1: Hestenes-Stiefel Conjugate Gradient [5]

Input: maximal iteration $i_{\max}$, accuracy ϵ, system matrix $\mathbf{A}$, right-hand side $\mathbf{d}$
Output: estimated solution $\hat{\mathbf{z}}$ such that $\mathbf{A}\hat{\mathbf{z}} \approx \mathbf{d}$

$i = 0; \mathbf{z}^{(0)} = 0; \mathbf{r}^{(0)} = \mathbf{d}$
while $(i < i_{\max}) \wedge \left(\mathbf{r}^{(i)} > \epsilon\|\mathbf{d}\|\right) \wedge \left(V(\mathbf{z}^{(i-1)}) - V(\mathbf{z}^{(i)}) > \epsilon\right)$ **do**
  $i \leftarrow i + 1$
  if $i = 1$ **then**
    $\boldsymbol{\rho}^{(i)} \leftarrow \mathbf{r}^{(0)}$
  else
    $\beta \leftarrow \|\mathbf{r}^{(i-1)}\|^2 / \|\mathbf{r}^{(i-2)}\|^2$
    $\boldsymbol{\rho}^{(i)} \leftarrow \mathbf{r}^{(i-1)} + \beta\boldsymbol{\rho}^{(i-1)}$
  end
  $v^{(i)} \leftarrow \|\mathbf{r}^{(i-1)}\|^2 / \|\boldsymbol{\rho}^{(i)}\|^2_{\mathbf{A}}$
  $\mathbf{z}^{(i)} \leftarrow \mathbf{z}^{(i-1)} + v^{(i)}\boldsymbol{\rho}^{(i)}$
  $\mathbf{r}^{(i)} \leftarrow \mathbf{r}^{(i-1)} - v^{(i)}\mathbf{A}\boldsymbol{\rho}^{(i)}$
  $V(\mathbf{z}^{(i)}) \leftarrow -\frac{1}{2}(\mathbf{r}^{(i)} + \mathbf{d})^{\mathsf{T}}\mathbf{z}^{(i)}$
end
return $\hat{\mathbf{z}} \leftarrow \mathbf{z}^{(i)}$

One approach is to find low-rank approximations to the kernel matrix $\boldsymbol{\Omega}$ and do the computations based on these approximations, e.g., the Nyström method [4]. Another approach is to solve LS-SVM problems in primal space because then the size of the vector of unknowns is proportional to the dimension of feature vector and not to the number of data points. However, the feature space mapping induced by the kernel is needed in order to obtain non-linearity. For this purpose, a method of Fixed Size LS-SVM (FS-LSSVM) is proposed [2], which uses the Nyström method to estimate the feature space mapping.

2-3-1 The Nyström method

The Nyström method is related to finding a low-rank approximation to the given kernel matrix $\boldsymbol{\Omega} \in \mathbb{R}^{N\times N}$ by randomly selecting M data points $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ from the training set. Let us denote this small kernel matrix by $\bar{\boldsymbol{\Omega}} \in \mathbb{R}^{M\times M}$. Consider the eigenvalue decomposition of $\bar{\boldsymbol{\Omega}}$

$$\bar{\boldsymbol{\Omega}}\bar{\mathbf{U}} = \bar{\mathbf{U}}\bar{\boldsymbol{\Lambda}}, \tag{2-22}$$

where $\bar{\boldsymbol{\Lambda}} = \mathrm{diag}([\bar{\lambda}_1, \ldots, \bar{\lambda}_M])$ contains the eigenvalues and $\bar{\mathbf{U}} = [\bar{\mathbf{u}}_1, \ldots, \bar{\mathbf{u}}_M] \in \mathbb{R}^{M\times M}$ the corresponding eigenvectors. According to Mercer's Theorem [12], the kernel function $K(\mathbf{x}, \mathbf{x}')$ can be expressed as

$$K(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n_{\mathcal{H}}} \lambda_i \phi_i(\mathbf{x})\phi_i(\mathbf{x}'), \tag{2-23}$$

where $n_{\mathcal{H}} \leq \infty$ is the size of the high dimensional feature space $\mathcal{H}$, $\lambda_1 \geq \lambda_2 \geq \ldots \geq 0$ the eigenvalues, and ϕ_i the eigenfunctions that yield

$$\int K(\mathbf{x}, \mathbf{x}')\phi_i(\mathbf{x})p(\mathbf{x})d\mathbf{x} = \lambda_i\phi_i(\mathbf{x}'), \tag{2-24}$$

with $p(\mathbf{x})$ the probability density of $\mathbf{x}$. To approximate (2-24) with M samples, we replace the integral over $p(\mathbf{x})$ by an empirical average [4] to obtain

$$\frac{1}{M}\sum_{k=1}^{M} K(\mathbf{x}_k, \mathbf{x}')\phi_i(\mathbf{x}_k) \approx \lambda_i \phi_i(\mathbf{x}'). \tag{2-25}$$

By combining (2-22) and (2-25) we arrive at the following approximations:

$$\phi_i(\mathbf{x}_k) \approx \sqrt{M}\bar{u}_{k,i}, \quad \lambda_i \approx \frac{1}{M}\bar{\lambda}_i, \tag{2-26}$$

where $\bar{u}_{k,i}$ denotes the element (k,i) of the matrix $\bar{\mathbf{U}}$. By plugging (2-26) back into (2-25) we obtain the *Nyström approximation* to the i-th eigenfunction at point $\mathbf{x}'$,

$$\hat{\phi}_i(\mathbf{x}') = \frac{\sqrt{M}}{\bar{\lambda}_i}\sum_{k=1}^{M} K(\mathbf{x}_k, \mathbf{x}')\bar{u}_{k,i}. \tag{2-27}$$

Furthermore, as explained in [4] the eigenvalues and eigenvectors of the complete kernel matrix $\mathbf{\Omega}$ can be approximated based on (2-22) as follows

$$\tilde{\lambda}_i = \frac{N}{M}\bar{\lambda}_i, \quad \tilde{\mathbf{u}}_i = \sqrt{\frac{N}{M}}\frac{1}{\bar{\lambda}_i}\mathbf{\Omega}_{(N,M)}\bar{\mathbf{u}}_i, \tag{2-28}$$

where $\mathbf{\Omega}_{(N,M)}$ is the $N \times M$ block matrix of $\mathbf{\Omega}$. Therefore, the linear system of the form $(\mathbf{\Omega} + \mathbf{I}_N/\gamma)\mathbf{z} = \mathbf{d}$ can be solved by applying the Sherman-Morrison-Woodbury formula [4]:

$$\mathbf{z} = \gamma\left(\mathbf{d} - \tilde{\mathbf{U}}\left(\frac{1}{\gamma}\mathbf{I} + \tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^{\mathsf{T}}\tilde{\mathbf{U}}\right)^{-1}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^{\mathsf{T}}\mathbf{d}\right), \tag{2-29}$$

where $\tilde{\mathbf{\Lambda}}$ and $\tilde{\mathbf{U}}$ are calculated from (2-28).

The Nyström method involves a memory requirements of $\mathcal{O}(MN)$ and a computational complexity of $\mathcal{O}(M^2N)$ [4], which can be significantly smaller than iterative methods when $M \ll N$.

2-3-2 Fixed Size LS-SVMs

Estimation in primal space

Since the linear KKT system scales with the number of data points, it would be advantageous for large-scale datasets if the problems could be solved in the primal weight space. To do that, we need to find a meaningful estimate of the nonlinear feature map $\varphi(\cdot)$, which is in principle implicitly determined by the kernel trick.

On the basis of the Nyström approximation (2-27) with M chosen points, at any point $\mathbf{x}'$, the approximate map $\hat{\varphi}_i(\cdot) : \mathbb{R}^n \to \mathbb{R}$, with $\hat{\varphi}(\cdot) = [\hat{\varphi}_1(\cdot), \ldots, \hat{\varphi}_M(\cdot)]^{\mathsf{T}}$, can be computed by

$$\hat{\varphi}_i(\mathbf{x}') = \sqrt{\tilde{\lambda}_i}\hat{\phi}_i(\mathbf{x}') = \frac{\sqrt{M}}{\sqrt{\tilde{\lambda}_i}}\sum_{k=1}^{M} K(\mathbf{x}_k, \mathbf{x}')\bar{u}_{k,i}. \tag{2-30}$$

The prediction model in primal space then takes the form

$$\begin{aligned}\hat{y}(\mathbf{x}) &= \operatorname{sign}\left(\tilde{\mathbf{w}}^{\mathsf{T}}\hat{\varphi}(\mathbf{x}) + b\right) \\ &= \operatorname{sign}\left(\sum_{i=1}^{M}\tilde{w}_i\frac{\sqrt{M}}{\sqrt{\tilde{\lambda}_i}}\sum_{k=1}^{M}K(\mathbf{x}_k,\mathbf{x})\bar{u}_{k,i} + b\right).\end{aligned} \tag{2-31}$$

With a finite dimensional approximation to the feature map $\hat{\varphi}(\cdot)$, the following *ridge regression* problem can be solved in primal space with unknowns $\tilde{\mathbf{w}} \in \mathbb{R}^M, b \in \mathbb{R}$:

$$\min_{\tilde{\mathbf{w}},b}\frac{1}{2}\tilde{\mathbf{w}}^{\mathsf{T}}\tilde{\mathbf{w}} + \gamma\frac{1}{2}\sum_{k=1}^{N}\left[1 - y_k\left(\tilde{\mathbf{w}}^{\mathsf{T}}\hat{\varphi}(\mathbf{x}_k) + b\right)\right]^2. \tag{2-32}$$

Active selection of support vectors

In order to make a more suitable selection of the support vectors instead of a random selection, an entropy based method is applied in the FS-LSSVM context. The method aims at maximizing the Quadratic Rényi Entropy, defined as

$$H_R = -\log\int p(\mathbf{x})^2 d\mathbf{x}.$$

In practice, an estimator based upon M samples $\{\mathbf{x}_1,\ldots,\mathbf{x}_M\}$ is given by

$$\hat{H}_R(\{\mathbf{x}_1,\ldots,\mathbf{x}_M\}) = -\log\frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M}K_p(\mathbf{x}_i,\mathbf{x}_j), \tag{2-33}$$

where $K_p(\cdot,\cdot)$ is a normalized kernel with respect to density estimation [21].

The selection method is explained as follows [2]. Given a training set $T_N = \{\mathbf{x}_k, y_k\}_{k=1}^N$ with N data points, we choose at random a working set W_M of size $M (M \ll N)$ containing candidate support vectors. In the working set W_M, a point $\mathbf{x}^-$ is randomly selected and replaced by a randomly selected point $\mathbf{x}^+$ from T_N if the new point $\mathbf{x}^+$ improves the Quadratic Rényi Entropy (2-33). The selection procedure stops if the change in entropy value is smaller than a given threshold or the number of iterations is exceeded.

After obtaining M representative support vectors, we can estimate the unknowns in primal space as discussed before. An optimized FS-LSSVM model is proposed in [21] with a fast cross-validation method.

The focus of our study is on solving large-scale problems. We will show in Chapter 3 that Tensor Networks provide an efficient way for making low-rank approximations to reduce the storage requirements without losing the accuracy.

Chapter 3

Tensor Networks

In this Chapter, we provide all the information we need about Tensor Networks to establish our model in Chapter 4. A brief introduction to tensors is first given in Section 3-1, including the required tensor operations and properties. We then discuss the Tensor Train (TT) decomposition in Section 3-2, which will be used to build and solve large-scale LS-SVM problems. We show that vectors and matrices can be represented in the TT format and many operations can be efficiently applied. In Section 3-3, several methods for solving linear systems are presented. Finally, in Section 3-4 the introduced algorithms are adapted to perform fast matrix inversion, which forms the basis for our proposed method.

3-1 Tensor Basics

Tensors are multi-dimensional arrays that generalize the notions of vectors and matrices to higher orders. A d-way or d-th order tensor is denoted by $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, and each of its elements $a_{i_1,i_2,\ldots,i_d}$ is determined by d indices (modes). A 1-way tensor is a vector and a 2-way tensor is a matrix. The numbers $I_1, I_2, \cdots, I_d$ are called the dimensions of the tensor. Subtensors are formed when a subset of tensor indices is fixed. Of particular interest are *fibers* which are vectors obtained by fixing every tensor index but one, and *slices* which are two-dimensional sections (matrices) of a tensor, obtained by fixing all the indices but two.

The description of tensors and their computation is facilitated by the use of diagrammatic notations borrowed from physics and quantum chemistry [22]. In this notation, a tensor is represented by a circle while the order of it is determined by the number of connecting edges. The size of each mode can be noted next to the corresponding edge. Figure 3-1 shows the notations of a scalar, a vector, a matrix, and a 3-way tensor. Some of the required definitions and operations about tensors are provided below.

Definition 3.1 (Kronecker product [23])**.** The Kronecker product of two matrices $\mathbf{X} \in \mathbb{R}^{M \times N}$ and $\mathbf{Y} \in \mathbb{R}^{P \times Q}$ is denoted by $\mathbf{X} \otimes \mathbf{Y}$. The result is a matrix of size $MP \times NQ$ and

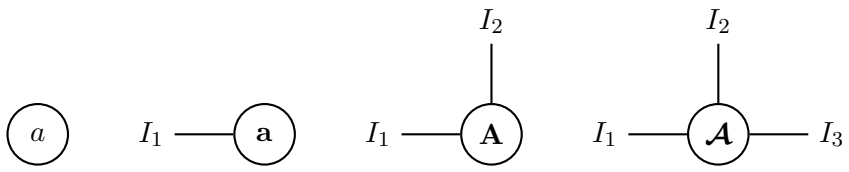


Figure 3-1: Diagrammatic notations of tensors. From left to right: a scalar $a \in \mathbb{R}$, a vector $\mathbf{a} \in \mathbb{R}^{I_1}$, a matrix $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2}$, and a 3-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.

defined by

$$\mathbf{X} \otimes \mathbf{Y} = \begin{bmatrix} x_{1,1}\mathbf{Y} & x_{1,2}\mathbf{Y} & \cdots & x_{1,N}\mathbf{Y} \\ x_{2,1}\mathbf{Y} & x_{2,2}\mathbf{Y} & \cdots & x_{2,N}\mathbf{Y} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M,1}\mathbf{Y} & x_{M,2}\mathbf{Y} & \cdots & x_{M,N}\mathbf{Y} \end{bmatrix}.$$

Definition 3.2 (Hadamard product [23])**.** The Hadamard product is the element-wise matrix product. Given matrices $\mathbf{X}$ and $\mathbf{Y}$ both of size $M \times N$, their Hadamard product is denoted by $\mathbf{X} \odot \mathbf{Y}$. The result is also of size $M \times N$ and defined by

$$\mathbf{X} \odot \mathbf{Y} = \begin{bmatrix} x_{1,1}y_{1,1} & x_{1,2}y_{1,2} & \cdots & x_{1,N}y_{1,N} \\ x_{2,1}y_{2,1} & x_{2,2}y_{2,2} & \cdots & x_{2,N}y_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M,1}y_{M,1} & x_{M,2}y_{M,2} & \cdots & x_{M,N}y_{M,N} \end{bmatrix}.$$

Definition 3.3 (Cubical and symmetric tensors [23])**.** A tensor is called cubical if every mode is the same size, i.e., $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I \times I \times \cdots \times I}$. A cubical tensor is called symmetric if its elements remain constant under any permutation of the indices.

Definition 3.4 (Tensor norm [23])**.** The (Frobenius) norm of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ is analogous to the matrix Frobenius norm, defined as the square root of the sum of the squares of all its elements, i.e.,

$$\|\boldsymbol{\mathcal{A}}\|_F = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_d=1}^{I_d} a_{i_1,i_2,\ldots,i_d}}.$$

Definition 3.5 (Inner product [23])**.** The inner product of two same-sized tensors $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ is the sum of the products of their elements, i.e.,

$$\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_d=1}^{I_d} a_{i_1,i_2,\ldots,i_d} b_{i_1,i_2,\ldots,i_d}.$$

It follows immediately that $\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{A}} \rangle = \|\boldsymbol{\mathcal{A}}\|_F^2$.

Reshaping is an essential operation when working with high-dimensional tensors. The dimensions of a tensor can be combined or separated to form a new tensor as long as the number of elements is consistent. We adopt the MATLAB® `reshape` operator defined as follows.

Definition 3.6 (Reshaping [24])**.** The operator "reshape($\boldsymbol{\mathcal{A}}, [I_1, I_2, \ldots, I_d]$)" reorders a given tensor $\boldsymbol{\mathcal{A}}$ into a d-way tensor with dimensions $I_1 \times I_2 \times \cdots \times I_d$. The total number of elements of $\boldsymbol{\mathcal{A}}$ should be the same as the products $I_1 I_2 \cdots I_d$.

The most common reshaping is the *matricization* (or *unfolding*), which reorders the elements of a tensor into a matrix. We introduce a canonical form of matricization in the following definition, which can be easily performed via reshaping.

Definition 3.7 (Unfolding matrices [25])**.** A d-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ has $d-1$ unfolding matrices. The k-th unfolding matrix of $\boldsymbol{\mathcal{A}}$, denoted by $\mathbf{A}_{[k]}$, is defined by

$$\mathbf{A}_{[k]} = \text{reshape}\left(\boldsymbol{\mathcal{A}}, \left[\prod_{i=1}^{k} I_i, \prod_{j=k+1}^{d} I_j\right]\right), \quad k = 1, \ldots, d-1,$$

and the size of $\mathbf{A}_{[k]}$ is $(I_1 \cdots I_k) \times (I_{k+1} \cdots I_d)$.

The following example is of help to understand the reshaping operator and unfolding matrices. Let a 3-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{3\times4\times2}$ be

$$\mathbf{A}_{:,:,1} = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}, \quad \mathbf{A}_{:,:,2} = \begin{bmatrix} 13 & 16 & 19 & 22 \\ 14 & 17 & 20 & 23 \\ 15 & 18 & 21 & 24 \end{bmatrix}. \tag{3-1}$$

The first and second unfolding matrices, $\mathbf{A}_{[1]} \in \mathbb{R}^{3\times8}$ and $\mathbf{A}_{[2]} \in \mathbb{R}^{12\times2}$, are given by

$$\mathbf{A}_{[1]} = \text{reshape}(\boldsymbol{\mathcal{A}}, [3, 8]) = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 \\ 2 & 5 & 8 & 11 & 14 & 17 & 20 & 23 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 \end{bmatrix},$$

$$\mathbf{A}_{[2]} = \text{reshape}(\boldsymbol{\mathcal{A}}, [12, 2]) = \begin{bmatrix} 1 & 13 \\ 2 & 14 \\ \vdots & \vdots \\ 12 & 24 \end{bmatrix}.$$

Another important reshaping of a tensor is its *vectorization*. The vectorization of a tensor $\boldsymbol{\mathcal{A}}$, denoted by $\text{vec}(\boldsymbol{\mathcal{A}})$, rearranges all its entries into one column vector. For the example above, the vectorization is

$$\text{vec}(\boldsymbol{\mathcal{A}}) = \text{reshape}(\boldsymbol{\mathcal{A}}, [24, 1]) = \begin{bmatrix} 1 & 2 & \cdots & 24 \end{bmatrix}^{\mathsf{T}}.$$

In addition to reshaping, there is another important operation when doing computations with higher-order tensors, called the *permutation*. We again adopt the MATLAB® `permute` operator given in the following definition.

Definition 3.8 (Permutation [24])**.** The operator "permute($\boldsymbol{\mathcal{A}}, \mathbf{p}$)" rearranges the indices of $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ in the order specified by the vector $\mathbf{p}$. The resulting tensor has the same values of $\boldsymbol{\mathcal{A}}$ but the order of the subscripts when accessing any particular element is rearranged by $\mathbf{p}$. All the elements of $\mathbf{p}$ must be unique, positive integers from 1 to d.

The transpose of a matrix $\mathbf{B}$ can be expressed as "permute($\mathbf{B}, [2, 1]$)". For a 3-way tensor $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, "permute($\boldsymbol{\mathcal{C}}, [1, 3, 2]$)" results in a 3-way tensor $\boldsymbol{\mathcal{D}} \in \mathbb{R}^{I_1 \times I_3 \times I_2}$ with $c_{i_1,i_2,i_3} = d_{i_1,i_3,i_2}$.

Tensors can be multiplied together like matrices and vectors. The definition of *k-mode product* is given below, where a tensor is multiplied by a matrix at the k-th mode.

Definition 3.9 (k-mode product [23])**.** The k-mode product of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_k \times \cdots \times I_d}$ with a matrix $\mathbf{U} \in \mathbb{R}^{M \times I_k}$ is denoted by $\boldsymbol{\mathcal{A}} \times_k \mathbf{U}$. The element of the resulting tensor $\boldsymbol{\mathcal{B}}$ is defined as

$$b_{i_1,\ldots,i_{k-1},m,i_{k+1},\ldots,i_d} = \sum_{i_k=1}^{I_k} a_{i_1,\ldots,i_k,\ldots,i_d} u_{m,i_k},$$

and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_{k-1} \times M \times I_{k+1} \times \cdots \times I_d}$.

The tenser-matrix multiplication is generalized to tensor-tensor version through the operation called the *tensor contraction*. The contraction is done via the summation of the products over equal-sized indices. For instance, the contraction of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and a tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_3 \times I_4 \times I_5}$ over the dimension of size I_3 is denoted by $\boldsymbol{\mathcal{A}} \times_3^1 \boldsymbol{\mathcal{B}}$ [25], which yields a tensor $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{I_1 \times I_2 \times I_4 \times I_5}$ whose entries are given by

$$c_{i_1,i_2,i_4,i_5} = \sum_{i_3=1}^{I_3} a_{i_1,i_2,i_3} b_{i_3,i_4,i_5}.$$

The notation can be simplified as $\boldsymbol{\mathcal{A}} \times^1 \boldsymbol{\mathcal{B}}$ when the last index of the first tensor is contracted. The graphical notation of (3-2) is depicted in Figure 3-2:

$$\boldsymbol{\mathcal{C}} = (\boldsymbol{\mathcal{A}} \times^1 \boldsymbol{\mathcal{B}}) \times_4 \mathbf{U} \ \in \mathbb{R}^{I_1 \times I_2 \times I_4 \times M}. \tag{3-2}$$

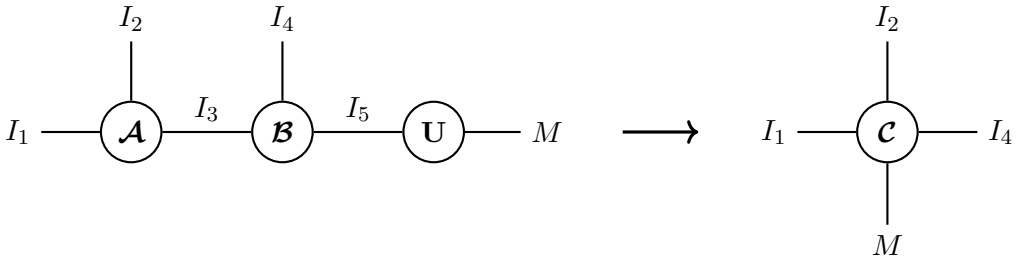


Figure 3-2: Diagrammatic notation of (3-2), showing the tensor contraction and k-mode product.

3-2 Tensor Train Decomposition

Tensors are a natural way of representing data in many fields such as quantum molecular dynamics [26], financial modelling [27] and scientific computing [28]. However, tensors cannot be handled by standard numerical methods due to the *curse of dimensionality*, since the required memory and amount of operations grow exponentially with the number of dimensions. Tensor Networks or tensor decompositions provide special structures that break down higher-order tensors into a set of sparsely interconnected lower-order core tensors. In this way, large-scale data can be approximately represented in highly compressed formats.

There are three most popular tensor decomposition methods: the canonical polyadic (CP) decomposition [29, 30], the Tucker decomposition [31], and the Tensor Train (TT) decomposition

[6]. The CP format decomposes a given tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ into d matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}$. The elements of the tensor are defined by

$$y_{i_1,i_2,\ldots,i_d} = \sum_{r=1}^{R} a^{(1)}_{i_1,r} a^{(2)}_{i_2,r} \cdots a^{(d)}_{i_d,r}, \tag{3-3}$$

where R is the (canonical) tensor rank and matrices $\mathbf{A}^{(k)} \in \mathbb{R}^{I_k \times R}$ are canonical factors. The CP format provides an effective way to represent large-sized tensors. Assume that $I = \max\{I_1, I_2, \ldots, I_d\}$, the storage complexity of a d-way tensor $\boldsymbol{\mathcal{Y}}$ in the CP format is $\mathcal{O}(dIR)$, while the original storage complexity is $\mathcal{O}(I^d)$. However, the computation of the tensor rank R is an NP-hard problem [32], and there is no robust algorithm for finding the best or quasi-best approximation.

The Tucker decomposition of a d-way tensor $\boldsymbol{\mathcal{Y}}$ is given by

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_d \mathbf{A}^{(d)}, \tag{3-4}$$

or element-wise by

$$y_{i_1,i_2,\ldots,i_d} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_d=1}^{R_d} g_{r_1,r_2,\ldots,r_d} a^{(1)}_{i_1,r_1} a^{(2)}_{i_2,r_2} \cdots a^{(d)}_{i_d,r_d}.$$

Here $\mathbf{A}^{(k)} \in \mathbb{R}^{I_k \times R_k}$ are the factor matrices, and the core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_d}$ can be thought of as a compressed version of $\boldsymbol{\mathcal{Y}}$. The Tucker format is numerically stable but its storage complexity is $\mathcal{O}(dIR + R^d)$ given that $R = \max\{R_1, R_2, \ldots, R_d\}$, which still grows exponentially with d. Hence, it is only suitable for tensors with low dimensions, especially for the three-dimensional case.

The TT format combines the advantages of the CP and Tucker formats. On the one hand, it does not have an intrinsic exponential dependence on the number of dimensions d. On the other hand, it is numerically stable in the sense that it can be computed by sequentially applying the Singular Value Decomposition (SVD) to unfolding matrices [6].

3-2-1 Definition

The TT decomposition, also known as the Matrix Product State (MPS) [22], represents a d-way tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ in terms of d 3-way tensors:

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{G}}^{(1)} \times^1 \boldsymbol{\mathcal{G}}^{(2)} \times^1 \cdots \times^1 \boldsymbol{\mathcal{G}}^{(d)}, \tag{3-5}$$

where 3-way tensors[1] $\boldsymbol{\mathcal{G}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$ are called the TT cores, interconnected via contractions. Parameters R_k are TT ranks and by definition $R_0 = R_d = 1$. Each element of $\boldsymbol{\mathcal{Y}}$ is determined by

$$y_{i_1,i_2,\ldots,i_d} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{d-1}=1}^{R_{d-1}} g^{(1)}_{1,i_1,r_1} g^{(2)}_{r_1,i_2,r_2} \cdots g^{(d)}_{r_{d-1},i_d,1}. \tag{3-6}$$

[1] The cores $\boldsymbol{\mathcal{G}}^{(1)}$ and $\boldsymbol{\mathcal{G}}^{(d)}$ are actually 2-way tensors (matrices), but for a uniform representation they are treated as 3-way tensors of sizes $1 \times I_1 \times R_1$ and $R_{d-1} \times I_d \times 1$, respectively.

The TT decomposition of a 3-way tensor $\boldsymbol{\mathcal{A}}$ is illustrated in Figure 3-3, where $R_0 = R_3 = 1$ are omitted for simplicity. The storage complexity of a d-way tensor in the TT format is $\mathcal{O}(dIR^2)$. The TT decomposition with small TT ranks can thus significantly reduce the memory requirements. In order to make the notation clear and concise, from now on a tensor in the TT format will be called a TT.

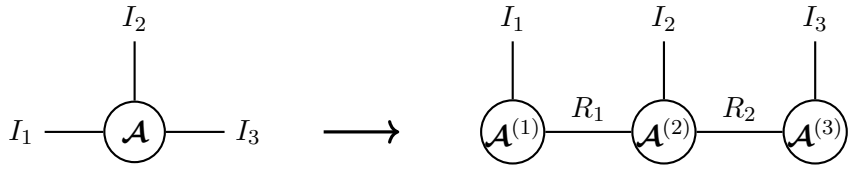


Figure 3-3: Diagrammatic notation of TT decomposition of a 3-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.

The TT ranks of any given tensor are upper bounded by the ranks of its unfolding matrices.

Theorem 3.1 (Upper bound of TT ranks [6])**.** For any tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ there exists a TT decomposition with TT ranks

$$R_k \leq \operatorname{rank}(\mathbf{Y}_{[k]}).$$

where $\mathbf{Y}_{[k]}$ is the k-th unfolding matrix of $\boldsymbol{\mathcal{Y}}$.

When TT ranks satisfy $R_k = \operatorname{rank}(\mathbf{Y}_{[k]})$, the tensor is exactly represented by the TT decomposition. Low-rank approximations are obtained if $R_k < \operatorname{rank}(\mathbf{Y}_{[k]})$, which can be done by the use of truncated SVD.

Definition 3.10 (δ-truncated SVD [5])**.** Given the SVD of an $m \times n$ matrix ($m \geq n$) $\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\mathsf{T}$, where $\mathbf{U} \in \mathbb{R}^{m \times n}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ have orthonormal columns, and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ is a diagonal matrix of the singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ with the fact that $\|\mathbf{A}\|_F^2 = \|\boldsymbol{\Sigma}\|_F^2 = \sum_{i=1}^n \sigma_i^2$. Then an approximation $\tilde{\mathbf{A}}$ with rank $r < n$ can be obtained by

$$\tilde{\mathbf{A}} = \mathbf{U}_{:,1:r}\boldsymbol{\Sigma}_{1:r,1:r}\mathbf{V}_{:,1:r}^\mathsf{T}.$$

The δ-truncated SVD of $\mathbf{A}$ gives the approximation with the minimum rank r satisfying

$$\|\mathbf{A} - \tilde{\mathbf{A}}\|_F = \sqrt{\sum_{i=r+1}^{n} \sigma_i^2} \leq \delta.$$

The corresponding rank is called δ-rank of matrix $\mathbf{A}$, denoted by $\operatorname{rank}_\delta(\mathbf{A})$.

We are now able to find a TT approximation of a given tensor by sequentially applying $(d-1)$ truncated SVDs to its unfolding matrices. The values at which these SVDs are truncated are indeed the TT ranks. This algorithm is called the TT-SVD and its computational complexity is $\mathcal{O}(I^d R^2)$ [7].

Theorem 3.2 (Approximation Error of TT-SVD [6])**.** Given a tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$, the TT-SVD computes a TT $\tilde{\boldsymbol{\mathcal{Y}}}$ with TT ranks R_k. The total approximation error yields

$$\|\boldsymbol{\mathcal{Y}} - \tilde{\boldsymbol{\mathcal{Y}}}\|_F^2 \leq \sum_{k=1}^{d-1} \sum_{i=R_k+1}^{\min\{I_l, I_r\}} \sigma_i^2(\mathbf{Y}_{[k]}), \quad I_l = I_k R_{k-1},\ I_r = \prod_{i=k+1}^{d} I_i,$$

where $\sigma_i(\mathbf{Y}_{[k]})$ denotes the i-th largest singular value of the k-th unfolding matrix.

Algorithm 2: TT-SVD [6]

Input: tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$, prescribed accuracy ε
Output: TT $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with TT cores $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$ such that $\|\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{B}}\|_F \leq \varepsilon \|\boldsymbol{\mathcal{A}}\|_F$

Compute truncation parameter $\delta \leftarrow \frac{\varepsilon}{\sqrt{d-1}} \|\boldsymbol{\mathcal{A}}\|_F$
Sizes of unfolding matrix: $I_l = I_1, I_r = \prod_{i=2}^{d} I_i$
First unfolding matrix $\mathbf{M} \leftarrow \text{reshape}(\boldsymbol{\mathcal{A}}, [I_l, I_r])$
Compute δ-truncated SVD[2]: $\mathbf{M} \approx \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathsf{T}}$, $R_1 \leftarrow \text{rank}(\boldsymbol{\Sigma})$
First TT core $\boldsymbol{\mathcal{B}}^{(1)} \leftarrow \text{reshape}(\mathbf{U}, [1, I_1, R_1])$
Compute $\mathbf{M} \leftarrow \boldsymbol{\Sigma}\mathbf{V}^{\mathsf{T}}$
for $k = 2, 3, \ldots, d-1$ **do**
    Redefine the sizes: $I_l \leftarrow I_k R_{k-1}, I_r \leftarrow I_r / I_k$
    k-th unfolding matrix $\mathbf{M} \leftarrow \text{reshape}(\mathbf{M}, [I_l, I_r])$
    Compute δ-truncated SVD: $\mathbf{M} \approx \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathsf{T}}$, $R_k \leftarrow \text{rank}(\boldsymbol{\Sigma})$
    k-th TT core $\boldsymbol{\mathcal{B}}^{(k)} \leftarrow \text{reshape}(\mathbf{U}, [R_{k-1}, I_k, R_k])$
    Recompute $\mathbf{M} \leftarrow \boldsymbol{\Sigma}\mathbf{V}^{\mathsf{T}}$
end
Last TT core $\boldsymbol{\mathcal{B}}^{(d)} \leftarrow \mathbf{M}$

From Theorem 3.2 it follows that we have two options to transform a tensor $\boldsymbol{\mathcal{Y}}$ into the TT format. Firstly, we can prescribe the relative accuracy ε such that $\|\boldsymbol{\mathcal{Y}} - \tilde{\boldsymbol{\mathcal{Y}}}\|_F \leq \varepsilon \|\boldsymbol{\mathcal{Y}}\|_F$. To achieve that, the truncated error of each unfolding matrix should not exceed the threshold value $\frac{\varepsilon}{\sqrt{d-1}} \|\boldsymbol{\mathcal{Y}}\|_F$. This algorithm is shown in Algorithm 2. Secondly, we can just set a maximal bound for each TT rank and estimate the approximation error using Theorem 3.2. This way provides us with full control over the resulting TT structure, but it cannot guarantee anything about the relative error. In practice, these two options can even be combined together. Generally, there is always a trade-off between the accuracy and the memory requirements since the TT format is efficient in terms of storage only if the TT ranks stay small.

An example of image compression is provided in Figure 3-4, where a 512×512 grayscale image is first converted into a 9-way cubical tensor of dimensions 4 and then approximated in the TT format by the TT-SVD with a relative accuracy $\varepsilon = 0.1$. The total number of elements stored in the TT format is 22356, which reduced 91.5% of storage requirements.

3-2-2 Vectors and matrices

Given a vector $\mathbf{y} \in \mathbb{R}^N$, we can reshape it into a d-way tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ such that $N = I_1 I_2 \cdots I_d$. This tensor can then be transformed into the TT format. By doing so, each element of the original vector is determined by a multi-index instead of just one. For example, suppose we have a vector $\mathbf{a} \in \mathbb{R}^8$ that can be reshaped to a 3-way tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{2 \times 2 \times 2}$. Then

[2] Given a tensor $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$, $\text{size}(\boldsymbol{\mathcal{B}}, k) = I_k$.

(a) Original image. (b) Image approximated by TT-SVD with $\varepsilon = 0.1$.

Figure 3-4: Example of image compression using TT decomposition.

the corresponding elements of $\mathbf{a}$ in tensor are given by

$$a_1 \Leftrightarrow b_{1,1,1}, \quad a_2 \Leftrightarrow b_{2,1,1}, \quad a_3 \Leftrightarrow b_{1,2,1}, \quad a_4 \Leftrightarrow b_{2,2,1},$$
$$a_5 \Leftrightarrow b_{1,1,2}, \quad a_6 \Leftrightarrow b_{2,1,2}, \quad a_7 \Leftrightarrow b_{1,2,2}, \quad a_8 \Leftrightarrow b_{2,2,2}.$$

Now consider the case of matrices. Each element of a matrix is determined by two indices, thus matrices can be multiplied by vectors (or matrices) with matched dimensions. To achieve that, we now introduce an adapted version for matrices called the Tensor Train matrix (TTm) decomposition [7], also known as the Matrix Product Operator (MPO) [22].

A matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ can be treated as a d-level matrix with rows indexed by a multi-index $(i_1, i_2, \ldots, i_d)$ and columns by $(j_1, j_2, \ldots, j_d)$. This d-level matrix can be seen as a $2d$-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times J_1 \times I_2 \times J_2 \times \cdots \times I_d \times J_d}$ such that $M = I_1 I_2 \cdots I_d$ and $N = J_1 J_2 \cdots J_d$. The TTm decomposition represents a $2d$-way tensor in terms of d 4-way tensors as

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{G}}^{(1)} \times^1 \boldsymbol{\mathcal{G}}^{(2)} \times^1 \cdots \times^1 \boldsymbol{\mathcal{G}}^{(d)}, \tag{3-7}$$

where $\boldsymbol{\mathcal{G}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}$ are called the TTm cores and R_k the TTm ranks with $R_0 = R_d = 1$. Each entry of $\boldsymbol{\mathcal{A}}$ is determined by

$$a_{i_1,j_1,i_2,j_2,\ldots,i_d,j_d} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{d-1}=1}^{R_{d-1}} g^{(1)}_{1,i_1,j_1,r_1} g^{(2)}_{r_1,i_2,j_2,r_2} \cdots g^{(d)}_{r_{d-1},i_d,j_d,1}. \tag{3-8}$$

If all TT ranks R_k are equal to 1, then TT cores are basically matrices, $\mathbf{G}^{(k)} \in \mathbb{R}^{I_k \times J_k}$, and (3-7) reduces to the Kronecker product:

$$\mathbf{A} = \mathbf{G}^{(d)} \otimes \mathbf{G}^{(d-1)} \otimes \cdots \otimes \mathbf{G}^{(1)}. \tag{3-9}$$

Definition 3.11 (Operator τ and sub-trains [33])**.** Given a vector $\mathbf{x} \in \mathbb{R}^{I_1 \cdots I_d}$ and a matrix $\mathbf{A} \in \mathbb{R}^{(I_1 \cdots I_d) \times (J_1 \cdots J_d)}$ in the TT(m) format with cores $\boldsymbol{\mathcal{X}}^{(k)} \in R^{R_{k-1} \times I_k \times R_k}$ and $\boldsymbol{\mathcal{A}}^{(k)} \in R^{R_{k-1} \times I_k \times J_k \times R_k}$, respectively, the operator τ allows us to write them as

$$\mathbf{x} = \tau(\boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(d)}), \quad \mathbf{A} = \tau(\boldsymbol{\mathcal{A}}^{(1)}, \ldots, \boldsymbol{\mathcal{A}}^{(d)}). \tag{3-10}$$

With notation τ, it also enables us to merge a subset of TT(m) cores into a 3-way or 4-way tensor called a *sub-train*:

$$\begin{aligned}\boldsymbol{\mathcal{X}}^{(k:p)} &= \tau(\boldsymbol{\mathcal{X}}^{(k)}, \dots, \boldsymbol{\mathcal{X}}^{(p)}) \in \mathbb{R}^{R_{k-1} \times (I_k \cdots I_p) \times R_p}, \\ \boldsymbol{\mathcal{A}}^{(k:p)} &= \tau(\boldsymbol{\mathcal{A}}^{(k)}, \dots, \boldsymbol{\mathcal{A}}^{(p)}) \in \mathbb{R}^{R_{k-1} \times (I_k \cdots I_p) \times (J_k \cdots J_p) \times R_p},\end{aligned} \tag{3-11}$$

for $1 \leq k \leq p \leq d$. For boundary cases of vectors, sub-trains become 2-way tensors called the *interface matrices*:

$$\begin{aligned}\mathbf{X}^{(1:k)} &= \tau(\boldsymbol{\mathcal{X}}^{(1)}, \dots, \boldsymbol{\mathcal{X}}^{(k)}) \in \mathbb{R}^{(I_1 \cdots I_k) \times R_k}, \\ \mathbf{X}^{(k+1:d)} &= \tau(\boldsymbol{\mathcal{X}}^{(k+1)}, \dots, \boldsymbol{\mathcal{X}}^{(d)}) \in \mathbb{R}^{R_k \times (I_{k+1} \cdots I_d)}.\end{aligned} \tag{3-12}$$

The definition is extended to have $\mathbf{X}^{(1:0)} = \mathbf{X}^{(d+1:d)} = 1$.

For the sake of simplicity, a vector (matrix) represented in the TT (TTm) format will be called a TT-vector (TT-matrix). The graphical notation of a TT-matrix with 3 cores is shown in Figure 3-5. By merging the second and third modes of each TTm core, we can obtain a TT-vector with mixed row and column indices.

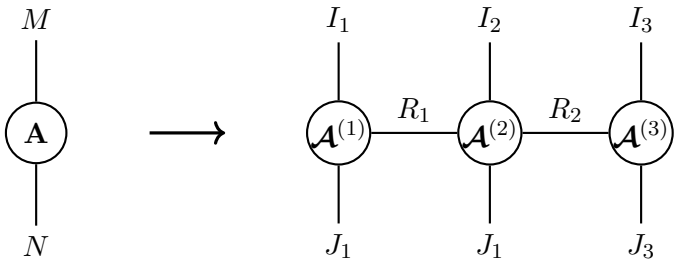


Figure 3-5: A matrix $\mathbf{A}$ is represented by the TTm decomposition, such that $M = I_1 I_2 I_3$ and $N = J_1 J_2 J_3$.

When creating higher-order tensors from vectors and matrices, known as the *tensorization*, each mode of resulting tensors in general has a small size (e.g 2, 3 or 4). In this way, a low-rank tensor approximation with high compression ratios can be obtained [7]. In addition to vectors and matrices, lower-order tensors can also be reshaped into higher-order ones to attain more efficient representations [34, 25].

3-2-3 Canonical form and Recompression

The TT format of a tensor is not unique, i.e., there are infinitely many TT decompositions that represent the same tensor. By applying orthogonalization to the TT cores, we can obtain a canonical form, which is essential for many algorithms [6, 35, 36, 37]. Given a TT $\boldsymbol{\mathcal{A}}$, each 3-way TT core $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$ has two unfolding matrices: the left unfolding matrix

$$\mathbf{A}_{\mathsf{L}}^{(k)} := \mathbf{A}_{[2]}^{(k)} \in \mathbb{R}^{R_{k-1} I_k \times R_k}, \tag{3-13}$$

and the right unfolding matrix

$$\mathbf{A}_{\mathsf{R}}^{(k)} := \mathbf{A}_{[1]}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k R_k}. \tag{3-14}$$

Algorithm 3: Left-orthogonalization [25]

Input: TT $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with TT cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$
Output: TT $\boldsymbol{\mathcal{A}}$ with all TT cores except $\boldsymbol{\mathcal{A}}^{(d)}$ are left-orthogonal
for $k = 1, 2, \ldots, d-1$ **do**
  QR decomposition of left unfolding $\mathbf{A}_{\mathsf{L}}^{(k)} = \mathbf{QR}$, $R_k \leftarrow \text{size}(\mathbf{Q}, 2)$
  Update core $\boldsymbol{\mathcal{A}}^{(k)} \leftarrow \text{reshape}(\mathbf{Q}, [R_{k-1}, I_k, R_k])$
  Replace $\mathbf{A}_{\mathsf{R}}^{(k+1)} \leftarrow \mathbf{R}\mathbf{A}_{\mathsf{R}}^{(k+1)}$
end

Algorithm 4: Right-orthogonalization [25]

Input: TT $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with TT cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$
Output: TT $\boldsymbol{\mathcal{A}}$ with all TT cores except $\boldsymbol{\mathcal{A}}^{(1)}$ are right-orthogonal
for $k = d, d-1 \ldots, 2$ **do**
  QR decomposition of right unfolding $\mathbf{A}_{\mathsf{R}}^{(k)\mathsf{T}} = \mathbf{QR}$, $R_{k-1} \leftarrow \text{size}(\mathbf{Q}, 2)$
  Update core $\boldsymbol{\mathcal{A}}^{(k)} \leftarrow \text{reshape}(\mathbf{Q}^{\mathsf{T}}, [R_{k-1}, I_k, R_k])$
  Replace $\mathbf{A}_{\mathsf{L}}^{(k-1)} \leftarrow \mathbf{A}_{\mathsf{L}}^{(k-1)}\mathbf{R}^{\mathsf{T}}$
end

Definition 3.12 (Left- and right-orthogonal TT cores [35])**.** A given TT core $\boldsymbol{\mathcal{A}}^{(k)}$ is left-orthogonal if its left unfolding matrix satisfies

$$\mathbf{A}_{\mathsf{L}}^{(k)\mathsf{T}}\mathbf{A}_{\mathsf{L}}^{(k)} = \mathbf{I}_{R_k}. \tag{3-15}$$

Similarly, $\boldsymbol{\mathcal{A}}^{(k)}$ is right-orthogonal if its right unfolding matrix satisfies

$$\mathbf{A}_{\mathsf{R}}^{(k)}\mathbf{A}_{\mathsf{R}}^{(k)\mathsf{T}} = \mathbf{I}_{R_{k-1}}. \tag{3-16}$$

We are now able to define the *site-k mixed-canonical form* of a TT [22].

Definition 3.13 (site-k mixed-canonical form [22])**.** A d-way TT is in site-k mixed-canonical form ($1 \leq k \leq d$), if all the cores $\boldsymbol{\mathcal{A}}^{(p)} (1 \leq p \leq k-1)$ are left-orthogonal and all the cores $\boldsymbol{\mathcal{A}}^{(q)} (k+1 \leq q \leq d)$ are right-orthogonal.

One advantage of a TT $\boldsymbol{\mathcal{A}}$ being in the site-k mixed-canonical form is that its Frobenius norm is equal to the norm of the k-th TT core, i.e., $\|\boldsymbol{\mathcal{A}}\|_F = \|\boldsymbol{\mathcal{A}}^{(k)}\|_F$. A TT can be orthogonalized efficiently by applying recursive QR decompositions. Recall that the QR decomposition [5] of a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ ($m \leq n$) is given by $\mathbf{X} = \mathbf{QR}$, where $\mathbf{Q} \in \mathbb{R}^{m \times m}$ has orthonormal columns and $\mathbf{R} \in \mathbb{R}^{m \times n}$ is an upper triangular matrix. Each core is replaced by the $\mathbf{Q}$-factor while the $\mathbf{R}$-factor is merged into the next core. The procedures of left- and right-orthogonalization are summarized in Algorithm 3 and 4, respectively.

A dense tensor can be converted into the TT format by the TT-SVD. Now consider the case when the tensor is already in the TT format, but with suboptimal TT ranks w.r.t the prescribed accuracy. In Section 3-2-4, we will show that many operations lead to the increase of TT ranks, which quickly makes the TT formats inefficient, thus a recompression is

Algorithm 5: TT rounding [6]

Input: TT $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with TT cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$, prescribed accuracy ε
Output: TT $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with optimized TT ranks such that $\|\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{B}}\|_F \leq \varepsilon \|\boldsymbol{\mathcal{A}}\|_F$

$\boldsymbol{\mathcal{A}} \leftarrow$ right-orthogonalize($\boldsymbol{\mathcal{A}}$)
Compute norm $\|\boldsymbol{\mathcal{A}}\|_F \leftarrow \text{vec}(\boldsymbol{\mathcal{A}}^{(1)})^\mathsf{T} \text{vec}(\boldsymbol{\mathcal{A}}^{(1)})$
Compute truncation parameter $\delta \leftarrow \frac{\varepsilon}{\sqrt{d-1}} \|\boldsymbol{\mathcal{A}}\|_F$
Initialize $\boldsymbol{\mathcal{B}} \leftarrow \boldsymbol{\mathcal{A}}$
for $k = 1, 2 \ldots, d-1$ **do**
    δ-truncated SVD of left unfolding: $\mathbf{B}_{\mathsf{L}}^{(k)} \approx \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\mathsf{T}$, $R_k \leftarrow \text{rank}(\boldsymbol{\Sigma})$
    Update core $\boldsymbol{\mathcal{B}}^{(k)} \leftarrow \text{reshape}(\mathbf{U}, [R_{k-1}, I_k, R_k])$
    Replace $\mathbf{B}_{\mathsf{R}}^{(k+1)} \leftarrow (\boldsymbol{\Sigma}\mathbf{V}^\mathsf{T})\mathbf{B}_{\mathsf{R}}^{(k+1)}$
end

necessary. To this end, the TT rounding algorithm can be used as a post-processing procedure to avoid rank growth.

The rounding algorithm consists of two steps. First, the given TT is orthogonalized from right to left to make it in the canonical form. Norm of the TT is now located in the first TT core. Next, we apply the truncated SVD to each of the unfolding matrices from left to right. This compression step can be done either by setting an upper bound for the relative error ε or by specifying the maximum TT ranks. Note that the TT rounding is mathematically the same as the TT-SVD. However, due to the use of the TT format, the SVDs are computed only for relatively small matrices, so that the complexity is greatly reduced. The whole procedure of the TT rounding algorithm is shown in Algorithm 5, which costs $\mathcal{O}(dIR^3)$ operations.

3-2-4 Basic operations

Multiplication by a scalar From the definition of the TT decomposition, it follows that multiplying a TT by a scalar amounts to multiplying one of the TT cores by the given value. If a TT is in the site-k mixed-canonical form, the orthogonality can be preserved by multiplying the scalar with the k-th TT core.

Consider two TTs with same dimensions: $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, for which the TT ranks are $\mathbf{r}(\boldsymbol{\mathcal{A}}) = [R_1, R_2, \ldots, R_{d-1}]$ and $\mathbf{r}(\boldsymbol{\mathcal{B}}) = [R'_1, R'_2, \ldots, R'_{d-1}]$, respectively. The addition and inner product between them can be performed as follows.

Addition The element-wise definition of the TT decomposition (3-6) of a d-way tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ can also be expressed as

$$y_{i_1, i_2, \ldots, i_d} = \mathbf{G}_{i_1}^{(1)} \mathbf{G}_{i_2}^{(2)} \cdots \mathbf{G}_{i_d}^{(d)}, \tag{3-17}$$

where $\mathbf{G}_{i_k}^{(k)} := \mathbf{G}_{:, i_k, :}^{(k)} \in \mathbb{R}^{R_{k-1} \times R_k}$ are matrix slices of the TT core $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$, with $i_k = 1, \ldots, I_k$. It follows that the summation of two TTs, $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{B}}$, has TT ranks $\mathbf{r}(\boldsymbol{\mathcal{C}}) = \mathbf{r}(\boldsymbol{\mathcal{A}}) + \mathbf{r}(\boldsymbol{\mathcal{B}})$ and can be computed by concatenating the corresponding TT cores as

$$\mathbf{C}_{i_k}^{(k)} = \begin{bmatrix} \mathbf{A}_{i_k}^{(k)} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{i_k}^{(k)} \end{bmatrix}, \quad k = 2, 3, \ldots, d-1, \tag{3-18}$$

Algorithm 6: TT inner product [6]

Input: TT $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with TT cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$,
TT $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with TT cores $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{R'_{k-1} \times I_k \times R'_k}$
Output: $z = \langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle \in \mathbb{R}$

```
Initialize Z = 1
for k = 1, 2 ..., d do
    M_k ← reshape(Z^T A_R^(k), [R'_{k-1} I_k, R_k])
    Z_k ← M_k^T B_L^(k)  ∈ R^{R_k × R'_k}
end
z ← Z_d
```

and for border cores we have

$$\mathbf{C}_{i_1}^{(1)} = \begin{bmatrix} \mathbf{A}_{i_1}^{(1)} & \mathbf{B}_{i_1}^{(1)} \end{bmatrix}, \quad \mathbf{C}_{i_d}^{(d)} = \begin{bmatrix} \mathbf{A}_{i_d}^{(d)} \\ \mathbf{B}_{i_d}^{(d)} \end{bmatrix}. \tag{3-19}$$

The sum of two TT-matrices can be performed in a similar way by replacing $\mathbf{A}_{i_k}^{(k)}$ with $\mathbf{A}_{i_k,j_k}^{(k)}$, obtained by fixing the second and third indices of the TTm cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}$.

Hadamard product The computation of the Hadamard (element-wise) product of two TTs of the same size, $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} \odot \boldsymbol{\mathcal{B}}$, can be performed by expressing the slices of the cores, $\boldsymbol{\mathcal{C}}^{(k)}$, as

$$\mathbf{C}_{i_k}^{(k)} = \mathbf{A}_{i_k}^{(k)} \otimes \mathbf{B}_{i_k}^{(k)}, \quad k = 1, \ldots, d. \tag{3-20}$$

The resulting TT has ranks $\mathbf{r}(\boldsymbol{\mathcal{C}}) = [R_1 R'_1, R_2 R'_2, \ldots, R_{d-1} R'_{d-1}]$. The computational complexity is $\mathcal{O}(dI(RR')^2)$.

Inner product The inner product of two TTs $z = \langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle$ is computed by successive contractions of TT cores from left to right. As an example, the inner product of two 3-way TTs is shown in Figure 3-6. Since there is no free edge, the result is a scalar. The full computation is given in Algorithm 6, with the computational complexity of $\mathcal{O}(dI(R^2R' + RR'^2))$. This provides us with a new way to compute the norm in the TT format: $\|\boldsymbol{\mathcal{A}}\|_F = \sqrt{\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{A}} \rangle}$.

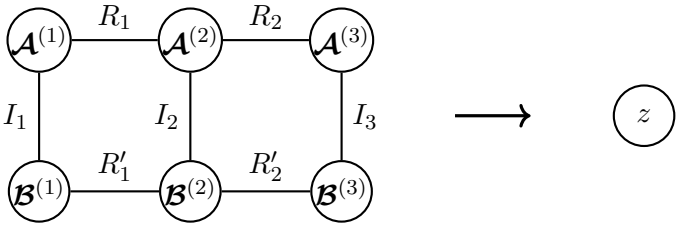


Figure 3-6: Diagrammatic notation of the inner product of two 3-way tensors in the TT format.

Now consider two TT-matrices $\mathbf{A} \in \mathbb{R}^{(M_1 \cdots M_d) \times (I_1 \cdots I_d)}$ with cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{P_{k-1} \times M_k \times I_k \times P_k}$ and $\mathbf{B} \in \mathbb{R}^{(I_1 \cdots I_d) \times (J_1 \cdots J_d)}$ with cores $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{Q_{k-1} \times I_k \times J_k \times Q_k}$, and a TT-vector $\mathbf{x} \in \mathbb{R}^{I_1 \cdots I_d}$ with cores $\boldsymbol{\mathcal{X}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$.

Algorithm 7: TT matrix-vector product [6]

Input: TT-matrix $\mathbf{A} \in \mathbb{R}^{(M_1 \cdots M_d) \times (I_1 \cdots I_d)}$ with cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{P_{k-1} \times M_k \times I_k \times P_k}$,
TT-vector $\mathbf{x} \in \mathbb{R}^{I_1 \cdots I_d}$ with cores $\boldsymbol{\mathcal{X}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$
Output: TT-vector $\mathbf{y} \in \mathbb{R}^{M_1 \cdots M_d}$ with cores $\boldsymbol{\mathcal{Y}}^{(k)} \in \mathbb{R}^{P_{k-1}R_{k-1} \times M_k \times P_k R_k}$ such that $\mathbf{y} = \mathbf{A}\mathbf{x}$

```
for k = 1, 2 ..., d do
    A_k ← reshape(permute(A^(k), [1, 2, 4, 3]), [P_{k-1} M_k P_k, I_k])
    X_k ← reshape(permute(X^(k), [2, 1, 3]), [I_k, R_{k-1} R_k])
    Y_k ← reshape(A_k X_k, [P_{k-1}, M_k, P_k, R_{k-1}, R_k])
    Y^(k) ← reshape(permute(Y_k, [1, 4, 2, 3, 5]), [P_{k-1} R_{k-1}, M_k, P_k R_k])
end
```

Matrix transpose From definition of the TTm decomposition the transpose of $\mathbf{A}$ is simply obtained by interchanging the second and third modes of each TTm core. That is, $\mathbf{A}^\mathsf{T} \in \mathbb{R}^{(I_1 \cdots I_d) \times (M_1 \cdots M_d)}$ can be represented in the TTm format with cores $\bar{\boldsymbol{\mathcal{A}}}^{(k)} \in \mathbb{R}^{P_{k-1} \times I_k \times M_k \times P_k}$.

Matrix-vector product When vectors and matrices are represented in the TT format, the matrix-vector product can be computed through tensor contractions just similar to the computation of the inner product. The product of a TT-matrix and a TT-vector leads to a TT-vector. The resulting TT ranks are also the product of the original ranks. It is thus generally followed by a rounding procedure to limit rank growth. The diagrammatic notation of $\mathbf{y} = \mathbf{A}\mathbf{x}$ with $d = 3$ are depicted in Figure 3-7. The full computation process of matrix-vector product is provided in Algorithm 7, with the computational complexity of $\mathcal{O}(dIM(PR)^2)$.

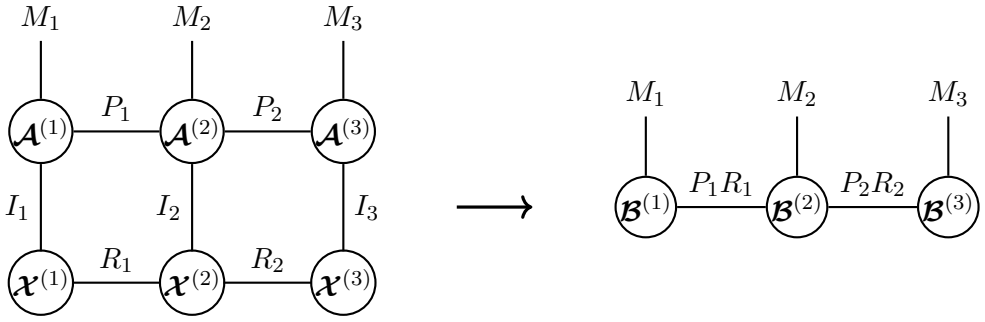


Figure 3-7: Diagrammatic notation of the matrix-vector product in the TT format.

Matrix-matrix product The product of a TT-matrix and a TT-matrix leads to a TT-matrix, which only requires small modifications from the matrix-vector product. The computational complexity of $\mathbf{C} = \mathbf{A}\mathbf{B}$ is $\mathcal{O}(dIJM(PR)^2)$.

3-3 Solving Linear Systems in TT Formats

The TT operations together with the rounding procedure allows us to first think about solving linear systems with the usage of traditional iterative methods [38, 39, 40]. However, these

methods are not very robust since computations of auxiliary quantities may require large TT ranks [1]. In order to get rid of additional vectors, another family of methods suggests to search for the cores of the TT format directly.

Suppose we choose to minimize the Frobenius norm error in terms of the full tensors,

$$\|\mathbf{x}^* - \tau(\boldsymbol{\mathcal{X}}^{(1)}, \dots, \boldsymbol{\mathcal{X}}^{(d)})\|^2 \to \min \tag{3-21}$$

over the TT cores $\boldsymbol{\mathcal{X}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$. This optimization problem becomes highly nonlinear and nonconvex w.r.t. the TT cores, due to the polylinear action of the τ operation (Definition 3-10). To relax the nonlinearity, the so-called alternating methods were proposed. A general approach is to substitute a simultaneous optimization over all TT cores by a subsequent optimization over each TT core, which is based on the fact that the TT format is linear w.r.t each of its cores.

Definition 3.14 (Frame matrices [33])**.** Given a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \dots \times I_d}$ in the TT format with cores $\boldsymbol{\mathcal{X}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$, the k-th frame matrix reads

$$\mathbf{X}_{\neq k} = (\mathbf{X}^{(k+1:d)})^{\mathsf{T}} \otimes \mathbf{I}_{I_k} \otimes \mathbf{X}^{(1:k-1)} \in \mathbb{R}^{(I_1 \cdots I_d) \times (R_{k-1} I_k R_k)}. \tag{3-22}$$

The frame matrix performs a linear map from the elements of the TT core to the elements of the initial tensor,

$$\mathbf{x} = \mathbf{X}_{\neq k} \mathbf{x}^{(k)}, \tag{3-23}$$

where $\mathbf{x} = \mathrm{vec}(\boldsymbol{\mathcal{X}})$ and $\mathbf{x}^{(k)} = \mathrm{vec}(\boldsymbol{\mathcal{X}}^{(k)}) \in \mathbb{R}^{R_{k-1} I_k R_k}$. Moreover, imposing the orthogonality conditions on the TT cores enables us to make the whole frame matrix orthogonal. That is, if $\mathbf{x}$ is put in the site-k mixed-canonical form, we have $\mathbf{X}_{\neq k}^{\mathsf{T}} \mathbf{X}_{\neq k} = \mathbf{I}_{R_{k-1} I_k R_k}$.

We now introduce several alternating methods for solving SPD linear systems $\mathbf{A}\mathbf{x} = \mathbf{b}$, where both the matrix $\mathbf{A} \in \mathbb{R}^{(I_1 \cdots I_d) \times (I_1 \cdots I_d)}$ and the right-hand side $\mathbf{b} \in \mathbb{R}^{I_1 \cdots I_d}$ are represented in the TT format, with TTm cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{P_{k-1} \times I_k \times I_k \times P_k}$ and TT cores $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{Q_{k-1} \times I_k \times Q_k}$, respectively. The solution $\mathbf{x} \in \mathbb{R}^{I_1 \cdots I_d}$ is also given in the TT format. We aim at optimizing the following objective function

$$J(\mathbf{x}) = \frac{1}{2} \langle \mathbf{x}, \mathbf{A}\mathbf{x} \rangle - \langle \mathbf{x}, \mathbf{b} \rangle. \tag{3-24}$$

3-3-1 Alternating Linear Scheme

The Alternating Linear Scheme (ALS) algorithm [35] solves the optimization problem (3-24) by a sequence of micro-steps, i.e., consecutive optimizations over TT cores $\boldsymbol{\mathcal{X}}^{(k)}$ with *fixed* TT ranks $\mathbf{r} = [R_1, \dots, R_{d-1}]$. Each such local problem writes

$$\boldsymbol{\mathcal{U}}^{(k)} = \arg\min_{\boldsymbol{\mathcal{X}}^{(k)}} J(\mathbf{x}) \quad \text{over} \quad \boldsymbol{\mathcal{X}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}. \tag{3-25}$$

The TT core $\boldsymbol{\mathcal{X}}^{(k)}$ is then replaced by $\boldsymbol{\mathcal{U}}^{(k)}$, and the next core is considered. In general, the cores are updated one by one from $k = 1$ to d (forward half-sweep) or $k = d$ to 1 (backward half-sweep), and so on until reaching convergence.

The linearity of the TT format (3-23) allows us to rewrite (3-25) as follows:

$$\mathbf{u}^{(k)} = \arg\min_{\mathbf{x}^{(k)}} J_{\mathbf{A}_k,\mathbf{b}_k}(\mathbf{x}^{(k)}) \quad \text{over} \quad \mathbf{x}^{(k)} \in \mathbb{R}^{R_{k-1}I_kR_k}, \tag{3-26}$$

where matrices $\mathbf{A}_k$ and $\mathbf{b}_k$ are given by

$$\begin{aligned} \mathbf{A}_k &= \mathbf{X}_{\neq k}^{\mathsf{T}} \mathbf{A} \mathbf{X}_{\neq k} \in \mathbb{R}^{(R_{k-1}I_kR_k)\times(R_{k-1}I_kR_k)}, \\ \mathbf{b}_k &= \mathbf{X}_{\neq k}^{\mathsf{T}} \mathbf{b} \in \mathbb{R}^{R_{k-1}I_kR_k}. \end{aligned} \tag{3-27}$$

The unique minimum is delivered by the solution of the local system

$$\mathbf{A}_k \mathbf{u}^{(k)} = \mathbf{b}_k, \tag{3-28}$$

which is of a reasonable size and within capabilities of standard methods. The computational complexity of solving each local system is basically $\mathcal{O}((IR^2)^3)$. As depicted in Figure 3-8, $\mathbf{A}_k$ and $\mathbf{b}_k$ can be assembled from the TT representations of $\mathbf{A}$, $\mathbf{x}$ and $\mathbf{b}$.

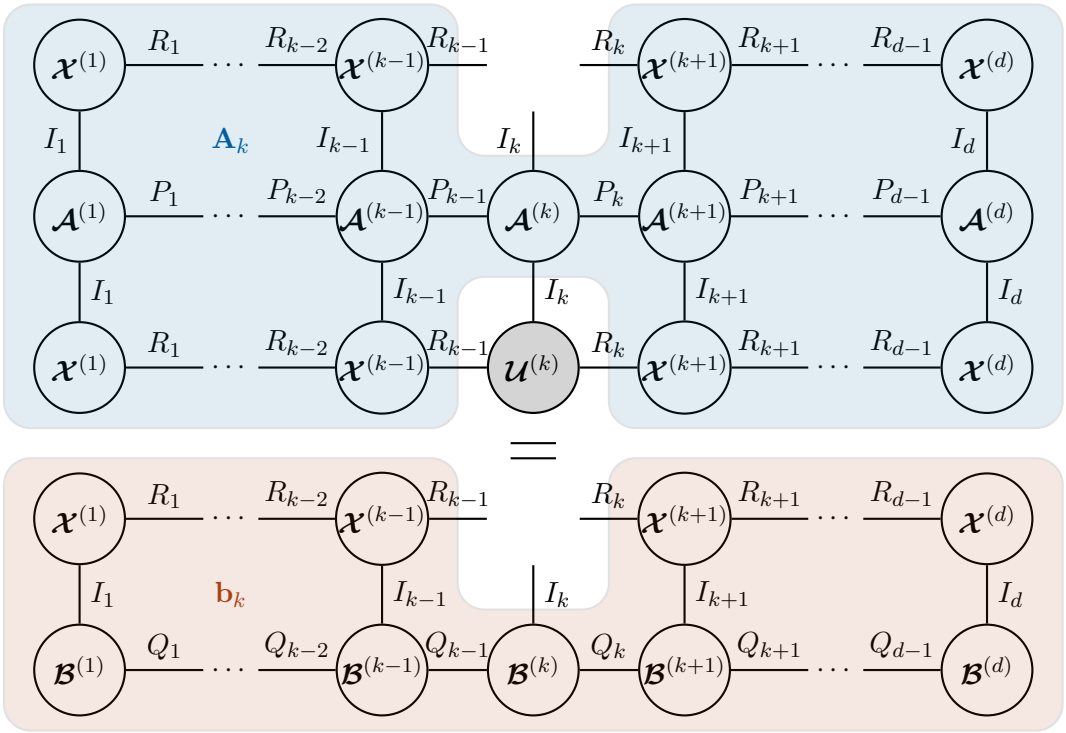


Figure 3-8: A local linear system (3-28) assembled directly from TT formats of $\mathbf{A}$, $\mathbf{x}$ and $\mathbf{b}$.

The accuracy of the obtained solution crucially depends on the conditioning of the local system. Fortunately, it can be put under control using the orthogonality constraints on the TT cores, and thus the orthogonality of the frame matrix [33]. If the TT-vector $\mathbf{x}$ is in the site-k mixed-canonical form when updating the k-th TT core, it can be shown that the

Algorithm 8: ALS for $\mathbf{Ax} = \mathbf{b}$ (forward half-sweep) [33]

Input: SPD TT-matrix $\mathbf{A} \in \mathbb{R}^{(I_1 \cdots I_d) \times (I_1 \cdots I_d)}$ with $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{P_{k-1} \times I_k \times I_k \times P_k}$,
right-hand side TT-vector $\mathbf{b} \in \mathbb{R}^{I_1 \cdots I_d}$ with $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{Q_{k-1} \times I_k \times Q_k}$,
initial guess TT-vector $\mathbf{t} \in \mathbb{R}^{I_1 \cdots I_d}$ with $\boldsymbol{\mathcal{T}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$

Output: optimized TT-vector $\mathbf{x} \in \mathbb{R}^{I_1 \cdots I_d}$ with $\boldsymbol{\mathcal{X}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$ such that $J(\mathbf{x}) \leq J(\mathbf{t})$

Copy $\boldsymbol{\mathcal{X}}^{(k)} \leftarrow \boldsymbol{\mathcal{T}}^{(k)},\ k = 1, \ldots, d$
Initialize $\boldsymbol{\mathcal{A}}^{<1} = \boldsymbol{\mathcal{A}}^{>d} = \boldsymbol{\mathcal{B}}^{<1} = \boldsymbol{\mathcal{B}}^{>d} = 1$
{Orthogonalization and right reductions}
for $k = d, d-1 \ldots, 2$ **do**
  QR decomposition $\mathbf{X}_{\mathsf{R}}^{(k)\mathsf{T}} = \mathbf{QR}$, $R_{k-1} \leftarrow \text{size}(\mathbf{Q}, 2)$
  Update core $\boldsymbol{\mathcal{X}}^{(k)} \leftarrow \text{reshape}(\mathbf{Q}^{\mathsf{T}}, [R_{k-1}, I_k, R_k])$
  Replace $\mathbf{X}_{\mathsf{L}}^{(k-1)} \leftarrow \mathbf{X}_{\mathsf{L}}^{(k-1)} \mathbf{R}^{\mathsf{T}}$
  Compute and store $\boldsymbol{\mathcal{A}}^{>k-1}, \boldsymbol{\mathcal{B}}^{>k-1}$ from $\boldsymbol{\mathcal{A}}^{>k}, \boldsymbol{\mathcal{B}}^{>k}$
end
{Optimization over TT cores}
for $k = 1, 2 \ldots, d$ **do**
  Form $\mathbf{A}_k$, $\mathbf{b}_k$ by (3-29) and solve $\mathbf{A}_k \mathbf{u}^{(k)} = \mathbf{b}_k$
  Replace $\boldsymbol{\mathcal{X}}^{(k)} \leftarrow \text{reshape}(\mathbf{u}^{(k)}, [R_{k-1}, I_k, R_k])$
  if $k \neq d$ **then**
    QR decomposition $\mathbf{X}_{\mathsf{L}}^{(k)} = \mathbf{QR}$, $R_k \leftarrow \text{size}(\mathbf{Q}, 2)$
    Update core $\boldsymbol{\mathcal{X}}^{(k)} \leftarrow \text{reshape}(\mathbf{Q}, [R_{k-1}, I_k, R_k])$
    Replace $\mathbf{X}_{\mathsf{R}}^{(k+1)} \leftarrow \mathbf{R} \mathbf{X}_{\mathsf{R}}^{(k+1)}$
    Compute left reductions $\boldsymbol{\mathcal{A}}^{<k+1}, \mathbf{b}^{<k+1}$ from $\boldsymbol{\mathcal{A}}^{<k}, \boldsymbol{\mathcal{B}}^{<k}$
  end
end

spectrum of $\mathbf{A}_k$ lies with the spectral range of $\mathbf{A}$ [33, 35]:

$$\begin{aligned}
\lambda_{\min}(\mathbf{A}_k) &= \lambda_{\min}(\mathbf{X}_{\neq k}^{\mathsf{T}} \mathbf{A} \mathbf{X}_{\neq k}) \\
&= \min_{\|\mathbf{v}\|=1} \langle \mathbf{X}_{\neq k} \mathbf{v}, \mathbf{A} \mathbf{X}_{\neq k} \mathbf{v} \rangle \\
&= \min_{\substack{\|\mathbf{z}\|=1 \\ \mathbf{z} \in \operatorname{span} \mathbf{X}_{\neq k}}} \langle \mathbf{z}, \mathbf{A}\mathbf{z} \rangle \\
&\geq \min_{\|\mathbf{z}\|=1} \langle \mathbf{z}, \mathbf{A}\mathbf{z} \rangle = \lambda_{\min}(\mathbf{A}),
\end{aligned}$$

and similarly $\lambda_{\max}(\mathbf{A}_k) \leq \lambda_{\max}(\mathbf{A})$. It follows that the condition numbers yield $\kappa(\mathbf{A}_k) \leq \kappa(\mathbf{A})$, i.e., the local system (3-28) is conditioned not worse than the original system $\mathbf{Ax} = \mathbf{b}$.

Computation of local systems

In each optimization step, we create and solve the local system (3-28). In fact, only small corrections are required to update the local systems between micro-steps $k = 1, \ldots, d$. In

order to show that, we define, in Figure 3-9, the left reductions,

$$\boldsymbol{\mathcal{A}}^{<k} \in \mathbb{R}^{1\times R_{k-1}\times R_{k-1}\times P_{k-1}}, \quad \boldsymbol{\mathcal{B}}^{<k} \in \mathbb{R}^{1\times R_{k-1}\times Q_{k-1}},$$

and right reductions,

$$\boldsymbol{\mathcal{A}}^{>k} \in \mathbb{R}^{P_k\times R_k\times R_k\times 1}, \ \boldsymbol{\mathcal{B}}^{>k} \in \mathbb{R}^{Q_k\times R_k\times 1}.$$

They can be seen as special TT(m) cores. Given $\boldsymbol{\mathcal{A}}^{<k}$ and the k-th cores $\boldsymbol{\mathcal{A}}^{(k)}$ and $\boldsymbol{\mathcal{X}}^{(k)}$, $\boldsymbol{\mathcal{A}}^{<k+1}$ can be easily computed via tensor contractions with the computational complexity of $\mathcal{O}(IR^3P + I^2R^2P^2)$. The computations of $\boldsymbol{\mathcal{B}}^{<k}$ and $\boldsymbol{\mathcal{B}}^{>k}$ are indeed the TT inner product, which cost $\mathcal{O}(IR^3)$ operations for each micro-step.

For initialization, we take $\boldsymbol{\mathcal{A}}^{<1} = \boldsymbol{\mathcal{A}}^{>d} = 1$. Now the local system (3-28) can be assembled as

$$\mathbf{A}_k = \tau(\boldsymbol{\mathcal{A}}^{<k}, \boldsymbol{\mathcal{A}}^{(k)}, \boldsymbol{\mathcal{A}}^{>k}), \quad \mathbf{b}_k = \tau(\boldsymbol{\mathcal{B}}^{<k}, \boldsymbol{\mathcal{B}}^{(k)}, \boldsymbol{\mathcal{B}}^{>k}). \tag{3-29}$$

In practice, the right reductions are computed and stored during the orthogonalization, while the left reductions are computed after each of the TT cores is updated. The detailed procedure of the ALS is shown in Algorithm 8.

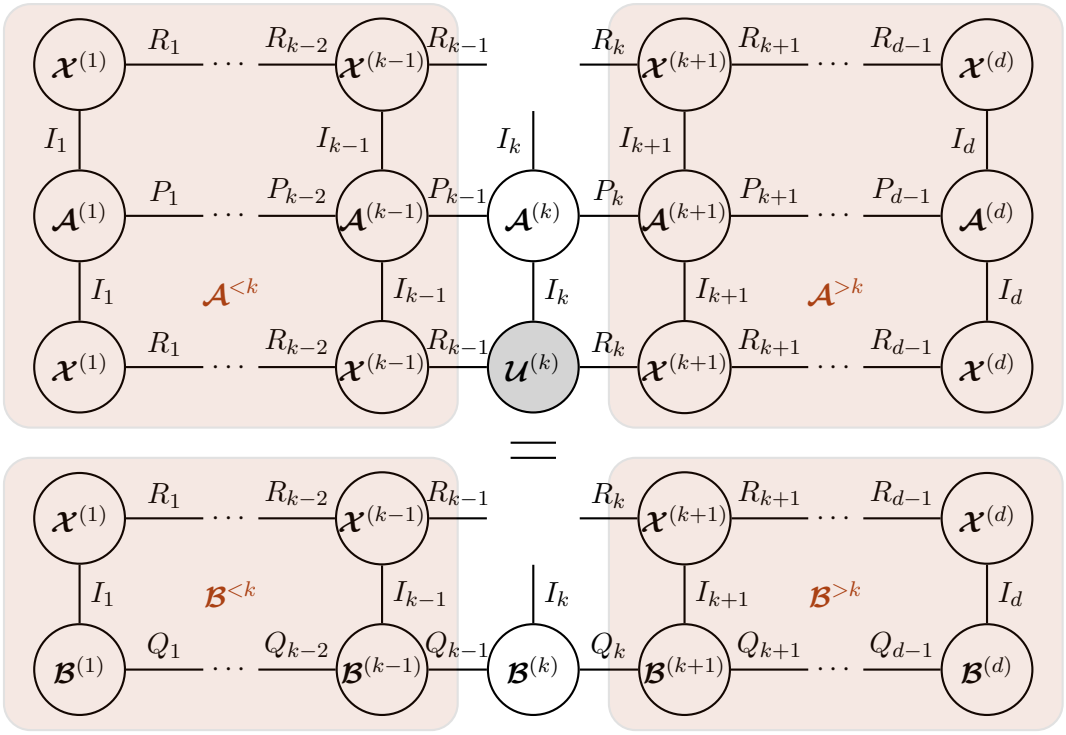


Figure 3-9: The graphical explanation of left reductions, $\boldsymbol{\mathcal{A}}^{<k}$ and $\boldsymbol{\mathcal{B}}^{<k}$, and right reductions, $\boldsymbol{\mathcal{A}}^{>k}$ and $\boldsymbol{\mathcal{B}}^{>k}$, used for assembling the linear system (3-28) [1].

TT approximation problems

As a special case, the ALS may be targeted to solve the problem of approximating a given tensor in the TT format from an initial guess, which is the case if the matrix $\mathbf{A} = \mathbf{I}$ and the

right-hand side is a normal tensor instead of a TT. With interface matrices being orthogonal, the identity is preserved in the local problem (3-27):

$$\mathbf{A}_k = \mathbf{X}_{\neq k}^{\mathsf{T}} \mathbf{A} \mathbf{X}_{\neq k} = \mathbf{X}_{\neq k}^{\mathsf{T}} \mathbf{I} \mathbf{X}_{\neq k} = \mathbf{I}.$$

Therefore, we just compute $\mathbf{b}_k$ according to (3-27) and assign $\mathbf{u}_k = \mathbf{b}_k$. When the expected TT ranks of the target tensor is the same as the TT ranks of the initial guess, the ALS only needs one half sweep for returning a good approximation. Furthermore, if the initial guess is close to the target, the approximated error can be further reduced.

3-3-2 Rank adaptation and Modified ALS

The drawback of the ALS algorithm is that the TT ranks remain the same during the computations. We thus have to guess the TT ranks of the solution a priori, which might be difficult. If they are underestimated, the obtained solution will be far from the exact one; if they are overestimated, the complexity of solving local systems may be too high.

The modified ALS (MALS) algorithm [35] allows us to change the TT ranks adaptively during the computations. Instead of a single TT core, the local optimization step is performed over a supercore $\boldsymbol{\mathcal{X}}^{(k:k+1)} = \tau(\boldsymbol{\mathcal{X}}^{(k)}, \boldsymbol{\mathcal{X}}^{(k+1)})$ as follows,

$$\mathbf{u}^{(k,k+1)} = \arg \min_{\mathbf{x}^{(k,k+1)}} J_{\mathbf{A}_{k,k+1}, \mathbf{b}_{k,k+1}}(\mathbf{x}^{(k,k+1)}) \quad \text{over} \quad \mathbf{x}^{(k,k+1)} \in \mathbb{R}^{R_{k-1} I_k I_{k+1} R_{k+1}}. \tag{3-30}$$

The minimizer is the solution of the local system $\mathbf{A}_{k,k+1}\mathbf{u}^{(k,k+1)} = \mathbf{b}_{k,k+1}$ with

$$\begin{aligned} \mathbf{A}_{k,k+1} &= \mathbf{X}_{\neq k,k+1}^{\mathsf{T}} \mathbf{A} \mathbf{X}_{\neq k,k+1} \in \mathbb{R}^{(R_{k-1} I_k I_{k+1} R_{k+1}) \times (R_{k-1} I_k I_{k+1} R_{k+1})}, \\ \mathbf{b}_{k,k+1} &= \mathbf{X}_{\neq k,k+1}^{\mathsf{T}} \mathbf{b} \in \mathbb{R}^{R_{k-1} I_k I_{k+1} R_{k+1}}, \end{aligned} \tag{3-31}$$

where $\mathbf{X}_{\neq k,k+1}$ is the *two-site* frame matrix

$$\mathbf{X}_{\neq k,k+1} = (\mathbf{X}^{(k+2:d)})^{\mathsf{T}} \otimes \mathbf{I}_{I_{k+1}} \otimes \mathbf{I}_{I_k} \otimes \mathbf{X}^{(1:k-1)} \in \mathbb{R}^{(I_1 \cdots I_d) \times (R_{k-1} I_k R_k)}. \tag{3-32}$$

The computational complexity of solving each local system becomes $\mathcal{O}((I^2R^2)^3)$. Similar to the ALS, the local system can be assembled as

$$\mathbf{A}_k = \tau(\boldsymbol{\mathcal{A}}^{<k}, \boldsymbol{\mathcal{A}}^{(k)}, \boldsymbol{\mathcal{A}}^{(k+1)}, \mathbf{A}^{>k+1}), \quad \mathbf{b}_k = \tau(\boldsymbol{\mathcal{B}}^{<k}, \boldsymbol{\mathcal{B}}^{(k)}, \boldsymbol{\mathcal{B}}^{(k+1)}, \mathbf{b}^{>k+1}). \tag{3-33}$$

The canonical form has to be ensured before solving each local system. The updated super core $\boldsymbol{\mathcal{U}}^{(k,k+1)}$ is then decomposed into $\boldsymbol{\mathcal{U}}^{(k)}$ and $\boldsymbol{\mathcal{U}}^{(k+1)}$ by applying the truncated SVD to its second unfolding matrix,

$$\begin{aligned} &\mathbf{U}_{[2]}^{(k,k+1)} \approx \mathbf{U}_{\mathsf{L}}^{(k)} \mathbf{U}_{\mathsf{R}}^{(k+1)} \in \mathbb{R}^{(R_{k-1} I_k) \times (I_{k+1} R_{k+1})}, \\ &\mathbf{U}_{\mathsf{L}}^{(k)} \in \mathbb{R}^{(R_{k-1} I_k) \times R'_k}, \quad \mathbf{U}_{\mathsf{R}}^{(k+1)} \in \mathbb{R}^{R'_k \times (I_{k+1} R_{k+1})}. \end{aligned} \tag{3-34}$$

This can be implemented either by setting an upper bound for the TT ranks or by giving a relative accuracy ε. The Frobenius norm of the local permutation to $\boldsymbol{\mathcal{U}}^{(k,k+1)}$ is the same as the global permutation due to the orthogonality of the frame matrix. After the decomposition,

Algorithm 9: MALS for $\mathbf{Ax} = \mathbf{b}$ (forward half-sweep) [33]

Input: SPD $\mathbf{A} \in \mathbb{R}^{(I_1 \cdots I_d) \times (I_1 \cdots I_d)}$ with $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{P_{k-1} \times I_k \times I_k \times P_k}$,
right-hand side $\mathbf{b} \in \mathbb{R}^{I_1 \cdots I_d}$ with $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{Q_{k-1} \times I_k \times Q_k}$,
initial guess $\mathbf{t} \in \mathbb{R}^{I_1 \cdots I_d}$ with $\boldsymbol{\mathcal{T}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$,
accuracy ε, rank bound $R_{\max}$

Output: rank-updated $\mathbf{x} \in \mathbb{R}^{I_1 \cdots I_d}$ with $\boldsymbol{\mathcal{X}}^{(k)} \in \mathbb{R}^{R'_{k-1} \times I_k \times R'_k}$ such that $R'_k \leq R_{\max}$

Copy $\boldsymbol{\mathcal{X}}^{(k)} \leftarrow \boldsymbol{\mathcal{T}}^{(k)},\ k = 1, \ldots, d$
Initialize $\boldsymbol{\mathcal{A}}^{<1} = \boldsymbol{\mathcal{A}}^{>d} = \boldsymbol{\mathcal{B}}^{<1} = \boldsymbol{\mathcal{B}}^{>d} = 1$
{Orthogonalization and right reductions}
for $k = d, d-1 \ldots, 2$ **do**
  QR decomposition $\mathbf{X}_{\mathsf{R}}^{(k)\mathsf{T}} = \mathbf{QR}$, $R_{k-1} \leftarrow \text{size}(\mathbf{Q}, 2)$
  Update core $\boldsymbol{\mathcal{X}}^{(k)} \leftarrow \text{reshape}(\mathbf{Q}^{\mathsf{T}}, [R_{k-1}, I_k, R_k])$
  Replace $\mathbf{X}_{\mathsf{L}}^{(k-1)} \leftarrow \mathbf{X}_{\mathsf{L}}^{(k-1)} \mathbf{R}^{\mathsf{T}}$
  Compute and store $\boldsymbol{\mathcal{A}}^{>k-1}, \boldsymbol{\mathcal{B}}^{>k-1}$ from $\boldsymbol{\mathcal{A}}^{>k}, \boldsymbol{\mathcal{B}}^{>k}$
end
{Optimization over TT cores}
for $k = 1, 2 \ldots, d-1$ **do**
  Form $\mathbf{A}_{k,k+1}$, $\mathbf{b}_{k,k+1}$ by (3-33) and solve $\mathbf{A}_{k,k+1}\mathbf{u}^{(k,k+1)} = \mathbf{b}_{k,k+1}$
  Compute ε-truncated SVD: $\mathbf{U}_{[2]}^{k,k+1} \approx \mathbf{U\Sigma V}^{\mathsf{T}}$ and $\text{rank}(\mathbf{\Sigma}) \leq R_{\max}$
  Replace $\mathbf{X}_{\mathsf{L}}^{(k)} \leftarrow \mathbf{U}$, $\mathbf{X}_{\mathsf{R}}^{(k+1)} \leftarrow \mathbf{\Sigma V}^{\mathsf{T}}$
  Compute left reductions $\boldsymbol{\mathcal{A}}^{<k+1}, \mathbf{b}^{<k+1}$ from $\boldsymbol{\mathcal{A}}^{<k}, \boldsymbol{\mathcal{B}}^{<k}$
end

the TT cores $\boldsymbol{\mathcal{X}}^{(k)}$ and $\boldsymbol{\mathcal{X}}^{(k+1)}$ are replaced by $\boldsymbol{\mathcal{U}}^{(k)}$ and $\boldsymbol{\mathcal{U}}^{(k+1)}$, respectively. And the TT rank R_k is substituted by R'_k. The procedure of the MALS is summarized in Algorithm 9.

The TT structure changes in each micro-step, and further optimization is carried out over the updated *tensor manifold*. This generally speeds up the convergence [36, 35], but also makes it more difficult to analyse the process. Actually, there is no systematic analysis yet of the theoretic convergence behavior of the MALS, though it works quick well in practice. Just like the ALS, the MALS algorithm can also be used to find TT approximations of given tensors.

3-3-3 Alternating Minimal Energy algorithm

Due to the local manner of optimization, the ALS (even MALS) may lose important information about the direction towards the exact solution, and stagnate at some local minima [1, 36, 35]. By combining the alternating optimization scheme with the steepest descent method, the Alternating Minimal Energy (AMEn) algorithm [33] is able to overcome this problem.

The basic idea comes from the conception of *enrichment*. It is found in [36, 41] that by adding to the solution a TT of small ranks with randomly filled TT cores from time to time, the convergence becomes faster and the accuracy level is also maintained. As an example, the

ALS (Algorithm 8) equipped with the enrichment after Line 12 can be written as follows,

$$\mathbf{X}_{\mathsf{L}}^{(k)} = \begin{bmatrix} \mathbf{U}_{\mathsf{L}}^{(k)} & \mathbf{Z}_{\mathsf{L}}^{(k)} \end{bmatrix}, \quad \mathbf{T}_{\mathsf{R}}^{(k+1)} = \begin{bmatrix} \mathbf{T}_{\mathsf{R}}^{(k+1)} \\ \mathbf{0} \end{bmatrix}, \tag{3-35}$$

where $\mathbf{Z}_{\mathsf{L}}^{(k)} \in \mathbb{R}^{R_{k-1} I_k \times \rho_k}$ are the left-unfolding of randomly populated tensors. This step may be seen as a zero TT-sum with TT ranks R_k becoming $R_k + \rho_k$, so that both $\mathbf{x} = \tau(\boldsymbol{\mathcal{X}}^{(1)}, \ldots, \boldsymbol{\mathcal{X}}^{(d)})$ and $J(\mathbf{x})$ are preserved. But the frame matrix is changed with new basis components for the optimization and hence may accelerate the convergence [1]. In order to avoid the stagnation in local minima, it is reasonable to choose the expansion related to the residual.

Given an initial guess $\mathbf{t}$, the residual is computed by $\mathbf{z} = \mathbf{b} - \mathbf{A}\mathbf{t}$. However, the TT cores of the residual would not match the gradient precisely after several updates of the TT cores of the solution. Note that the updated solution enters only the k-th core. Hence if we compute the *exact* residual after each micro-step, we are able to better approximates the current gradient. This process can be done efficiently since the TT-sum and matrix-vector product are performed core by core.

To be more clear, let $\mathbf{y} = -\mathbf{A}\mathbf{x}$, where we assume the first $k-1$ cores of $\mathbf{x}$ has been updated (and expanded) and the k-th core $\boldsymbol{\mathcal{X}}^{(k)} = \boldsymbol{\mathcal{U}}^{(k)}$ is the solution of the local system. Instead of concatenating the cores $\boldsymbol{\mathcal{Y}}^{(k)} \in \mathbb{R}^{R'_{k-1} P_{k-1} \times I_k \times R_k P_k}$ and $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{Q_{k-1} \times I_k \times Q_k}$ by (3-18), we take into account the cores $\boldsymbol{\mathcal{X}}^{(i)}$ for all $i < k$. Given the left reductions $\boldsymbol{\mathcal{A}}^{<k}$ and $\boldsymbol{\mathcal{B}}^{<k}$, the reduced residual is computed as follows,

$$\begin{aligned} \hat{\mathbf{Y}}_{\mathsf{R}}^{(k)} &= \mathbf{A}_{[2]}^{<k} \mathbf{Y}_{\mathsf{R}}^{(k)} \in \mathbb{R}^{R'_{k-1} \times I_k (R_k P_k)}, \\ \hat{\mathbf{B}}_{\mathsf{R}}^{(k)} &= \mathbf{B}_{\mathsf{L}}^{<k} \mathbf{B}_{\mathsf{R}}^{(k)} \in \mathbb{R}^{R'_{k-1} \times I_k Q_k}, \\ \hat{\mathbf{Z}}_{\mathsf{L}}^{(k)} &= \begin{bmatrix} \hat{\mathbf{B}}_{\mathsf{L}}^{(k)} & \hat{\mathbf{Y}}_{\mathsf{L}}^{(k)} \end{bmatrix} \in \mathbb{R}^{R'_{k-1} I_k \times (Q_k + R_k P_k)}, \end{aligned} \tag{3-36}$$

where $\mathbf{A}_{[2]}^{<k} \in \mathbb{R}^{R'_{k-1} \times (R'_{k-1} P_{k-1})}$ is the second unfolding matrix of $\boldsymbol{\mathcal{A}}^{<k}$. The computation is also described in Figure 3-10. The reduced residual $\hat{\mathbf{Z}}_{\mathsf{L}}^{(k)}$ is then used to expand the core by (3-35). To limit the rank growth, both $\mathbf{U}_{\mathsf{L}}^{(k)}$ and $\hat{\mathbf{Z}}_{\mathsf{L}}^{(k)}$ can be easily truncated by only one additional SVD. We summarize the whole procedure of the AMEn in Algorithm 10. The detailed convergence analysis can be accessed in [33, 1].

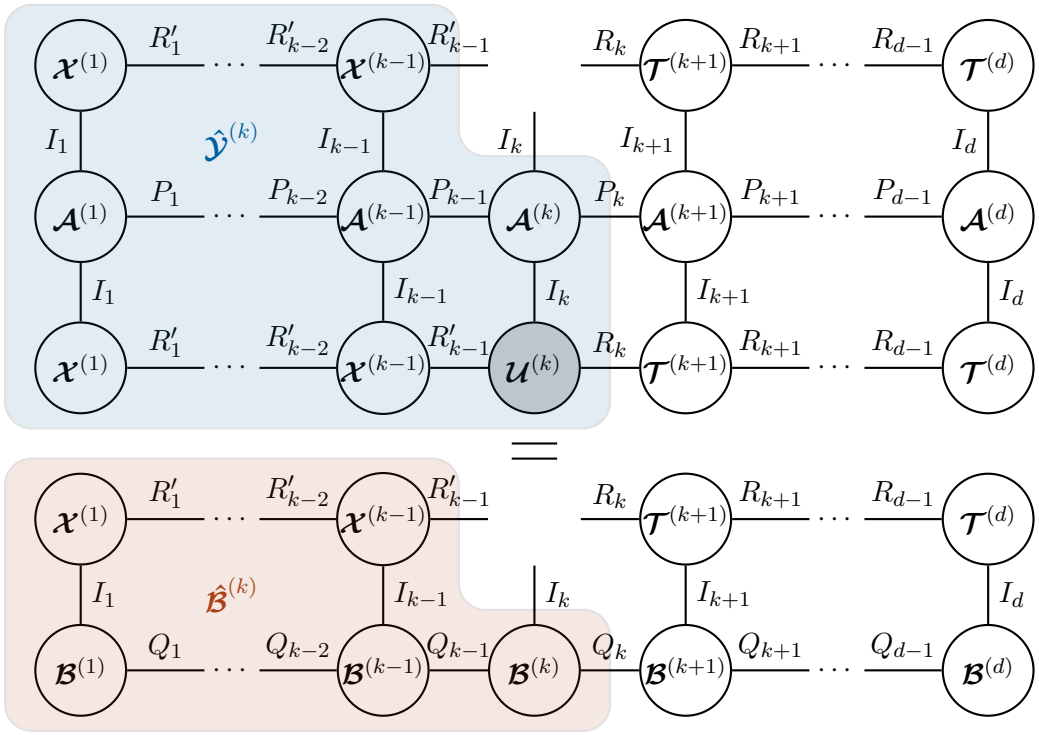


Figure 3-10: The computation of the reduced residual (3-36) based on the left reductions and the k-th TT cores.

Algorithm 10: AMEn for $\mathbf{Ax} = \mathbf{b}$ (one iteration) [1]

Input: SPD TT-matrix $\mathbf{A} \in \mathbb{R}^{(I_1 \cdots I_d) \times (I_1 \cdots I_d)}$ with $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{P_{k-1} \times I_k \times I_k \times P_k}$,
right-hand side TT-vector $\mathbf{b} \in \mathbb{R}^{I_1 \cdots I_d}$ with $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{Q_{k-1} \times I_k \times Q_k}$,
initial guess TT-vector $\mathbf{t} \in \mathbb{R}^{I_1 \cdots I_d}$ with $\boldsymbol{\mathcal{T}}^{(k)} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$,
accuracy ε and rank bound $R_{\max}$ for solution,
accuracy ϵ and rank bound $\rho_{\max}$ for residual

Output: optimized TT-vector $\mathbf{x} \in \mathbb{R}^{I_1 \cdots I_d}$ with $\boldsymbol{\mathcal{X}}^{(k)} \in \mathbb{R}^{R'_{k-1} \times I_k \times R'_k}$ such that $R'_k \leq R_{\max} + \rho_{\max}$

Copy $\boldsymbol{\mathcal{X}}^{(k)} \leftarrow \boldsymbol{\mathcal{T}}^{(k)},\ k = 1, \ldots, d$
Initialize $\boldsymbol{\mathcal{A}}^{<1} = \boldsymbol{\mathcal{A}}^{>d} = \boldsymbol{\mathcal{B}}^{<1} = \boldsymbol{\mathcal{B}}^{>d} = 1$
Compute the initial residual $\mathbf{z} = \mathbf{b} - \mathbf{Ax}$
{Orthogonalization and right reductions}
for $k = d, d-1 \ldots, 2$ **do**
  QR decomposition $\mathbf{X}_{\mathsf{R}}^{(k)\mathsf{T}} = \mathbf{QR}$, $R_{k-1} \leftarrow \text{size}(\mathbf{Q}, 2)$
  Update core $\boldsymbol{\mathcal{X}}^{(k)} \leftarrow \text{reshape}(\mathbf{Q}^{\mathsf{T}}, [R_{k-1}, I_k, R_k])$
  Replace $\mathbf{X}_{\mathsf{L}}^{(k-1)} \leftarrow \mathbf{X}_{\mathsf{L}}^{(k-1)} \mathbf{R}^{\mathsf{T}}$
  Compute and store $\boldsymbol{\mathcal{A}}^{>k-1}, \boldsymbol{\mathcal{B}}^{>k-1}$ from $\boldsymbol{\mathcal{A}}^{>k}, \boldsymbol{\mathcal{B}}^{>k}$
  QR decomposition $\mathbf{Z}_{\mathsf{R}}^{(k)\mathsf{T}} = \mathbf{QR}_{k-1}$, store $\mathbf{R}_{k-1}$
  Replace $\mathbf{Z}_{\mathsf{L}}^{(k-1)} \leftarrow \mathbf{Z}_{\mathsf{L}}^{(k-1)} \mathbf{R}_{k-1}^{\mathsf{T}}$
end
{Optimization over TT cores}
for $k = 1, 2 \ldots, d$ **do**
  Form $\mathbf{A}_k$, $\mathbf{b}_k$ by (3-29) and solve $\mathbf{A}_k \mathbf{u}^{(k)} = \mathbf{b}_k$
  Compute ε-truncated SVD: $\mathbf{U}_{\mathsf{L}}^{(k)} \approx \mathbf{U}_1 \boldsymbol{\Sigma}_1 \mathbf{V}_1^{\mathsf{T}}$ and $\text{rank}(\boldsymbol{\Sigma}_1) \leq R_{\max}$
  if $k \neq d$ **then**
    Compute reduced residual $\hat{\mathbf{Z}}_{\mathsf{L}}^{(k)}$ by (3-36)
    COmpute ϵ-truncated SVD: $\hat{\mathbf{Z}}_{\mathsf{L}}^{(k)} \mathbf{R}_k^{\mathsf{T}} \approx \mathbf{U}_2 \boldsymbol{\Sigma}_2 \mathbf{V}_2^{\mathsf{T}}$ and $\text{rank}(\boldsymbol{\Sigma}_2) \leq \rho_{\max}$
    Expand $\mathbf{X}_{\mathsf{L}}^{(k)} \leftarrow \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix}$, $\mathbf{X}_{\mathsf{R}}^{(k+1)} \leftarrow \begin{bmatrix} \mathbf{X}_{\mathsf{R}}^{(k+1)} \\ \mathbf{0} \end{bmatrix}$
    QR decomposition $\mathbf{X}_{\mathsf{L}}^{(k)} = \mathbf{QR}$, $R'_k \leftarrow \text{size}(\mathbf{Q}, 2)$
    Update core $\boldsymbol{\mathcal{X}}^{(k)} \leftarrow \text{reshape}(\mathbf{Q}, [R'_{k-1}, I_k, R'_k])$
    Replace $\mathbf{X}_{\mathsf{R}}^{(k+1)} \leftarrow \mathbf{R} \mathbf{X}_{\mathsf{R}}^{(k+1)}$
    Compute left reductions $\boldsymbol{\mathcal{A}}^{<k+1}, \mathbf{b}^{<k+1}$ from $\boldsymbol{\mathcal{A}}^{<k}, \boldsymbol{\mathcal{B}}^{<k}$
  end
end

3-4 Matrix Inversion

Matrix inversion forms the basis for our proposed method in Chapter 4. With all the TT arithmetics introduced before, the Newton method for the matrix inversion can be used to compute approximate inverses [42, 7]. Given a matrix $\mathbf{A}$, it has the following iterative form

$$\mathbf{X}_{k+1} = 2\mathbf{X}_k - \mathbf{X}_k\mathbf{A}\mathbf{X}_k, \quad k = 0, 1, \ldots.$$

If $\|\mathbf{A}\mathbf{X}_0 - \mathbf{I}\| < 1$ for some matrix norm, it converges to the inverse quadratically. The problem is that the newton method requires two matrix-matrix products in each step, and thus the TT ranks of $\mathbf{X}_k\mathbf{A}\mathbf{X}_k$ will quickly become prohibitively large. We now introduce an alternative approach that is much simpler to implement given an algorithm that solves linear systems (in the TT format).

The inverse matrix $\mathbf{X}$ to a given matrix $\mathbf{A}$ satisfies

$$\mathbf{A}\mathbf{X} = \mathbf{I}, \tag{3-37}$$

which can be written as a linear system:

$$(\mathbf{I} \otimes \mathbf{A})\,\mathrm{vec}(\mathbf{X}) = \mathrm{vec}(\mathbf{I}).$$

From the definition of the TTm decomposition, an identity matrix can be represented by a rank-1 TT-matrix, since it can be written as the Kronecker product of several small identity matrices. To benefit from this rank-1 structure, we choose the *vectorized* identity TT-matrix as the right-hand side and we are going to find a vectorized TT-matrix as the solution. Indeed, by merging or splitting the two dimensions in the middle, a 4-way TTm core can be easily converted into a 3-way TT core and vise versa. Since the vectorization is applied core by core, the Kronecker product $(\mathbf{I} \otimes \mathbf{A})$ should also be computed core by core. This can be simply done by an outer product of two vectorized TTm cores, as shown in Algorithm 11.

Now suppose that $\mathbf{X}$ is an approximate solution of $\mathbf{A}\mathbf{X} \approx \mathbf{I}$, i.e., $\|\mathbf{A}\mathbf{X} - \mathbf{I}\|$ is small. But it does not mean that $\|\mathbf{X}\mathbf{A} - \mathbf{I}\|$ is small, even for a symmetric matrix $\mathbf{A}$. That is, the obtained inverse for a symmetric matrix can be nonsymmetric. To avoid this and make the approach more robust [36], let us solve

$$\mathbf{A}\mathbf{X} + \mathbf{X}\mathbf{A} = 2\mathbf{I}, \tag{3-38}$$

instead of (3-37). This is the Lyapunov equation with the symmetric right-hand side, being known to have a symmetric solution. In addition, the TT ranks for the corresponding matrix are only two times larger than that for the initial one.

Algorithm 11: TT Kronecker product for inversion [36]

Input: TT-matrix $\mathbf{A}$ with cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{R}^{P_{k-1} \times I_k \times I_k \times P_k}$,
TT-matrix $\mathbf{B}$ with cores $\boldsymbol{\mathcal{B}}^{(k)} \in \mathbb{R}^{Q_{k-1} \times I_k \times I_k \times Q_k}$
Output: TT-matrix $\mathbf{C}$ with cores $\boldsymbol{\mathcal{C}}^{(k)} \in \mathbb{R}^{Q_{k-1}P_{k-1} \times I_k^2 \times I_k^2 \times Q_kP_k}$ such that $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$

```
for k = 1, 2 ..., d do
    C_k ← vec(B^(k)) vec(A^(k))^T
    C_k ← reshape(C_k, [Q_{k-1}, I_k, I_k, Q_k, P_{k-1}, I_k, I_k, P_k])
    C_k ← permute(C_k, [1, 5, 2, 6, 3, 7, 4, 8])
    C^(k) ← reshape(C_k, [Q_{k-1}P_{k-1}, I_k^2, I_k^2, Q_kP_k])
end
```

Chapter 4

Tensor Network Batch Bayesian Learning of LS-SVMs

In this Chapter, we develop the method of Tensor Network Batch Bayesian Learning of LS-SVM (TNBBL-LSSVM). The theoretical formulation of the model is given in Section 4-1, where we introduce the Bayesian learning framework of LS-SVMs and how to implement it in the TT format. Thereafter, we apply the TNBBL-LSSVM to a biomedical dataset to perform image classification. The results and analysis of the experiments are provided in Section 4-2. We will discuss how to determine the rank parameters of the model and compare the results given by the MALS and AMEn when computing matrix inversion. Finally, we compare the TNBBL-LSSVM with the state-of-the-art methods for solving large-scale LS-SVM problems.

Note that to represent vectors and matrices in the TT format, we assume that they are all first converted into a cubical tensor of dimensions 2, that is, the mode size of all TT-vectors and TT-matrices is 2.

4-1 Bayesian Learning in TT Formats

There are two main drawbacks associated with the original LS-SVMs. First, since the size of the linear KKT system scales with the number of data points, it becomes burdensome to solve the dual problem when dealing with large-scale datasets. In order to circumvent the bottleneck of the computational requirements and speed up the training process, we use the low-rank TT decomposition to approximate the kernel matrix and perform the computations in the TT format.

Another drawback is that the LS-SVM classifiers only predict the class label of a given instance. There is nothing we can tell about how confident they are to make such predictions. To tackle this problem, we develop a Bayesian learning method for solving the LS-SVM problems. As a result, we are able to obtain a probability distribution of the parameters, which enables us to construct confidence levels of the predictions, which is of use when we need to make further decisions.

Recall from Chapter 2 that given a training set $\{\mathbf{x}_k, y_k\}_{k=1}^N$ the dual problem of an LS-SVM classifier is given by

$$\begin{bmatrix} 0 & \mathbf{y}^\mathsf{T} \\ \mathbf{y} & \boldsymbol{\Omega} + \mathbf{I}_N/\gamma \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{1}_N \end{bmatrix}, \tag{4-1}$$

where the element of $\boldsymbol{\Omega}$ is defined as $\Omega_{k,l} = y_k y_l K(\mathbf{x}_k, \mathbf{x}_l)$ for $k, l = 1, \ldots, N$. It can be written in a more compact way as

$$\mathbf{H}\bar{\boldsymbol{\alpha}} = \mathbf{z}, \quad \text{with} \quad \bar{\boldsymbol{\alpha}} = \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} \in \mathbb{R}^{N+1}, \ \mathbf{z} = \begin{bmatrix} 0 \\ \mathbf{1}_N \end{bmatrix} \in \mathbb{R}^{N+1}. \tag{4-2}$$

In Bayesian way, we treat each row of matrix $\mathbf{H}$ as a "measurement" $\mathbf{h}_i$ (for $i = 1, \ldots, N+1$) [43] that is related to the following linear model,

$$z_i = \mathbf{h}_i \bar{\boldsymbol{\alpha}} + e_i, \tag{4-3}$$

where we assume the measurement noise e_i are i.i.d. random variables with zero-mean Gaussian distribution $e_i \sim \mathrm{N}(0, v^2)$, and the prior distribution of the parameters $\bar{\boldsymbol{\alpha}}$ is Gaussian with known mean and covariance $\bar{\boldsymbol{\alpha}} \sim \mathrm{N}(\boldsymbol{\mu}_0, \mathbf{P}_0)$. Then the linear model (4-3) can be expressed in probabilistic notation as[1]

$$\begin{aligned} P(z_i \,|\, \bar{\boldsymbol{\alpha}}) &= \mathrm{N}(z_i \,|\, \mathbf{h}_i \bar{\boldsymbol{\alpha}}, v^2), \\ P(\bar{\boldsymbol{\alpha}}) &= \mathrm{N}(\bar{\boldsymbol{\alpha}} \,|\, \boldsymbol{\mu}_0, \mathbf{P}_0). \end{aligned} \tag{4-4}$$

By application of Bayes' theorem, the posterior probability is derived as a consequence of the prior probability and the likelihood function,

$$\begin{aligned} P(\bar{\boldsymbol{\alpha}} \,|\, z_{1:N+1}) &\propto P(\bar{\boldsymbol{\alpha}}) \prod_{i=1}^{N+1} P(z_i \,|\, \bar{\boldsymbol{\alpha}}) \\ &= \mathrm{N}(\bar{\boldsymbol{\alpha}} \,|\, \boldsymbol{\mu}_0, \mathbf{P}_0) \prod_{i=1}^{N+1} \mathrm{N}(z_i \,|\, \mathbf{h}_i \bar{\boldsymbol{\alpha}}, v^2). \end{aligned} \tag{4-5}$$

The advantage of the prior distribution being Gaussian is that the posterior distribution will also be Gaussian:

$$P(\bar{\boldsymbol{\alpha}} \,|\, z_{1:N+1}) = \mathrm{N}(\bar{\boldsymbol{\alpha}} \,|\, \boldsymbol{\mu}_{N+1}, \mathbf{P}_{N+1}). \tag{4-6}$$

The analytic *batch solution* [43] of mean and covariance in (4-6) is written as

$$\begin{aligned} \mathbf{P} &:= \mathbf{P}_{N+1} = \left[\mathbf{P}_0^{-1} + \frac{1}{v^2} \mathbf{H}^\mathsf{T} \mathbf{H} \right]^{-1}, \\ \boldsymbol{\mu} &:= \boldsymbol{\mu}_{N+1} = \left[\mathbf{P}_0^{-1} + \frac{1}{v^2} \mathbf{H}^\mathsf{T} \mathbf{H} \right]^{-1} \left[\frac{1}{v^2} \mathbf{H}^\mathsf{T} \mathbf{z} + \mathbf{P}_0^{-1} \boldsymbol{\mu}_0 \right]. \end{aligned} \tag{4-7}$$

In brief, training an LS-SVM model by the Batch Bayesian Learning approach amounts to computing the inverse of an SPD matrix (and also a matrix-vector product). As a result, we can not only obtain the optimized model parameters, $[b^*; \boldsymbol{\alpha}^*] = \boldsymbol{\mu}$, but also compute a confidence interval based on the covariance matrix $\mathbf{P}$.

[1] The likelihood of z_i is also conditional on $\mathbf{h}_i$, but for simplicity this dependence is assumed to be explicit from the context.

By applying the TT-SVD to matrix $\mathbf{H}$ (reshaped to a $2d$-way tensor) with a prescribed rank bound $r(\mathbf{H})_{\max}$, we can convert it into a low-rank TT-matrix $\tilde{\mathbf{H}}$. Given the variance v^2 of the measurement noise and a predefined diagonal matrix $\mathbf{P}_0$, we are able to compute the matrix to be inverted,

$$\mathbf{A} = \mathbf{P}_0^{-1} + \frac{1}{v^2}\tilde{\mathbf{H}}^{\mathsf{T}}\tilde{\mathbf{H}}, \tag{4-8}$$

with a simple TT matrix-matrix product and a TT addition. To guarantee positive definiteness of $\tilde{\mathbf{H}}^{\mathsf{T}}\tilde{\mathbf{H}}$, no TT rounding step is followed. Therefore, the largest TT rank of $\mathbf{A}$ is basically $(r^2(\mathbf{H}) + 1)$.

Next we form the Lyapunov equation (3-38) by computing the core-based Kronecker product (i.e., Algorithm 11),

$$(\mathbf{I} \otimes \mathbf{A} + \mathbf{A} \otimes \mathbf{I})\,\mathrm{vec}(\mathbf{P}) = \mathrm{vec}(2\mathbf{I}), \tag{4-9}$$

which is the linear system to be solved by the MALS or AMEn, with the right-hand side being a rank-1 (vectorized) TT-matrix. The solution is indeed the posterior covariance $\mathbf{P}$. When solving this linear system, we also need to prescribe a rank bound $r(\mathbf{P})_{\max}$ for the solution. To make sure the obtained solution is symmetric, we use a rank-1 random symmetric TT-matrix[2] as the initial guess.

Suppose we have now obtained the approximate inverse matrix $\mathbf{P}$, the mean of parameters $\boldsymbol{\mu}$ can be computed by (4-7) with a TT matrix-vector product and a TT addition, where $\mathbf{z}$ is represented by a rank-2 TT-vector. In order to limit the rank growth, we may add one additional rounding step.

Making predictions is straightforward. Consider a new data point $\mathbf{x}'$, the class label can be computed by a TT Hadamard product and a TT inner product,

$$\begin{aligned}\hat{y}(\mathbf{x}') &= \mathrm{sign}\left(b^* + \sum_{k=1}^{N} \alpha_k^* y_k K(\mathbf{x}_k, \mathbf{x}')\right) \\ &= \mathrm{sign}\left(\begin{bmatrix}1 & \mathbf{k}^{\mathsf{T}}\end{bmatrix}\left(\begin{bmatrix}1\\ \mathbf{y}\end{bmatrix} \odot \boldsymbol{\mu}\right)\right),\end{aligned}$$

where the k-th entry of $\mathbf{k} \in \mathbb{R}^N$ is $K(\mathbf{x}_k, \mathbf{x}')$. To simplify the notation, we define

$$\bar{\mathbf{k}} = \begin{bmatrix}1\\ \mathbf{k}\end{bmatrix}, \quad \bar{\mathbf{y}} = \begin{bmatrix}1\\ \mathbf{y}\end{bmatrix}. \tag{4-10}$$

Then the label prediction can be written as

$$\hat{y}(\mathbf{x}') = \mathrm{sign}\left(\bar{\mathbf{k}}^{\mathsf{T}}(\bar{\mathbf{y}} \odot \boldsymbol{\mu})\right), \tag{4-11}$$

where $\bar{\mathbf{k}}$ can be converted into the TT format by the TT-SVD or ALS. The Hadamard product can be precomputed and may be followed by a rounding step. Since we assume a binary classification problem, i.e., $y_k \in \{+1, -1\}$, $\bar{\mathbf{y}}$ can be exactly represented by a rank-2 TT-vector if we rearrange the training set to put all the samples with positive labels in the first half such that

$$\bar{\mathbf{y}} = [+1 \ \cdots \ +1 \ -1 \ \cdots \ -1]^{\mathsf{T}}.$$

[2]Given a rank-1 TT-matrix $\mathbf{B}$, an easy way to get a symmetric TT-matrix is by computing $\mathbf{B}^{\mathsf{T}}\mathbf{B}$.

Experiments show that no matter how many +1s it has, the resulting TT-vector always has a maximal TT rank of 2.

The variance of the sample can be computed using the covariance $\mathbf{P}$ as

$$\sigma^2 = \bar{\mathbf{k}}^\mathsf{T}\mathbf{P}\bar{\mathbf{k}} + v^2. \tag{4-12}$$

Now we are able to establish the confidence interval according to the property of the Gaussian distribution. For example,

$$\begin{aligned} \hat{y}_\sigma^+(\mathbf{x}') &= \operatorname{sign}\left(\bar{\mathbf{k}}^\mathsf{T}(\bar{\mathbf{y}} \odot \boldsymbol{\mu}) + \sigma\right), \\ \hat{y}_\sigma^-(\mathbf{x}') &= \operatorname{sign}\left(\bar{\mathbf{k}}^\mathsf{T}(\bar{\mathbf{y}} \odot \boldsymbol{\mu}) - \sigma\right). \end{aligned} \tag{4-13}$$

If the values of $\hat{y}_\sigma^+(\mathbf{x}')$ and $\hat{y}_\sigma^-(\mathbf{x}')$ are the same as $\hat{y}(\mathbf{x}')$, we can say that we have a confidence level of 68.27% (1σ) about the prediction of $\mathbf{x}'$. For higher levels we may choose, e.g., 95.45% (2σ), 99.73% (3σ) and 99.99% (4σ).

Related work

In [2], a hierarchical framework for Bayesian inference of LS-SVMs is presented, which allows us to take probabilistic interpretations of the outputs and automatically determine the tuning parameters. This framework also enables us to do input selection using the principle of automatic relevance of determination. However, it has the same drawback as the basic LS-SVMs, i.e., its computational complexity grows fast as the number of data points increases. Hence, it is difficult to apply this framework on large-scale datasets.

In [44], a Tensor Network Kalman filter is developed to compute the posterior mean and covariance in a recursive manner and has been implemented on several large-sized datasets. In each iteration a single row of the kernel matrix is used to update the posterior distribution, and the posterior distribution in step i is interpreted as the prior distribution in step $i+1$. Thus, there is no need to actually build and store the entire kernel matrix, which reduces the storage requirements from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. This approach also avoids the computation of a large-scale matrix inverse, but in each iteration it involves 2 matrix-vector products, 1 inner product, 2 matrix additions and 4 rounding steps and it needs to run N iterations for a training set of N points.

4-2 Experiments and Analysis

In this Section, the TNBBL-LSSVM is applied to a retinal Optical coherence tomography (OCT) imaging dataset to make classifications for medical diagnosis. The model performance is compared with the state-of-the-art methods for large-scale LS-SVMs, including the Nyström method and FS-LSSVMs. The RBF kernel is used for all models, which means there are two hyper-parameters to be tuned: γ and σ^2_{rbf}.

All the experiments are conducted in MATLAB® R2019b. The laptop on which the computations are performed has an Intel® i5-8250U CPU running at 1.60 GHz with 8.00 GB of RAM. The code can be accessed from https://gitlab.tudelft.nl/msc-projects/tnbbl-lssvm.

4-2-1 OCTMNIST dataset

OCT is an imaging technique that uses low-coherence light to capture high-resolution images from within optical scattering media (e.g., biological tissue) [45]. It is now a standard of care for guiding the diagnosis and treatment of some of the leading causes of blindness [46].

We aim at testing the capacity of our model, thus we choose to work with a lightweight dataset called OCTMNIST[3] instead of the original high-resolution OCT dataset that contains more than 8 GB of data. OCTMNIST is a part of MedMNIST [47], a collection of 10 pre-processed medical open datasets. Each image is center-cropped and resized from $(384-1536)\times(277-512)$ into 28×28 (that is further stacked into a vector of length 784), with the value of each pixel ranging from 0 to 255. There are four types involved, but we only use two of them leading to a binary classification problem: CNV (simply defined as diseased and labeled as $+1$) and Normal (labeled as $+1$). Some samples of the dataset are already shown in Figure 1-1.

Since the mode size is set to 2, matrix $\mathbf{H}$ will be converted into a TT-matrix of size $2^d\times 2^d$. Hence, the number of data points in the training set should yield $N=2^d-1$. Table 4-1 gives an overview of the datasets to be used. The training set #1 and #2 are both subsets of training set #3, while training set #3, validation set and test set have no intersection with each other. The values are shifted and rescaled so that they end up ranging from 0 to 1. Most of the experiments start with training set #1, and further extend to #2 and #3 to explore whether the results obtained on smaller datasets can be well generalized to larger datasets.

Table 4-1: Information of the training and test sets to be used.

Dataset	Normal	Diseased	Total, N
Training set #1	2048	2047	4095
Training set #2	4096	4095	8191
Training set #3	8192	8191	16383
Validation set	2560	2559	5119
Test set	1000	1000	2000

Prior distribution

The Bayesian learning method allows us to include the information about the model parameters and dataset in the prior distribution and measurement noise. For OCTMNIST, there is no information available about its measurement noise, which is typically brought by sensors during the data collection. Thus we regard the variance of measurement noise as a hyper-parameter to be tuned. To begin with, it is set to $v^2=0.1^2$.

Since the values of $\boldsymbol{\alpha}$ can be either positive or negative, the prior mean is simply set to $\boldsymbol{\mu}_0=\mathbf{0}_{N+1}$, and the covariance is initialized as $\mathbf{P}_0=10\cdot\mathbf{I}_{N+1}$, yielding a wide distribution. It can be seen from (4-7) that $\mathbf{P}_0$ and v^2 together define how we obtain the posterior distribution. A small value of v^2 means that we trust the data more than the prior assumption. In general, $\mathbf{P}_0$ can be set to some coefficient a times an identity matrix, i.e., $\mathbf{P}_0=a\mathbf{I}$. This coefficient

[3] http://medmnist.com/

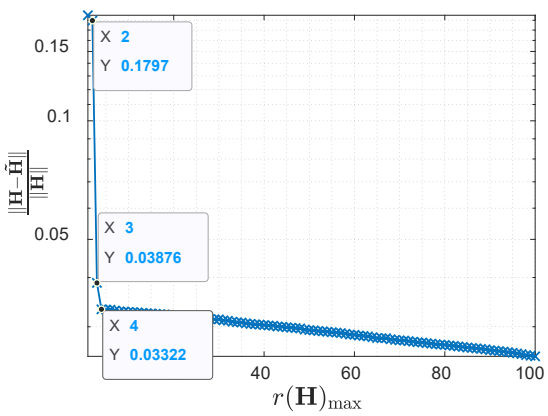


Figure 4-1: The relationship between the relative error of the TT-SVD and the rank bound $r(\mathbf{H})_{\max}$ up to 100 (on training set #1).

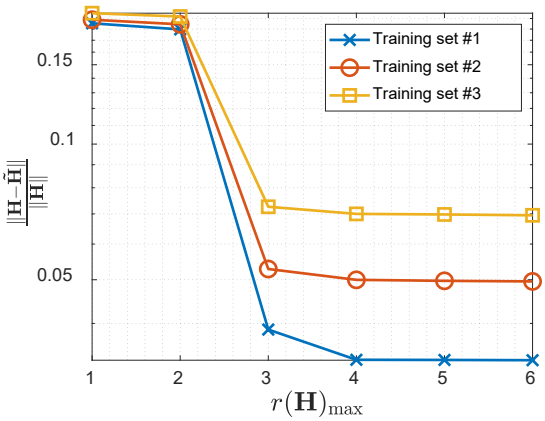


Figure 4-2: The relationship between the relative error of the TT-SVD and the rank bound $r(\mathbf{H})_{\max}$ up to 6 (on all training sets).

can also be seen as a hyper-parameter to be tuned. In this report, we only consider tuning v^2.

4-2-2 TT ranks for kernel matrix

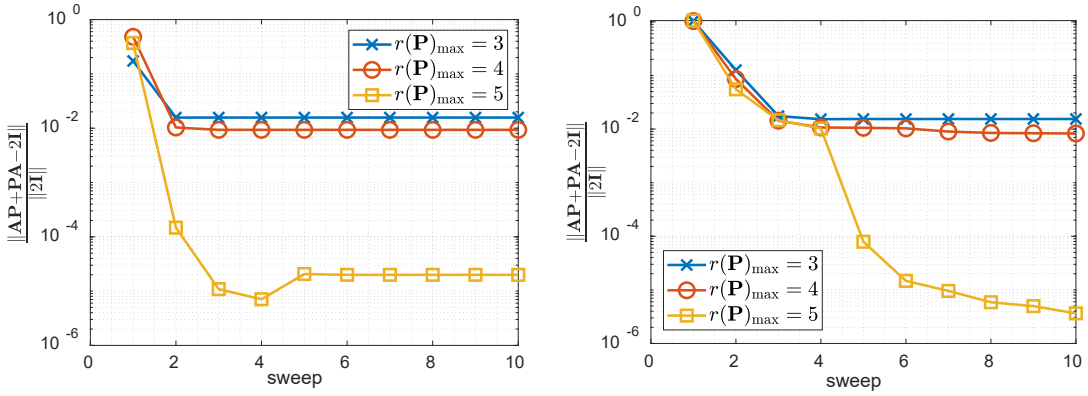
In order to find suitable values for the TT rank bound $r(\mathbf{H})_{\max}$, we apply the TT-SVD to matrix $\mathbf{H}$ generated from training set #1, and compute the approximate error $\|\mathbf{H}-\tilde{\mathbf{H}}\|/\|\mathbf{H}\|$ of the obtained TT-matrix $\tilde{\mathbf{H}}$. If there exists a jump (or several jumps) somewhere, then it would be reasonable to pay more attention to this value (or these values). Before that, we need to set the hyper-parameters first. A reasonable choice is to use the tuned values of the basic LS-SVM model. We do a 10-fold cross-validation on training set #1. The resulting values are $\gamma = 0.15$ and $\sigma^2_{\text{rbf}} = 1.15^2$.

The relationship between the relative error and the rank bound $r(\mathbf{H})_{\max}$ up to 100 is shown in Figure 4-1. It can be seen that a big jump appears between 2 and 3 with the error reducing from 0.180 to 0.039, and a small jump appears between 3 and 4 with the error reducing from 0.039 to 0.033. The experiments on larger datasets (zoomed in up to 6) report similar results, which are summarized in Figure 4-2. It is thus reasonable to focus on the cases where $r(\mathbf{H})_{\max} \in \{3, 4\}$.

The total number of elements of $\mathbf{H}$ stored in the TT format is about $4\log_2(N+1)r(\mathbf{H})_{\max}$, while in the normal form it is $(N+1)^2$. Take $r(\mathbf{H})_{\max} = 3$ on training set #3 as an example, it means we use only 0.00019% of the original space to retain more than 92.5% of the information.

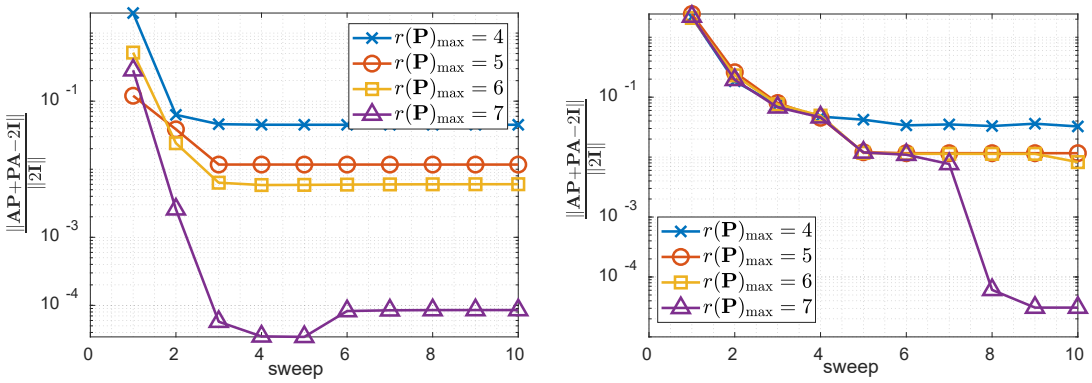
4-2-3 MALS vs. AMEn

We can now use the MALS and AMEn to compute the matrix inverse. The rank bound $r(\mathbf{P})_{\max}$ is determined as follows. For each value of $r(\mathbf{H})_{\max} \in \{3, 4\}$, we increase $r(\mathbf{P})_{\max}$



(a) MALS (b) AMEn

Figure 4-3: The relative residual versus the sweep number on training set #1 with $r(\mathbf{H})_{\max} = 3$. $r(\mathbf{P})_{\max} = 7$ for both the MALS and AMEn.



(a) MALS (b) AMEn

Figure 4-4: The relative residual versus the sweep number on training set #1 with $r(\mathbf{H})_{\max} = 4$. $r(\mathbf{P})_{\max} = 7$ for both the MALS and AMEn.

and run the algorithm until the converged residual yield

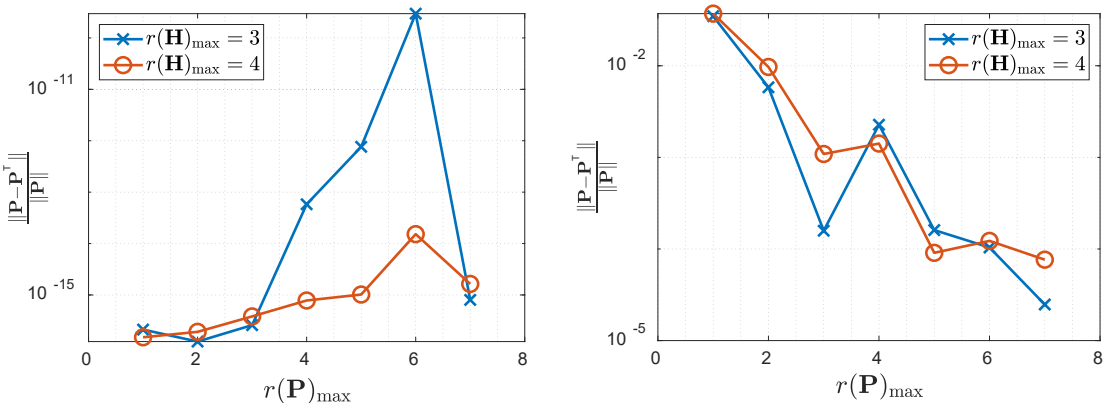
$$\frac{\|\mathbf{AP} + \mathbf{PA} - 2\mathbf{I}\|}{\|2\mathbf{I}\|} \leq 1 \times 10^{-4}, \tag{4-14}$$

where $\mathbf{A}$ is defined by (4-8). The rank bound for the residual in the AMEn algorithm is fixed to 1. Figure 4-3 and 4-4 show the convergence behavior of the MALS and AMEn running on training set #1 with $r(\mathbf{H})_{\max} = 3$ and $r(\mathbf{H})_{\max} = 4$, respectively. The rank parameters for all three training sets are summarized in Table 4-2.

The relative error in symmetry on training set #1 is shown in Figure 4-5, and the positive definiteness is summarized in Figure 4-6. It can be seen that the AMEn needs a value $r(\mathbf{P})_{\max} \geq 5$ to make the error close to 1×10^{-4}, while the error of the MALS can be always seen as 0. As for eigenvalues, the resulting inverse matrix is shown to be positive definite when $r(\mathbf{P})_{\max} \geq 4$. The experiments on training sets #2 and #3 indicate that when the rank

Table 4-2: The rank parameter of the inverse matrix, $r(\mathbf{P})_{\max}$, for all training sets.

Training set	$r(\mathbf{H})_{\max}=3$		$r(\mathbf{H})_{\max}=4$	
	MALS	AMEn	MALS	AMEn
#1	5	5	7	7
#2	5	5	7	7
#3	5	5	9	8



(a) MALS (b) AMEn

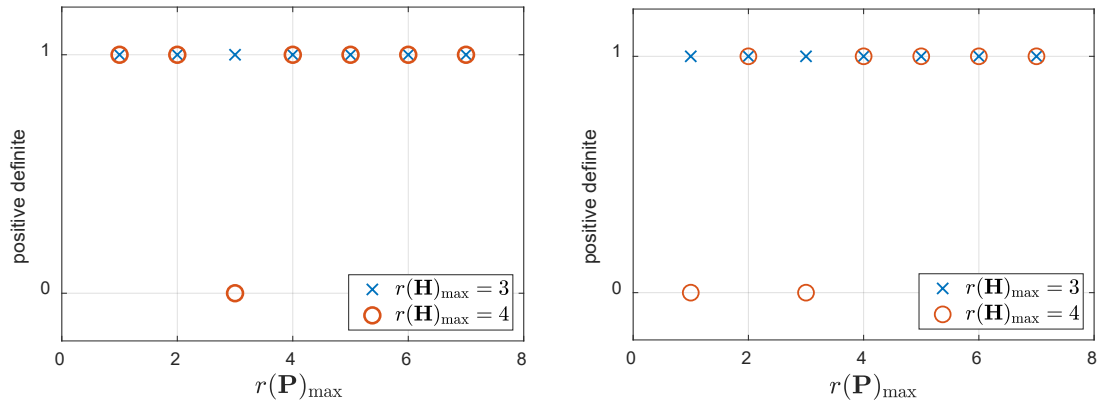
Figure 4-5: The relative error in symmetry of the inverse matrix.

bound $r(\mathbf{P})_{\max}$ is chosen according to Table 4-2, the obtained inverse matrix $\mathbf{P}$ is guaranteed to be positive definite with the error in symmetry less than 1×10^{-8} for the MALS and less than 5×10^{-4} for the AMEn.

The time needed for each sweep are shown in Figure 4-7. We can see that the time of one sweep for the MALS is longer than that for the AMEn due to its two-site optimization scheme (i.e., it optimizes a super-core in each micro-step). However, from Figure 4-3 and 4-4 it is clear that the MALS needs less sweeps to converge. In practice, an early stopping scheme is applied, i.e., the algorithm stops when the residuals of three consecutive sweeps are all below the tolerance or the error changes between them are both smaller than 1×10^{-4}. The total optimization time for the two algorithms are close, as will be shown in Section 4-2-6.

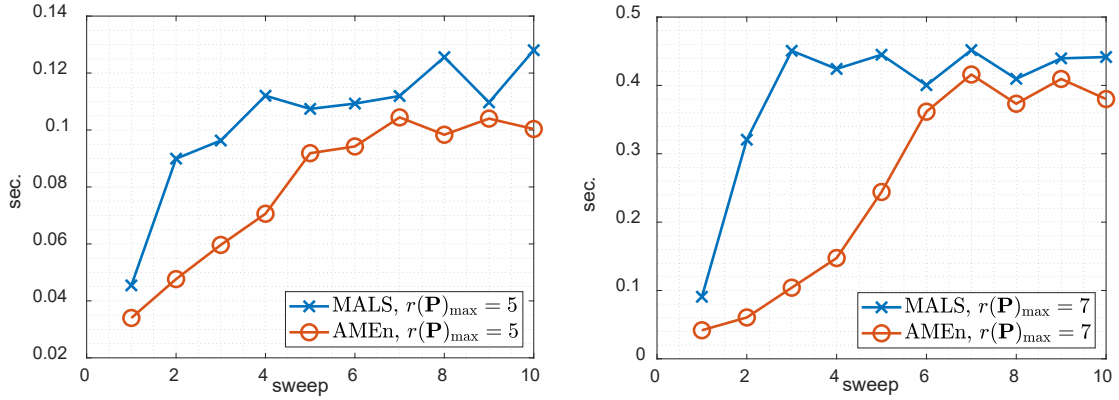
4-2-4 Tuning hyper-parameters

With the chosen rank bound parameters, $r(\mathbf{H})_{\max}$ and $r(\mathbf{P})_{\max}$, we are able to tune the hyper-parameters, v^2, γ and σ^2_{rbf}, of our TNBBL-LSSVM model using grid search and cross validation. Each time we use 4095 samples of validation set for training and the rest 1024 points for evaluating the performance (i.e., the accuracy), and we do this five times. Note that σ^2_{rbf} determines the values of $\bar{\mathbf{k}}$ when making predictions. Therefore, only after getting the tuned hyper-parameters, we can finally determine the truncation value for converting $\bar{\mathbf{k}}$ into the TT format.



(a) MALS (b) AMEn

Figure 4-6: The positive definiteness of the inverse matrix by checking if all eigenvalues are positive, (1 for Yes and 0 for No).



(a) $r(\mathbf{H})_{\max} = 3$ (b) $r(\mathbf{H})_{\max} = 4$

Figure 4-7: Time of one sweep versus the sweep number.

Since we do not need the confidence level now, we can convert the obtained TT-vector $\boldsymbol{\mu}$ back into a (normal) vector to compute the predicted labels by (4-11). The hyper-parameters for $r(\mathbf{H})_{\max} = 3$ and $r(\mathbf{H})_{\max} = 4$ are listed in Table 4-3. In Section 4-2-6, it will be shown that these values generalize well on training set #2 and #3. The variance of measurement noise is finally set to $v^2 = 0.05^2$.

Table 4-3: Model hyper-parameters γ and σ^2_{rbf} given by a 5-fold cross-validation.

$r(\mathbf{H})_{\max}$	γ	σ^2_{rbf}
3	0.15	0.73^2
4	0.25	0.68^2

4-2-5 Prediction in low-rank structures

In this part, we show that not only can the model training be carried out in low-rank structure, but the prediction can also be made with low-rank approximations. Given a test point, we need to compute $\bar{\mathbf{k}}$ based on the kernel function, convert it into a TT-vector, and compute the prediction by (4-11), where $\bar{\mathbf{y}}$ and $\boldsymbol{\mu}$ are both in low-rank TT forms. This allows us to choose a low-rank bound $r(\bar{\mathbf{k}})_{\max}$ when applying the TT-SVD.

Suppose that the resulting TT-vector of $\bar{\mathbf{k}}$ is denoted by $\bar{\mathbf{k}}_{\text{svd}}$, and the corresponding prediction computed in the TT format is denoted by $\hat{y}_{\text{svd}}$. Here we assume the sign function is not applied, i.e., $\hat{y} = \bar{\mathbf{k}}^{\mathsf{T}}(\bar{\mathbf{y}} \odot \boldsymbol{\mu})$. To measure how good the approximation is, we consider the relative error $\|\hat{y} - \hat{y}_{\text{svd}}\|/\|\hat{y}\|$ instead of $\|\bar{\mathbf{k}} - \bar{\mathbf{k}}_{\text{svd}}\|/\|\bar{\mathbf{k}}\|$. For a test set of more than one data point, the error becomes $\|\hat{\mathbf{y}} - \hat{\mathbf{y}}_{\text{svd}}\|/\|\hat{\mathbf{y}}\|$. Since the length of $\bar{\mathbf{k}} \in \mathbb{R}^N$ grows with the size of training set, the rank bound should also increases. The information about using the TT-SVD to convert $\bar{\mathbf{k}}$ into the TT format is summarized in Table 4-4 for $r(\mathbf{H})_{\max} = 3$, and in Table 4-5 for $r(\mathbf{H})_{\max} = 4$, including the rank bounds, relative error, computation time and storage reduction.

It can be seen that on training set #3 the low-rank approximation reduces the storage requirements by 73.0% with the relative error less than 0.1. Moreover, it is found in practice that computing the predictions with this low-rank TT structure may even increase the accuracy by about 1%.

Table 4-4: The rank bounds, relative error, computation time and storage reduction when using TT-SVD to convert $\bar{\mathbf{k}}$ into TT format, with $r(\mathbf{H})_{\max} = 3$.

Training set	$r(\bar{\mathbf{k}})_{\max}$	$\|\hat{\mathbf{y}} - \hat{\mathbf{y}}_{\text{svd}}\|/\|\hat{\mathbf{y}}\|$	Time (sec.)	Storage reduction (%)
#1	12	0.0931	11.36	58.4
#2	15	0.0986	17.12	64.6
#3	18	0.0956	23.63	73.0

Table 4-5: The rank bounds, relative error, computation time and storage reduction when using TT-SVD to convert $\bar{\mathbf{k}}$ into TT format, with $r(\mathbf{H})_{\max} = 4$.

Training set	$r(\bar{\mathbf{k}})_{\max}$	$\|\hat{\mathbf{y}} - \hat{\mathbf{y}}_{\text{svd}}\|/\|\hat{\mathbf{y}}\|$	Time (sec.)	Storage reduction (%)
#1	12	0.0830	11.90	58.4
#2	15	0.0962	17.75	64.6
#3	20	0.0941	27.32	68.5

Fast prediction: ALS on sorted datasets

Now consider the cases where we have access to the true labels of the data points to be predicted, and we want to do a quick test of our model performance before launching it. We may use the ALS to approximate $\bar{\mathbf{k}}$. When the initial guess is close to the target, only one half-sweep is enough to give a good approximation. To take advantage of this, we need to rearrange the dataset so that two adjacent points are as close as possible. Then we can take

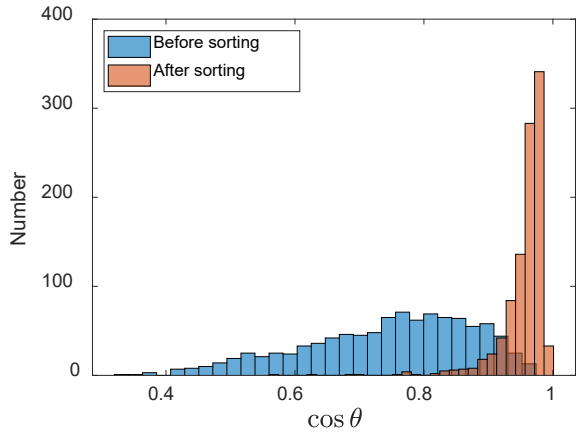


Figure 4-8: Cosine of angles between each two adjacent points (with positive labels) before and after sorting.

the TT form of the last point as the initial value when approximating the current one. We compute the (cosine of) angle between two data points to measure their closeness, i.e.,

$$\cos\theta = \frac{\mathbf{a}^{\mathsf{T}}\mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|}. \tag{4-15}$$

The closer the value is to 1, the closer they are with each other.

To begin with, the dataset is divided into two parts: one with positive labels and another with negative labels. Then we sort them separately based on the angle: find the most close one to the first point and put it in the second place; then find the most close one to the second point and put it in the third place, and so on again. The angles between each two adjacent points (with positive labels) before and after sorting are shown in Figure 4-8. About 94% of values are greater than 0.9.

Note that $\bar{\mathbf{k}}_1$ is converted using the TT-SVD with rank bound $r(\bar{\mathbf{k}})_{\max}$, which is a TT-vector in site-d mixed-canonical form. Then it is used as the initial guess for the ALS to perform one half-sweep from right to left, leading to a TT-vector in site-1 mixed-canonical form. In this way, we save the time spent on extra orthogonalization steps. The time and relative error given by the ALS is shown in Table 4-6 for $r(\mathbf{H})_{\max} = 3$, and in Table 4-7 for $r(\mathbf{H})_{\max} = 4$. The same values of $r(\bar{\mathbf{k}})_{\max}$ are used for direct comparison. We can see that not only the time required has been reduced to about 65% of the original but the relative error is also reduced by $0.01 - 0.02$.

Table 4-6: The rank bounds, relative error and computation time when using ALS to convert $\bar{\mathbf{k}}$ into TT format, with $r(\mathbf{H})_{\max} = 3$.

Training set	$r(\bar{\mathbf{k}})_{\max}$	$\|\hat{\mathbf{y}} - \hat{\mathbf{y}}_{\text{als}}\|/\|\hat{\mathbf{y}}\|$	Time (sec.)
#1	12	0.0987	7.07
#2	15	0.0856	11.57
#3	18	0.0717	15.29

Table 4-7: The rank bounds, relative error and computation time when using ALS to convert $\bar{\mathbf{k}}$ into TT format, with $r(\mathbf{H})_{\max} = 4$.

Training set	$r(\bar{\mathbf{k}})_{\max}$	$\|\hat{\mathbf{y}} - \hat{\mathbf{y}}_{\text{als}}\|/\|\hat{\mathbf{y}}\|$	Time (sec.)
#1	12	0.0960	8.30
#2	15	0.0896	12.13
#3	20	0.0701	18.76

4-2-6 Final performance

Finally, we compare the performance of our model with the Nyström method and FS-LSSVM. They are both trained on training set #3 using the LS-SVMlab[4]. The Nyström method uses a uniform random sampling procedure, and the FS-LSSVM chooses the support vectors based on the Rényi Entropy. The hyper-parameters of them are tuned separately by grid search.

To begin with, let us report the training time for the TNBBL-LSSVM to find which parts cost the most. The total training time (T) can be divided into four parts, i.e., computing the kernel matrix (t_1), applying the TT-SVD (t_2), matrix inversion by solving linear systems (t_3), and others (t_4). The results on training set # 3 are shown in Table 4-8, where the values are all taken from the average of five experiments. It can be seen that the time for computing the kernel matrix and converting it into the TT format accounts for more than 98.8% of the total time when $r(\mathbf{H})_{\max} = 3$.

Table 4-8: The total training time of the TNBBL-LSSVM on training set #3.

Rank Bound	t_1 (sec.)	t_2 (sec.)	t_3 (sec.)		t_4 (sec.)	T (sec.)	
			MALS	AMEn		MALS	AMEn
$r(\mathbf{H})_{\max} = 3$	56.54	52.72	1.073	0.921	0.181	110.51	110.36
$r(\mathbf{H})_{\max} = 4$	56.54	67.75	7.326	8.124	0.175	131.79	132.59

We compute the accuracy (i.e., ratio of correct predictions) on test set to measure the performance. The higher value between the MALS and AMEn is taken as the final result for the TNBBL-LSSVM. The accuracy versus the size of training set is shown in Figure 4-9. We can see that the hyper-parameters tuned on a smaller dataset also work well when the model is trained on larger datasets. The performance of all models are summarized in Table 4-9. We see that for the OCTMNIST dataset, the TNBBL-LSSVM performs even better than those two.

For the TNBBL-LSSVM, we also provide the number of correctly labeled points for different confidence levels and the corresponding accuracy, in Table 4-10. The higher the confidence level is, the higher the accuracy we get.

[4] https://www.esat.kuleuven.be/sista/lssvmlab/

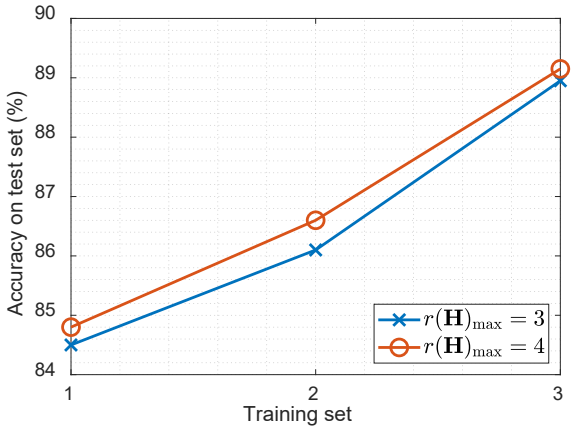


Figure 4-9: The accuracy of TNBBL-LSSVM on test set versus the size of training set.

Table 4-9: The accuracy on test set of the TNBBL-LSSVM, Nyström method and FS-LSSVM.

Model	**Accuracy** (%)
TNBBL-LSSVM, $r(\mathbf{H})_{\max} = 3$	**88.95**
TNBBL-LSSVM, $r(\mathbf{H})_{\max} = 4$	**89.15**
Nyström, $M = 50$	86.10
Nyström, $M = 100$	85.80
Nyström, $M = 200$	86.45
Nyström, $M = 300$	85.65
FS-LSSVM, $M = 100$	78.75
FS-LSSVM, $M = 200$	82.30
FS-LSSVM, $M = 300$	85.45
FS-LSSVM, $M = 400$	87.35

Table 4-10: The number of data points and the corresponding accuracy for different confidence levels.

Confidence Level	Number		Accuracy (%)	
	$r(\mathbf{H})_{\max} = 3$	$r(\mathbf{H})_{\max} = 4$	$r(\mathbf{H})_{\max} = 3$	$r(\mathbf{H})_{\max} = 4$
1σ	1352	1256	92.38	93.71
2σ	1097	1040	94.07	94.62
3σ	894	912	94.52	94.95
4σ	771	758	95.03	95.68

Chapter 5

Conclusion and Future Work

In this thesis, we give a comprehensive introduction to the LS-SVM and TT decomposition. A Tensor Network Batch Bayesian learning approach is developed to solve large-scale LS-SVM problems. The numerical experiments on a medical image dataset show that it performs competitively with the two state-of-the-art methods. We can now answer the questions posed before.

Q1. *By converting the kernel matrix in the TT format, can we find a low-rank structure to efficiently store the data without losing its accuracy?*
The relative error after applying the TT-SVD indicates that we are able to approximate the kernel matrix in a highly compressed low-rank TT form, without losing too much information.

Q2. *Can the matrix inversion be reliably computed in the TT format, i.e., is the resulting covariance matrix guaranteed to be symmetric positive definite (SPD)?*
The computation of the matrix inversion is quite accurate and reliable. By solving the Lyapunov equation, the resulting covariance matrix is guaranteed to be symmetric and positive definite if the convergence error is below some threshold.

Q3. *What are the advantages and disadvantages of our method compared with the state of the art, and what would be the suitable cases to implement our model?*
Both the Nyström method and FS-LSSVMs use a subset of dataset to obtain an approximation, so that they can reduce the storage requirements, which is one of the most important bottlenecks for large-scale LS-SVMs. For our method, however, we still need to compute and store the whole kernel matrix with the storage requirements of $\mathcal{O}(N^2)$, due to the use of TT-SVD. Therefore, our method currently cannot be well applied to the cases where the kernel matrix cannot be completely stored. When the complete kernel matrix can be stored and the TT-SVD can be applied, our method is shown to be able to outperform those two methods.

Q4. *In which aspects can the method be further improved?*
In order to avoid the storage requirements of $\mathcal{O}(N^2)$, the future research may focus

on finding approaches to directly constructing the kernel matrix in the TT format or converting it block by block. Due to time limit, we are not able to test the model on more datasets and problems, e.g., the regression problems. This method needs more cases to test.

Bibliography

[1] S. Dolgov, *Tenor Product Methods in Numerical Simulation of High-Dimensional Dynamical Problems*. PhD thesis, Universität Leipzig, Leipzig, 2014.

[2] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. World Scientific, 2002.

[3] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[4] C. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," *Advances in neural information processing systems*, vol. 13, pp. 682–688, 2000.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Johns Hopkins University Press, 1996.

[6] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[7] I. V. Oseledets, "Approximation of $2^d \times 2^d$ matrices using tensor decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 4, pp. 2130–2145, 2010.

[8] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[9] H. Li, *Statistical Learning Methods*. Tsinghua University Press, 2019.

[10] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[11] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[12] J. Mercer, "Functions of positive and negative type, and their connection the theory of integral equations," *Philosophical transactions of the royal society of London*, vol. 209, no. 441-458, pp. 415–446, 1909.

[13] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Tech. Rep. MSR-TR-98-14, Microsoft Research, April 1998.

[14] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[15] J. A. K. Suykens, L. Lukas, P. Van Dooren, B. De Moor, and J. Vandewalle, "Least squares support vector machine classifiers: a large scale algorithm," in *Proceedings of the European Conference on Circuit Theory and Design*, vol. 10, Citeseer, 1999.

[16] B. Hamers, *Kernel Models for Large Scale Applications*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 2004.

[17] D. M. Young, *Iterative Solution of Large Linear Systems*. Elsevier, 1971.

[18] A. Greenbaum, *Iterative Methods for Solving Linear Systems*. SIAM, 1997.

[19] S. S. Keerthi and S. K. Shevade, "SMO algorithm for least-squares SVM formulations," *Neural computation*, vol. 15, no. 2, pp. 487–507, 2003.

[20] T. Van Gestel, J. A. K. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle, "Benchmarking least squares support vector machine classifiers," *Machine learning*, vol. 54, no. 1, pp. 5–32, 2004.

[21] K. De Brabanter, J. De Brabanter, J. A. K. Suykens, and B. De Moor, "Optimized fixed-size kernel models for large data sets," *Computational Statistics and Data Analysis*, vol. 54, no. 6, pp. 1484–1504, 2010.

[22] U. Schollwöck, "The density-matrix renormalization group in the age of matrix product states," *Annals of physics*, vol. 326, no. 1, pp. 96–192, 2011.

[23] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[24] K. Batselier, A. Cichocki, and N. Wong, "MERACLE: constructive layer-wise conversion of a tensor train into a MERA," *Communications on Applied Mathematics and Computation*, 2020.

[25] A. Cichocki, N. Lee, I. V. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Foundations and Trends® in Machine Learning*, vol. 9, no. 4-5, pp. 249–429, 2016.

[26] C. Lubich, *From Quantum to Classical Molecular Dynamics: Reduced Models and Numerical Analysis*. European Mathematical Society, 2008.

[27] X. Wang and I. H. Sloan, "Why are high-dimensional finance problems often of low effective dimension?," *SIAM Journal on Scientific Computing*, vol. 27, no. 1, pp. 159–183, 2005.

[28] B. N. Khoromskij, "Tensors-structured numerical methods in scientific computing: Survey on recent advances," *Chemometrics and Intelligent Laboratory Systems*, vol. 110, no. 1, pp. 1–19, 2012.

[29] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[30] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis," *UCLA Working Papers in Phonetics*, pp. 1–84, 1970.

[31] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[32] J. Håstad, "Tensor rank is NP-complete," *Journal of algorithms*, vol. 11, no. 4, pp. 644–654, 1990.

[33] S. V. Dolgov and D. V. Savostyanov, "Alternating minimal energy methods for linear systems in higher dimensions," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. A2248–A2271, 2014.

[34] B. N. Khoromskij, "O($d\log N$)-quantics approximation of N-d tensors in high-dimensional numerical modeling," *Constructive Approximation*, vol. 34, no. 2, pp. 257–280, 2011.

[35] S. Holtz, T. Rohwedder, and R. Schneider, "The alternating linear scheme for tensor optimization in the tensor train format," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. A683–A713, 2012.

[36] I. V. Oseledets and S. V. Dolgov, "Solution of linear systems and matrix inversion in the tt-format," *SIAM Journal on Scientific Computing*, vol. 34, no. 5, pp. A2718–A2739, 2012.

[37] C.-Y. Ko, K. Batselier, L. Daniel, W. Yu, and N. Wong, "Fast and accurate tensor completion with total variation regularized tensor trains," *IEEE Transactions on Image Processing*, vol. 29, pp. 6918–6931, 2020.

[38] W. Hackbusch, B. N. Khoromskij, and E. E. Tyrtyshnikov, "Approximate iterations for structured matrices," *Numerical Mathematics*, vol. 109, no. 3, pp. 365–383, 2008.

[39] O. S. Lebedeva, "Tensor conjugate-gradient-type method for Rayleigh quotient minimization in block QTT-format," *Russian Journal of Numerical Analysis and Mathematical Modelling*, no. 5, pp. 465–489, 2011.

[40] S. V. Dolgov, "TT-GMRES: solution to a linear system in the structured tensor format," *Russian Journal of Numerical Analysis and Mathematical Modelling*, vol. 28, no. 2, pp. 149–172, 2013.

[41] I. V. Oseledets, "DMRG approach to fast linear algebra in the TT-format," *Computational Methods in Applied Mathematics*, vol. 11, no. 3, pp. 382–393, 2011.

[42] V. Olshevsky, I. V. Oseledets, and E. E. Tyrtyshnikov, "Superfast inversion of two-level Toeplitz matrices using Newton iteration and tensor-displacement structure," in *Recent Advances in Matrix and Operator Theory*, pp. 229–240, Springer, 2007.

[43] S. Särkkä, *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks, Cambridge University Press, 2013.

[44] M. J. Lucassen, "Recursive Tensor Network Bayesian Learning of Large-Scale LS-SVMs," Master's thesis, Delft University of Technology, The Netherlands, 2020.

[45] Wikipedia, "Optical coherence tomography — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Optical%20coherence%20tomography&oldid=1026649062, 2021. [Online; accessed 07-June-2021].

[46] D. S. Kermany, M. Goldbaum, W. Cai, C. C. Valentim, H. Liang, S. L. Baxter, A. McKeown, G. Yang, X. Wu, F. Yan, *et al.*, "Identifying medical diagnoses and treatable diseases by image-based deep learning," *Cell*, vol. 172, no. 5, pp. 1122–1131, 2018.

[47] J. Yang, R. Shi, and B. Ni, "Medmnist classification decathlon: A lightweight automl benchmark for medical image analysis," *arXiv preprint arXiv:2010.14925*, 2020.

Glossary

SVMs	Support Vector Machines
LS-SVMs	Least Squares Support Vector Machines
QP	Quadratic Programming
SMO	Sequential Minimal Optimization
KKT	Karush-Kuhn-Tucker
CG	Conjugate Gradient
FS-LSSVM	Fixed Size LS-SVM
CP	canonical polyadic
TT	Tensor Train
MPS	Matrix Product State
SVD	Singular Value Decomposition
TTm	Tensor Train matrix
MPO	Matrix Product Operator
SPD	symmetric positive definite
ALS	Alternating Linear Scheme
AMEn	Alternating Minimal Energy
TNBBL-LSSVM	Tensor Network Batch Bayesian Learning of LS-SVM
OCT	Optical coherence tomography