SURIM 2019 Final Report: The Tucker and Tensor Train Formats as Low-Parametric Representations of Tensors with Applications [*]

Jason Zhu

August 2019

Abstract

Tensors are multidimensional arrays that arise naturally in high-dimensional problems from modelling quantum mechanical systems, machine learning, solution of partial differential equations, signal processing, and other fields. Numerical solutions to these problems are difficult due to the curse of dimensionality, making the number of operations and amount of storage necessary to solve these problems increase exponentially with the number of dimensions. Analogous to low-rank matrix decompositions, low-parameter tensor decompositions have been introduced in order to make tensors easier to work with in applications. In this expository paper, we will study the Tucker decomposition as well as the Tensor Train decomposition as presented in sources [1] and [2]. We will investigate how each one of these decompositions present a low-parametric format of a tensor, as well as algorithms to develop the low-parametric approximations and reapproximations of a given tensor in these formats.

1 Introduction

Tensors are multidimensional generalizations of matrices and vectors and, in general, can be thought of as a multidimensional arrays containing data.

[*]Advised by Dr. Vladimir Kazeev

They find application in high-dimensional problems such as those in machine learning, modelling quantum mechanical systems, signal processing, solution of partial differential equations, financial modelling, and other areas of applied and computational mathematics. Throughout the paper, we will use calligraphic capital letters to refer to tensors, and we will denote a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$ to represent an order-d tensor of sizes n_k for $k \in \{1, 2, ..., d\}$. Note that in the case of $d = 1, 2$, we recover the basic case of column vectors and matrices.

To avoid the curse of dimensionality, we want to work with perhaps a low parametric format of a high-dimensional tensor, analogous to low-rank decompositions of matrices in linear algebra. As such, we will study two notable tensor decompositions, namely the Tucker format and the Tensor Train format, as low-parametric formats of a high-dimensional tensor. In this paper, I will cover the basic linear algebra necessary to develop fluency when working with tensors, the structure of each tensor decomposition, relevant approximation and reapproximation algorithms, and one application of the Tensor Train decomposition in supervised learning problems.

2 Review of Rank Revealing Factorizations and the SVD

If we wanted to store all entries of matrix A $\in \mathcal{R}^{m \times n}$ individually, it would take $O(mn)$ memory. However, we can do better by exploiting some of the structure of a matrix to perhaps reduce the amount of storage necessary to represent a matrix, namely its rank.

Definition 2.1. Let $A \in \mathbb{R}^{m \times n}$ be a matrix. The rank of A is the minimum number $r \in \mathbb{N}$ such that

$$A = \sum_{k=1}^{r} u_k v_k^\top$$

where $u_k \in \mathbb{R}^m$ and $v_k \in \mathbb{R}^n$ for all $k \in \{1, 2, ..., r\}$. We define the zero matrix to have rank 0.

Equivalently, we could say that a matrix of rank r admits a dyadic decomposition

$$A = UV^\top$$

where $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$. It then follows that if A is of rank r, then we can store $O(r(m+n))$ entries instead of the full $O(mn)$ matrix to represent the information in matrix A.

Furthermore, we will introduce an important matrix decomposition that will assist us in understanding low-rank factorizations and approximations, namely the compact singular value decomposition.

Theorem 2.1. *Let $A \in \mathbb{R}^{m\times n}$ be a matrix of rank r. Then there exist matrices U, S and V such that*

$$A = USV^\top$$

where $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ have orthonormal columns, and $S \in \mathbb{R}^{r\times r}$ is diagonal with the diagonal elements being called the singular values.

The singular value decomposition of a matrix exists for all matrices, and, as we will see below, provides a good starting point for finding low-rank approximations of a given matrix.

Below are three questions that will guide us in finding low-rank representation of a matrix, approximating a matrix by a lower-rank representation, and reapproximation of a matrix in its dyadic decomposition.

1. **Exact Factorization:** Let $A \in \mathbb{R}^{m\times n}$ be a matrix of rank r with all entries given. Find matrices $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ such that $A = UV^\top$.

2. **Approximation:** Let $A \in \mathbb{R}^{m\times n}$ be a matrix of rank r with all entries given. Find matrices $U \in \mathbb{R}^{m\times \hat{r}}$ and $V \in \mathbb{R}^{n\times \hat{r}}$ such that $A \approx \hat{A} = UV^\top$, with $||A - \hat{A}||_F$ minimized.

3. **Reapproximation:** Let $A \in \mathbb{R}^{m\times n}$ be a matrix of rank r, with $A \simeq \hat{A} = UV^\top$ as an approximate decomposition be given. Find matrices $\hat{U} \in \mathbb{R}^{m\times \hat{r}}$ and $\hat{V} \in \mathbb{R}^{n\times \hat{r}}$ with $\hat{r} \leq r$ such that $\hat{A} \approx \hat{U}\hat{V}^\top$.

To answer the question of exact factorization, we simply consider the compact SVD. The compact SVD automatically provides us with an exact rank r factorization of a given matrix, and by multiplying matrices, we can achieve the desired rank r representation easily.

$$\begin{aligned} A &= USV^\top \\ &= U\hat{V}^\top \end{aligned}$$

where $\hat{V}^\top = SV^\top$.

For the question of approximation of a matrix, If we wish to obtain a rank $\hat{r}$ approximation to matrix A, we can consider the truncated SVD of A. By this, we mean that we will truncate the tailing singular values of A to attain a low rank approximation of A. First we will partition the SVD as follows:

$$\begin{aligned} A &= USV^\top \\ &= \begin{pmatrix} \hat{U} & \tilde{U} \end{pmatrix} \begin{pmatrix} \hat{S} & 0 \\ 0 & \tilde{S} \end{pmatrix} \begin{pmatrix} \hat{V}^\top \\ \tilde{V}^\top \end{pmatrix} \\ &= \hat{U}\hat{S}\hat{V}^\top + \tilde{U}\tilde{S}\tilde{V}^\top \end{aligned}$$

In this process, we partition U, S, and V into matrices $\hat{U} \in \mathbb{R}^{m\times\hat{r}}$, $\tilde{U} \in \mathbb{R}^{m\times(r-\hat{r})}$, $\hat{S} \in \mathbb{R}^{\hat{r}\times\hat{r}}$, $\tilde{S} \in \mathbb{R}^{(r-\hat{r})\times(r-\hat{r})}$, $\hat{V} \in \mathbb{R}^{n\times\hat{r}}$, and $\tilde{V} \in \mathbb{R}^{n\times(r-\hat{r})}$. We can then consider the following rank $\hat{r}$ approximation $\hat{A}$ of A:

$$\hat{A} = \hat{U}\hat{S}\hat{V}^\top$$

By the Eckard-Young theorem, we have that $\hat{A}$ is the best rank $\hat{r}$ approximation to A. Since we have $\hat{A}$ in its SVD format, we can again merge matrices to achieve the dyadic format. In addition, we have a nice expression to describe the error of $\hat{A}$ in terms of the singular values of A.

We have that

$$\begin{aligned} ||A - \hat{A}||_F &= ||USV^\top - \hat{U}\hat{S}\hat{V}^\top||_F \\ &= ||\hat{U}\hat{S}\hat{V}^\top + \tilde{U}\tilde{S}\tilde{V}^\top - \hat{U}\hat{S}\hat{V}^\top||_F. \\ &= ||\tilde{U}\tilde{S}\tilde{V}^\top||_F \end{aligned}$$

By the unitary invariance of the Frobenius norm,

$$||\tilde{U}\tilde{S}\tilde{V}^\top||_F = ||\tilde{S}||_F = \sqrt{\sum_{k=\hat{r}+1}^{r} \sigma_k^2}$$

As a corollary, if we wanted to find a matrix B such that $||A - B||_F \leq \varepsilon$, for some $\varepsilon \geq 0$, we could truncate the last $r - k$ singular values appropriately to achieve an approximation of error ε away from A.

Remark 2.2. *For the problem of exact factorization, it may be helpful to set ε to be a value close to machine precision in the case of working with finite precision arithmetic to truncate the spurious nonzero values.*

Finally, to answer the question of reapproximation, suppose $A = UV^\top$, but with sub-optimal rank r for matrices U and V. We want to avoid forming the matrix A, again so we will work solely with matrices U and V. To achieve a lower rank decomposition, perform a QR decomposition on U and V so that

$$U = Q_u R_u$$

and

$$V = Q_v R_v.$$

We can then assemble a smaller matrix $P = R_u R_v$ such that

$$A = Q_u P Q_v^\top.$$

We can then perform an SVD on matrix P to get that

$$P = XDY^\top.$$

Letting $\hat{U} = Q_u X$ and $\hat{V} = Q_v Y$, we have that $A = \hat{U} D \hat{V}^\top$ is an SVD of A. We can then easily apply the problem of approximation in this format.

These matrix questions can be used to motivate the algorithms we will develop in sections 2 and 3 to find (approximate) decompositions of tensors in the Tucker and Tensor Train format. Additionally, it's worth noting that in this paper, we make the strong assumption that the tensors we work with have some kind of low rank structure. This is due to the contexts in which we work with tensors (signal processing, machine learning, quantum mechanical systems, PDEs) where there is a high degree of symmetry, smoothness, or structure in the underlying data, so we can expect some low-parametric format to fall out from the decompositions, as in the case of low-rank matrix factorizations.

3 The Tucker Decomposition

3.1 Introduction to the Tucker Decomposition

The Tucker Decomposition is a low parametric format of a tensor that factorizes a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$ into a core tensor $S \in r_1 \times r_2 \times \ldots \times r_d$

where $r_k \leq n_k$ and a set of matrices $U^{(k)} \in \mathbb{R}^{n_k \times r_k}$ for all $k \in \{1, 2, ..., d\}$. Elementwise, we have the following:

$$\mathcal{A}_{i_1 i_2 ... i_d} = \sum_{\alpha_1=1}^{r_1} \sum_{\alpha_2=1}^{r_2} ... \sum_{\alpha_d=1}^{r_d} S_{\alpha_1 \alpha_2 ... \alpha_d} U^{(1)}_{i_1 \alpha_1} U^{(2)}_{i_2 \alpha_2} ... U^{(d)}_{i_d \alpha_d} \tag{1}$$

Again, we will ask the same questions as we do in the matrix case, namely those of achieving this format with access to all entries of the original tensor and those of (re)approximations. This can be done via a method known as the higher order singular value decomposition, otherwise known as the HOSVD.

3.2 Mode-k Unfoldings and the Mode-k product

Before introducing the HOSVD, we need to first introduce the idea of the *matricizations* or *unfoldings* of a tensor of arbitrary size.

Definition 3.1. Let $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 ... \times n_d}$. The matrix unfolding $A_{(k)} \in \mathbb{R}^{n_k \times p_k}$ where

$$p_k = \prod_{\substack{s=1 \\ s \neq k}}^{d} n_s$$

contains the element $A_{i_1 i_2 ... i_d}$ at the position with row number i_k and column number equal to

$$\sum_{\substack{\alpha=1 \\ \alpha \neq k}}^{d} (i_\alpha - 1) \prod_{\substack{\beta=1 \\ \beta > \alpha \\ \beta \neq k}}^{d} n_\beta$$

Example 3.1. Let $\mathcal{A} \in \mathbb{R}^{2 \times 3 \times 2}$ where $\mathcal{A}_{i_1 i_2 ... i_d} = 100 i_1 + 10 i_2 + i_3$. We have that $\mathcal{A}$ admits three mode-k unfoldings, namely $A_{(1)}, A_{(2)}$ and $A_{(3)}$, where

$$A_{(1)} = \begin{pmatrix} 111 & 121 & 131 & 112 & 122 & 132 \\ 211 & 221 & 231 & 212 & 222 & 232 \end{pmatrix}$$

$$A_{(2)} = \begin{pmatrix} 111 & 211 & 112 & 212 \\ 121 & 221 & 122 & 222 \\ 131 & 231 & 132 & 232 \end{pmatrix}$$

$$A_{(3)} = \begin{pmatrix} 111 & 121 & 131 & 211 & 221 & 231 \\ 112 & 122 & 132 & 212 & 222 & 232 \end{pmatrix}$$

Note that the columns of the mode-k unfoldings contain all of the mode-k fibers, that is, the column vectors attained by fixing all indices except that of the k'th position.

Next we will define the mode-k multiplication between a tensor and a matrix. This gives us a convenient way to express basis transformations in the language of tensors.

Definition 3.2. The mode-k product of a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_d}$ and a matrix $U \in \mathbb{R}^{m_k \times n_k}$, denoted $\mathcal{A} \times_k U$ is an $(n_1 \times n_2 \times n_{k-1} \times m_k \times n_{k+1} \times \ldots \times n_d)$ tensor where elementwise, we have

$$(\mathcal{A} \times_k U)_{i_1 i_2 \ldots i_{k-1} j i_{k+1} \ldots i_d} = \sum_{i_k=1}^{n_k} A_{i_1 i_2 \ldots i_{k-1} i_k i_{k+1} \ldots i_d} U_{j i_k}$$

Equivalently, in the language of the mode-k unfoldings and regular matrix multiplication, we have that

$$(\mathcal{A} \times_k U)_{(k)} = U \cdot A_{(k)}$$

3.3 The Higher Order SVD Algorithm

For any matrix $A \in \mathbb{R}^{m \times n}$, we have that $A = USV^\top$ is an SVD of A. However, in the language of the mode-k product, this can be recast in the form

$$A = USV^\top = S \times_1 U \times_2 V.$$

This makes it convenient to express the idea row and column transformation in terms of this new product. Similarly, the Tucker Format can be expressed in terms of the mode-k product as well:

$$\mathcal{A} = S \times_1 U^{(1)} \times_2 U^{(2)} \times_3 \ldots \times_d U^{(d)}$$

This format of a tensor can be achieved via an algorithm known as the higher order singular value decomposition, or HOSVD. This is one generalization of the ordinary matrix SVD to tensors.

The HOSVD starts by storing the tensor $\mathcal{A}$ as core tensor S, which will be modified through the algorithm. Starting from $k = 1$ and iterating through

$k = d$, the algorithm proceeds by computing the mode-k unfolding $S_{(k)}$ of tensor $\mathcal{S}$ and then performing the SVD of $A_{(1)}$. We then store the U factor of the SVD and update the S tensor by multiplying it along mode k by $U^{(k)}$. The algorithm reduces the Tucker ranks of the original tensor $\mathcal{A}$ with each step of the HOSVD, working only with a smaller tensor to unfold at each step. The formal description of the algorithm is as presented in Algorithm 1.

Algorithm 1 Compute the Higher Order SVD of a tensor $\mathcal{A}$

```
Require: $\mathcal{A} = S \times_1 U^{(1)} \times_2 ... \times_d U^{(d)}$
  {Initialization}
  $S = \mathcal{A}$
  for $k = 1, 2, ...d$ do
    $S_{(k)} \leftarrow$ unfold($S$, $k$)
    $[U, \Sigma, V^\top] \leftarrow$ SVD($S_{(k)}$)
    $U^{(k)} \leftarrow U$
    $S \leftarrow S \times_k U^{(k)}$
  end for
  return $S, U^{(1)}, U^{(2)}, ..., U^{(d)}$
```

3.4 Approximation and Reapproximation

Algorithm 1 ensures an exact decomposition of $\mathcal{A}$ into its constituent factors. However, when working with finite precision arithmetic, it may be more helpful to actually truncate within some error ε close to machine precision at each iteration of the SVD. As such, we may replace the SVD() function called in each step of the HOSVD with SVD$_\delta$() where δ represents the value such that we discard all singular values larger than δ.

Theorem 3.1. *Let $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 ... \times n_d}$ be a tensor and $\mathcal{A}^{best}$ be an approximation of ε of ranks $r_1, r_2, ...r_d$ of $\mathcal{A}$. If $\hat{\mathcal{A}}$ is an approximation of $\mathcal{A}$ computed through the HOSVD with truncation ranks $r_1, r_2, ...r_d$, then*

$$||\mathcal{A} - \hat{\mathcal{A}}||_F \leq \sqrt{d}\varepsilon \tag{2}$$

Proof. It can be shown that each iteration of the truncated SVD of the of the k'th unfolding can be written as an orthogonal projection onto a subspace of matrices of rank r_k. It can also be shown that these projections freely commute with one another. We will denote each one of these projections for step k as P_k. We also have that the matrices for each of these orthogonal projections is a matrix with orthonormal columns As such, we have that

$$||\mathcal{A} - \hat{\mathcal{A}}||_F^2 = ||\mathcal{A} - P_d P_{d-1} ... P_1 \mathcal{A}||_F^2$$

We will first consider the auxiliary problem

$$||\mathcal{A} - P_d P_{d-1} ... P_k \mathcal{A}||_F^2.$$

Also note that

$$\mathcal{A} - P_d ... P_k(\mathcal{A}) = [P_d ... P_{k+1}(\mathbf{id} - P_k)(A)] + [\mathcal{A} - P_d ... P_{k+1}(\mathcal{A})]$$

By orthogonality and the Pythagorean Theorem, we have that

$$||\mathcal{A} - P_d ... P_k(\mathcal{A})||_F^2 = ||P_d ... P_{k+1}(\mathbf{id} - P_k)(A)||_F^2 + ||\mathcal{A} - P_d ... P_{k+1}(\mathcal{A})||_F^2$$

Let ε_k be the error attained in the k'th SVD truncation. We have that $\varepsilon_k \leq \varepsilon$ for $k = 1, 2, ..., d$. Then

$$\begin{aligned}
||\mathcal{A} - \hat{\mathcal{A}}||_F^2 &= ||\mathcal{A} - P_d P_{d-1} ... P_k \mathcal{A}||_F^2 \\
&= ||P_d ... P_{k+1}(\mathbf{id} - P_k(\mathcal{A}))||_F^2 + ||\mathcal{A} - P_d ... P_{k+1}(\mathcal{A})||_F^2 \\
&\leq ||(\mathbf{id} - P_k)(\mathcal{A})||_F^2 + ||\mathcal{A} - P_d ... P_{k+1}(\mathcal{A})||_F^2 \\
&= \varepsilon_k^2 + ||\mathcal{A} - P_d ... P_{k+1}(\mathcal{A})||_F^2
\end{aligned}$$

Iteration from $k = 1$ gives that

$$||\mathcal{A} - P_d ... P_1(\mathcal{A})||_F^2 \leq \sum_{k=1}^{d} \varepsilon_k \leq \sum_{k=1}^{d} \varepsilon^2 = d\varepsilon^2$$

Thus,

$$||\mathcal{A} - \hat{\mathcal{A}}||_F^2 \leq \sqrt{d}\varepsilon$$

□

From the proof, we see that the HOSVD algorithm provides a quasi-optimal approximation $\hat{\mathcal{A}}$ of certain ranks to the original tensor $\mathcal{A}$.

Now suppose we have a tensor $\mathcal{A}$ already in the Tucker format but with suboptimal Tucker ranks. If $\mathcal{A}$ is already in the Tucker format, then we can save on computational power by working only with the low-parametric format.

We begin with a decomposition of the form $\mathcal{A} = S \times_1 U^{(1)} \times_2 U^{(2)} ... \times_d U^{(d)}$ where $U^{(k)}$ does not necessarily have orthonormal columns for $k = 1, 2, ..., d$. The reapproximation works first by an orthogonalization prcoess of the $U^{(k)}$ factors. This is to ensure that errors of reapproximation do not propagate through the algorithm. We begin by orthonormalizing the columns of the matrices $U^{(k)}$. This can be done via the QR decomposition and merging the non-orthogonal factor with the core tensor to get a modified tensor S'. We then perform the HOSVD algorithm on the modified core to get $S' = T \times_1 V^{(1)} \times_2^{(2)} ... \times_d V^{(d)}$ and merge the $U^{(k)}$ and $V^{(k)}$ factors via matrix multiplication to get matrices $W^{(k)} = U^{(k)} V^{(k)}$. The decomposition $\hat{\mathcal{A}} = T \times_1 W^{(1)} \times_2 W^{(2)} ... \times_d W^{(d)}$ will be an approximation of $\mathcal{A}$. The full algorithm is presented in Algorithm 2.

Algorithm 2 Compute a reapproximation of tensor $\mathcal{A}$ already in the Tucker format

```
Require: $\mathcal{A} = S \times_1 U^{(1)} \times_2 ... \times_d U^{(d)}$ with reduced ranks $r_k$
  {Orthogonalize the $U^{(k)}$ factors}
  for k = 1, 2, ..., d do
    $Q_k, R_k := \mathrm{qr}(U^{(k)})$
    $S \leftarrow S \times_k R_k$
    $U^{(k)} \leftarrow Q$
  end for
  $[T, V^{(1)}, V^{(2)}, ...V^{(d)}]$ = approximate$(S, \varepsilon)$, as in Algorithm 1
  for k = 1, 2, ..., d do
    $W^{(k)} = U^{(k)} \cdot V^{(k)}$
  end for
  return $[T, W^{(1)}, W^{(2)}, ..., W^{(d)}]$
```

Since the error is only introduced in the core tensor, and all the other factors have been orthogonalized, the error of approximation is again, quasi-optimal (i.e. the lower rank approximation $\mathcal{B}$ satisfies $||\mathcal{A} - \mathcal{B}||_F \leq \sqrt{d}\varepsilon$).

4 The Tensor-Train Decomposition

4.1 Introduction to the Tensor-Train Decomposition

One of the unfortunate downfalls of the Tucker format is that it still suffers from the curse of dimensionality. In particular, even if the Tucker ranks are small, storing all entries of the core tensor is still exponential in d. Specifically, it uses $O(dnr + r^d)$ storage to store all values. We will now look at another decomposition of a Tensor into a low parametric format, namely the Tensor Train (TT) format.

In this section, we will use functional notation to refer to refer to an entry or a subtensor of a tensor, as will be more convenient when discussing reshapings in the TT format.

Let $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$. Elementwise, we say that $\mathcal{A}$ is in the TT format with

$$\mathcal{A}(i_1, i_2, ..., i_d) = G_1(i_1)G_2(i_2)...G_d(i_d) \tag{3}$$

where $G_k(i_k)$ is an $r_{k-1} \times r_k$ matrix. In order to ensure the resulting matrix product is a scalar, we ensure that $r_0 = r_d = 1$. In order to represent every entry, we have that G_k is actually a third order tensor of size $r_{k-1} \times n_k \times r_k$, and $G_k(i_k)$ can be thought of as the i_k'th slice of this tensor. In the index form, tensor $\mathcal{A}$ in the TT format can be written as

$$\mathcal{A}(i_1, i_2, ..., i_d) = \sum_{\alpha_0}^{r_0} \sum_{\alpha_1}^{r_1} ... \sum_{\alpha_d}^{r_d} G_1(\alpha_0, i_1, \alpha_1)G_2(\alpha_1, i_2, \alpha_2)...G_d(\alpha_{d-1}, i_d, \alpha_d) \tag{4}$$

Analogous to the Tucker case, the r_k for $k = 0, 1, ..., d$ are known as the TT ranks of the decomposition.

4.2 The Tensor Train SVD Algorithm

For the Tensor Train decomposition, we will work with a different system of matricizations from the Tucker format. Here the unfoldings we consider are as follows

$$A_k(i_1, i_2, ..., i_k; i_{k+1}, ..., i_d) = \mathcal{A}(i_1, i_2, ..., i_d) \tag{5}$$

where $(i_1, i_2, ..., i_k)$ and $(i_{k+1}, ..., i_d)$ denote multi-indices representation a position in the row and column of A_k respectively. This can be achieved in MATLAB or Julia with a call to the reshape() function.

$$A_k = \text{reshape}(\mathcal{A}, \prod_{s=1}^{k} n_s, \prod_{s=k+1}^{d} n_s)$$

Example 4.1. Let $\mathcal{A} \in \mathbb{R}^{2\times 3\times 2}$ where $\mathcal{A}_{i_1 i_2 \dots i_d} = 100 i_1 + 10 i_2 + i_3$. We have that $\mathcal{A}$ admits two matrix reshapings, namely A_1, A_2 and A_3, where

$$A_1 = \begin{pmatrix} 111 & 121 & 131 & 112 & 122 & 132 \\ 211 & 221 & 231 & 212 & 222 & 232 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 111 & 112 \\ 211 & 212 \\ 121 & 122 \\ 221 & 222 \\ 131 & 132 \\ 231 & 232 \end{pmatrix}$$

We can also consider the vectorization of $\mathcal{A}$, reshaping it into a column vector $a \in \mathbb{R}^{2\cdot 3\cdot 2}$

$$a = \begin{pmatrix} 111 \\ 211 \\ 121 \\ 221 \\ 131 \\ 231 \\ 112 \\ 212 \\ 122 \\ 222 \\ 132 \\ 232 \end{pmatrix}$$

It can also be shown that there exists a TT decomposition of tensor $\mathcal{A}$ such that the TT ranks do not exceed r_k where $r_k = \text{rank}(A_k)$. More details are to be found in [2].

Similar to how we construct the Tucker format of a tensor via the HOSVD algorithm, we will exploit the matrix SVD to build an algorithm computing a TT format of a tensor. We will first store the tensor $\mathcal{A}$ into a temporary tensor C. Iterating from $k = 1$ to $k = d - 1$, we then compute the k'th matrix reshaping of C and perform the SVD. We reshape the U factor into a the tensor core G_k of size $r_{k-1} \times n_k \times r_k$. We then update C to be the $SV^\top$ factor that came out of the SVD. At the end we return the cores G_k for $k = 1, 2, ...d$ as our decomposition. Again, we will consider SVD truncations with a truncation parameter δ.

The formal description of this algorithm is presented in Algorithm 3, paraphrased from [2].

Again, similar to the Tucker case, we have the following theorem

Theorem 4.1. *Let $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$ and assume rank bounds r_k for $k = 1, 2, ..., d$. The best approximation $\mathcal{A}^{best}$ to $\mathcal{A}$ with TT-ranks bounded by r_k always exists, and the approximation $\mathcal{B}$ produced by the TT-SVD algorithm is quasi optimal:*

$$||\mathcal{A} - \mathcal{B}||_F \leq \sqrt{d-1}||\mathcal{A} - \mathcal{A}^{\boldsymbol{best}}||_F$$

We omit the proof, but details can be found in [2].

Algorithm 3 Compute a Tensor Train format of tensor $\mathcal{A}$ with the TT-SVD algorithm, paraphrased from [2]

```
Require: $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$
  Initialization: $C \leftarrow A, r_0 = 1$
  for $k = 1, 2, ...d - 1$ do
    $C \leftarrow \text{reshape}(C, [r_{k-1}, n_k, \frac{\text{numel}(C)}{r_{k-1} n_k}])$
    $[U, S, V^\top] = \text{SVD}_\delta(C)$
    $G_k \leftarrow \text{reshape}(U, [r_{k-1}, n_k, r_k])$
    $C \leftarrow SV^\top$
  end for
  $G_d = C$
  return $[G_1, G_2, ..., G_d]$
```

4.3 The Tensor Train Rounding Process

Like in the Tucker format, we may be given a tensor $\mathcal{A}$ already in the TT format, but with suboptimal TT ranks. These decompositions can often appear when performing operations between two tensors already in the TT format, so it will be convenient to develop a procedure that reapproximates a given tensor with some prescribed accuracy ε.

Suppose that $\mathcal{A}$ is already in the TT format, but with increased ranks r_k. This procedure is intended to estimate the true value of ranks $r'_k \leq r_k$ while maintaining the prescribed accuracy ε. We can achieve this via the unfolding matrices A_k and reducing their ranks via SVD.

Consider the first unfolding matrix A_1 with rank r_1. As such, it admits a dyadic decomposition

$$A_1 = UV^\top$$

where

$$U(i_1, \alpha_1) = G_1(i_1, \alpha_1), \quad V(i_2, i_3, ..., i_d; \alpha_1) = G_2(\alpha_1, i_2)G_3(i_3)...G_d(i_d)$$

We can perform the same procedure of finding the SVD from a dyadic decomposition as presented in section 1 so that we have that

$$A_1 = \hat{U}D\hat{V}^\top$$

Matrix U is relatively small, so we can compute its QR decomposition directly, but matrix V is large, so we will compute the QR decomposition of V in an auxiliary way, via cores $G_2, G_3, ..., G_d$ of the decomposition.

The central idea of this QR factorization is a right to left sweep across all of the cores. First we consider matrix $G_d(i_d)$. We then have that

$$G_d(i_d) = R_d Q_d(i_d)$$

via a row QR decomposition of $G_d(i_d)$. We then have that

$$\begin{aligned} V(i_2, ..., i_d) &= G_2(i_2)G_3(i_3)...G_{d-1}(i_{d-1})R_d Q_d(_d) \\ &= G_2(i_2)G_3(i_3)...G'_{d-1}(i_{d-1})Q_d(_d) \end{aligned}$$

where $G'_{d-1}(i_{d-1}) = G_{d-1}(i_{d-1})R_d$. This process of finding the QR decomposition of a core and merging the non-orthogonal factor with the previous core can be done iteratively by reshaping core G_k into a matrix of size $r_{k-1} \times (n_k r_k)$ and performing a row QR decomposition on it.

At the end of the procedure, we have the Q factor of V in the TT format, and the matrix R_v stored. At the end of this process, we will have reduced r_1 and modified cores G_1andG_2. Since $G_3, ..., G_d$ now have orthonormal columns, to compress in the second mode, we just have to orthogonalize

G_1 and G_2 by storing the R_v matrix that came from the orthogonalization algorithm. This just comes down to performing subsequent SVD truncations on the next matrices and moving the non-orthogonal part forwards through the cores. A formal description of the algorithm is provided in algorithm 4.

Algorithm 4 TT-Rounding Algorithm, paraphrased from [2]

```
Require: Tensor $\mathcal{A}$ in the TT format with prescribed accuracy $\varepsilon$
  Compute the truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}}||A||_F$
  for $k = d$ to 2, step -1 do
    $[G_k(\beta_{k-1}; i_k\beta_k), R(\alpha_{k-1}, \beta_{k-1})] \leftarrow QR_{\text{rows}}(G_k(\alpha_{k-1}); i_k\beta_k)$
    $G_{k-1} \leftarrow G_{k-1} \times_3 R$, where $\times_3$ denotes the mode 3 product
  end for
  for $k = 1, 2, ...d-1$ do
    $[G_k(\beta_{k-1}i_k; \gamma_k), \Sigma, V(\beta_k, \gamma_k)] \leftarrow \text{SVD}_\delta\ (G_k(\beta_{k-1}i_k; \beta_k))$
    $G_{k+1} = G_{k+1} \times_1 \Sigma V^\top$
  end for
  return $G_k$, $k = 1, 2, ...d$ as the cores of $\mathcal{B}$
```

4.4 Basic Operations in the TT Format

In this subsection, we will discuss simple arithmetic and linear algebra operations that can be done with tensors in the TT format. We will begin with addition of two tensors in the TT format.

Let $\mathcal{A}$ and $\mathcal{B}$ be two tensors in the TT format, i.e.

$$\mathcal{A}(i_1..., i_d) = A_1(i_1)...A_d(i_d), \quad \mathcal{B}(i_1..., i_d) = B_1(i_1)...B_d(i_d)$$

By addition of tensors, we have elementwise, that

$$\mathcal{C}(i_1, i_2, ..., i_d) = \mathcal{A}(i_1, i_2, ..., i_d) + \mathcal{B}(i_1, i_2, ..., i_d)$$

This can be done by a merge of the cores of $\mathcal{A}$ and $\mathcal{B}$. For $k = 2, 3, ...d-1$, we have that

$$C_k(i_k) = \begin{pmatrix} A_k(i_k) & 0 \\ 0 & B_k(i_k) \end{pmatrix}.$$

For cores C_1 and C_d, we have that

$$C_1(i_1) = \begin{pmatrix} A_1(i_1) & B_1(i_1) \end{pmatrix} \quad C_d(i_d) = \begin{pmatrix} A_d(i_d) \\ B_d(i_d) \end{pmatrix}$$

Indeed, by matrix multiplication, we do recover that

$$C_1(i_1)C_2(i_2)...C_d(i_d) = A_1(i_1)A_2(i_2)...A_d(i_d) + B_1(i_1)B_2(i_2)...B_d(i_d)$$

As seen here, the merge of cores can as much as double the TT ranks. The TT-rounding procedure will be helpful in maintaining low-rank structure in the decomposition.

Multiplication by a scalar in the TT format is simple, as it only comes down to scaling any one of the cores by that scalar.

We will now discuss the matrix by vector product in the TT format. In many problems of machine learning and quantum mechanical systems, exponentially large matrices and vectors may arise as representations of data, so we will discuss how to perform matrix-vector multiplication between matrix M and vector x in the TT format.

In this context, we assume a vector of length $N = n_1...n_d$ can be reshaped as a d-dimensional tensor with mode sizes n_k and represented in the TT-format with cores $X_1, X_2, ...X_d$ and elements $X(i_1, ..., i_d)$. Similarly, we say that M is in the TT-format if its elements are defined as

$$M(i_1, ..., i_d; j_1, ..., j_d) = M_1(i_1, j_1)...M_d(i_d, j_d)$$

where $M_k(i_k, j_k)$ is an $r_{k-1} \times r_k$ matrix and (i_k, j_k) is a multi-index. If $y = Mx$, we can also represent y in the TT format with entries $Y(i_1, ..., i_d)$. Elementwise, we have that

$$Y(i_1, i_2, ..., i_d) = \sum_{j_1, j_2, ..., j_d} M_1(i_1, ..., i_d; j_1, ..., j_d)X(j_1, ..., j_d)$$

The resulting tensor will also be in the TT-format. We have that

$$\begin{aligned} Y(i_1, ..., i_d) &= \sum_{j_1,...,j_d} M_1(i_1, j_1)...M_d(i_d, j_d)X_1(j_1)...X_d(j_d) \\ &= \sum_{j_1,...,j_d} (M_1(i_1, j_1)...M_d(i_d, j_d)) \otimes (X_1(j_1)...X_d(j_d)) \\ &= \sum_{j_1,...,j_d} (M_1(i_1, j_d) \otimes X_1(j_1))...(M_d(i_d, j_d) \otimes X_d(j_d)) \\ &= Y_1(i_d)...Y_d(i_d), \end{aligned}$$

where

$$Y_k(i_k) = \sum_{j_k} (M_k(i_k, j_k) \otimes X_k(j_k)).$$

Again, this process may result in cores Y_k with suboptimal ranks, so the TT-rounding algorithm may be applied again to estimate lower ranks.

We have now presented a short sample of the operations that can be done with tensors in the TT format. For a fuller list of operations, including multidimensional contraction, the Hadamard product, and scalar product, refer to [2].

5 Machine Learning Application: Supervised Learning with Tensorized Linear Systems

Most recently, one of the more promising applications of tensor decompositions is in the solution of large linear systems. This often arises in problems of non-linear kernal learning, where input kernal vectors x are mapped into a high dimensional space with a feature map before being classified by a decision function into N_L distinct classes (or labels). We consider the following model:

$$f(x) = W \cdot \Phi(x) \tag{6}$$

where $x \in \mathbb{R}^N$ represent a vector input (say, the pixels of an image in some vectorized form), $\Phi : \mathbb{R}^N \to \mathbb{R}^{\times_{k=1}^{d} N}$ is the high dimensional mapping into feature space, $W \in \mathbb{R}^{N_L \times d^N}$ is the matrix of weights, and $f : \mathbb{R}^N \to \mathbb{R}^{N_L}$ is the overall function mapping each input to a vector of labels. We will interpret the resulting vector $f(x)$ to be a vector of entries containing the likelihood that a given input belogs to a particular class l for $l \in \{1, 2, ..., N_L\}$.

Remark 5.1. *The notation for the function $\Phi : \mathbb{R}^N \to \mathbb{R}^{\times_{k=1}^{d} N}$ indicates that Φ maps from an $N-$vector to a $d-$order tensor with mode size N in each dimension. The choice of Φ can vary depending on application, and more research is necessary to understand which choice of Φ is most appropriate for a given context.*

Suppose that we have N_T training examples labelled $\{x_n, y_n\}$ for $n = 1, 2, ..., N_T$, where x_n denotes the n'th input and y_n is the corresponding n'th output. We want to minimize the cost function

$$C = \frac{1}{2} \sum_{n=1}^{N_T} ||f(x_n) - y_n||_2^2 \tag{7}$$

in the 2-norm. This comes down to finding the best weight matrix W such that the cost function has the least error. By applying the definition of f, we have the following expression for the cost function:

$$C = \frac{1}{2}\sum_{n=1}^{N_T} ||W \cdot \Phi(x_n) - y||_2^2 \tag{8}$$

Now, consider $z_n = \text{vec}(\Phi(x_n))$. This turns the high dimensional tensor $\Phi(x_n)$ into a column vector of size d^N. We then have that

$$\begin{aligned} C &= \frac{1}{2}\sum_{n=1}^{N_T} ||W z_n - y||_2^2 \\ &= \frac{1}{2}||Z \cdot w - Y||_2^2 \end{aligned}$$

$w = \text{vec}(W) \in \mathbb{R}^{N_L \cdot d^N}$,

$$Z = \begin{pmatrix} I \otimes z_1^\top \\ I \otimes z_2^\top \\ \vdots \\ I \otimes z_{N_T}^\top \end{pmatrix} \in \mathbb{R}^{N_L \cdot N_T \times N_L \times d^N},$$

and

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N_T} \end{pmatrix} \in \mathbb{R}^{N_L \cdot N_T}$$

Since we minimize this quadratic cost, this comes down to finding the least squares solution to the system

$$Zw = Y. \tag{9}$$

Solving this linear systems using traditional numerical linear algebra techniques are computationally infeasible due to the large dimensions of Z and w. However, if we have Z, w and Y in the tensor train format, we may use algorithms like the Linear Alternating Scheme or the Density Matrix Renormalization Group to find approximate solutions to the system. These algorithms rely on optimizing a few cores of w at a time, approaching a least squares solution to the system. Convergence properties of these algorithms are still not very well studied, but have promising results in practice. For more detailed information on these algorithms, view [4] and [5].

6 Conclusion

This SURIM project was intended as a literature review on existing works regarding two well-known tensor decompositions, namely the Tucker and Tensor Train formats of a tensor. It is clear that the Tensor Train format gives huge advantages in terms of storage and computational complexity, but combining this format with say the Tucker format, it is possible to decompose each one of the TT cores with the HOSVD algorithm, further reducing the storage complexity.

An interesting theoretical problem discovered in this project was perhaps something overlooked in the proof of the quasi-optimality bound in the approximation that comes out of the HOSVD algorithm, namely the sharpness of the bound. Presented in this paper is a classic proof of the bound, but something worth considering is the independence of each projection operation P_k. It is possible that the SVD truncations of the unfoldings are not independent of each other, so that there might be a more optimistic bound on the error arising out of each truncation.

As for future work related to more applied contexts, we hope to learn more about and understand the tensor methods used in solving huge linear systems, namely the Alternating Linear Scheme and the DMRG algorithms, as well as how they relate to problems in supervised machine learning tasks as in [3]. In addition, we hope to learn more about the convergence of these methods and how to achieve faster or more accurate results for the convergence of these methods.

7 Acknowledgements

This paper is dedicated to Joseph Tran. Thank you for being an amazing friend and roommate this summer.

I would like to thank my project advisor Dr. Vladimir Kazeev who has been instrumental in the success of this project. From the introduction to the project to refining my presentation skills, Dr. Kazeev has helped me incredibly in all aspects of this project. I'd also like to thank Dr. Chris Ohrt for running the 2019 SURIM Program and for proofreading the first draft of this paper. Finally, I'd also like to thank my fellow SURIM colleagues who listened to and asked questions at my midterm and final talks.

References

[1] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. ”A Multilinear Singular Value Decomposition”. SIAM J. Matrix Anal. Appl. 21 (2000), no. 4, 1253–1278.

[2] I. V. Oseledets. ”Tensor-Train Decomposition”. SIAM J. Sci. Comput. 33, 2295–2317 (2011).

[3] E. M. Stoudenmire and David J. Schwab. ”Supervised Learning with Tensor Networks”. Advances in Neural Information Processing Systems 29 (2016), 4799–4807

[4] I. V Oseledets and S. V. Dolgov. ”Solution of Linear Systems and Matrix Inversion in the TT-Format”. SIAM J. Sci. Comput. 34, 2718–2739 (2012)

[5] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. ”The Alternating Linear Scheme for Tensor Optimization in the Tensor Train Format”. SIAM J. Sci. Comput. 34, 683–713 (2012)