8-2018

Multi-dimensional data analytics and deep learning via tensor networks

Wenqi Wang
Purdue University

MULTI-DIMENSIONAL DATA ANALYTICS AND DEEP LEARNING VIA TENSOR NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Wenqi Wang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

ACKNOWLEDGMENTS

First of all, I would like to express my sincerest gratitude to my advisor Vaneet Aggarwal. I was so lucky to meet him during my graduate class Stochastic Process in the Fall 2015 and started my research in machine learning and deep learning since then. His patience during my early Ph.D. life, supervision in my research, and guidance in my career development enlighten my way of thinking and shape my philosophy in academic research. I especially thank him for giving me the academic advice at the start and academic freedom as I grow, which allows me to work on different projects of interest. Without him, I could not enter the world of machine learning and complete my thesis.

I would like to thank my collaborator Shuchin Aeron for his mentoring in my research. His guidance in my first several research works lays the foundation for my fundamental knowledge in optimization and multi-dimensional data analytics. Without the insightful thoughts and fruitful discussion from him, I could not make so much progress in my research. Also, I want to thank the committee members, Gesualdo Scutari and Christopher Quinn, for their valuable comments and thoughtful suggestions to my thesis.

I would like to thank Brian Eriksson and Yifan Sun for offering me the great internship at Technicolor Research AI Lab. It is a valuable learning and collaborating experience, and I would like to especially thank Brain Eriksson for his mentoring on my presentation techniques and Yifan Sun for her assistance in improving my writing skills.

Next, I gratefully acknowledge the Bilsland Dissertation Fellowship from Purdue University that supports my research and dissertation in my final year. This dissertation would not be possible without the funding support.

Last but not least, I would like to sincerely thank my father Shengguo Wang and my mother Yumei Shao for their unconditional love and support throughout my life, my twin brother Wenlin Wang for his constant accompanying in my growth and selfless sharing of knowledge, and my beloved girlfriend Xiudan Wang for her persistent trust, love, and encouragement. I would like to thank two more important persons in my life, my aunt Xiurong Wang for taking care of me like her child for more than six years from my elementary school age to high school age, and Yanping Wang for her love and support in every possible way.

Wenqi Wang

Purdue University

August, 2018

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

SYMBOLS

$x \in \mathbb{R}$	a scaler
$\mathbf{x} \in \mathbb{R}^n$	a vector
$\mathbf{X} \in \mathbb{R}^{m \times n}$	a matrix
$\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$	a d-order tensor
$\mathcal{X}(i_1, \cdots, i_d)$	an element in tensor $\mathcal{X}$
$\mathcal{X}(i_1, \cdots, i_{k-1}, :, i_{k+1}, \cdots, i_d)$	a fiber along mode k in tensor $\mathcal{X}$
$\mathcal{X}(i_1, \cdots, i_{k-1}, :, :, i_{k+2}, \cdots, i_d)$	a slice along mode k and $k+1$ in tensor $\mathcal{X}$
$\mathbf{V}(\bullet)$	vectorization operator
$\mathrm{tr}_i^j(\bullet)$	trace operator along mode i and j
$\times$	standard matrix product operation
	kronecker product

ABBREVIATIONS

TT	Tensor Train
TR	Tensor Ring
TN	Tensor Networks
ATN	Approximated Tensor Networks
TTS	Tensor Train Subspace
PCA	Principle Component Analysis
NPE	Neighbor Preserving Embedding
KNN	K-Nearest Neighbor
ALS	Alternating Least Square
FCL	Fully Connected Layer
ConL	Convolutional Layer
TRN	Tensor Ring Nets

ABSTRACT

Wang, Wenqi PhD, Purdue University, August 2018. Multi-dimensional data analytics and deep learning via tensor networks. Major Professor: Vaneet Aggarwal.

With the booming of big data and multi-sensor technology, multi-dimensional data, known as *tensors*, has demonstrated promising capability in capturing multi-dimensional correlation via efficiently extracting the latent structures, and drawn considerable attention in multiple disciplines such as image processing, recommender system, data analytics, etc. In addition to the multi-dimensional nature of real data, artificially designed tensors, referred as *layers* in deep neural networks, have also been intensively investigated and achieved the state-of-the-art performance in imaging processing, speech processing, and natural language understanding.

However, algorithms related with multi-dimensional data are unfortunately expensive in computation and storage, thus limiting its application when the computational resources are limited. Although tensor factorization has been proposed to reduce the dimensionality and alleviate the computational cost, the trade-off among computation, storage, and performance has not been well studied.

To this end, we first investigate an efficient dimensionality reduction method using a novel Tensor Train (TT) factorization. In particular, we propose a Tensor Train Principal Component Analysis (TT-PCA) and a Tensor Train Neighborhood Preserving Embedding (TT-NPE) to project data onto a Tensor Train Subspace (TTS) and effectively extract the discriminative features from the data. Mathematical analysis and simulation demonstrate TT-PCA and TT-NPE achieve better trade-off among computation, storage, and performance than the bench-mark tensor-based dimensionality reduction approaches. We then extend the TT factorization into general Tensor Ring (TR) factorization and propose a tensor ring completion algorithm, which can utilize 10% randomly observed pixels to recover the gunshot video at an error rate of only 6.25%. Inspired by the novel trade-off between model complexity and data representation, we introduce a Tensor Ring Nets (TRN) to compress the deep neural networks significantly. Using the benchmark 28-layer WideResNet architectures, TRN is able to compress the neural network by 243$\times$ with only 2.3% degradation in Cifar10 image classification.

1. INTRODUCTION

1.1 Motivation

The fast development of online data collection, the widespread use of multi-sensor technology, and the emergence of the large datasets have enabled the possibilities of utilizing a tremendous amount of data. Unlike the common one-dimensional or two-dimensional data that are in the form of *vectors* or *matrices*, the current large-scale datasets are collected with multi-dimensional correlated information and modeled into multi-dimensional arrays, also referred as *tensors*. For instance, in recommender system [1, 2], conventional recommender system only deals with users and items (posts, movies, musics, etc.) while the modern context-aware recommender system is developed to make personalized recommendation by considering context information, such as location (private location or public location), time (daytime or night), devices (mobile devices or desktop), etc. Similarly, in modern healthcare imaging, such as Electroencephalography (EEG) [3] or Functional Magnetic Resonance Imaging (fMRI) [4], the video data is collected by using multiple sensors to record the bio-signal from different domains of human bodies simultaneously, and the spatial and sequential information the data consequently increases the dimensionality of the data. Other popular examples are self-driving cars [5], which employ multiple cameras to record live videos in the real-time and form the data in the multi-dimensional format.

In addition to the multi-dimensional nature of the large newborn datasets, artificially designed tensors have also been widely employed in Deep Neural Networks (DNN). Convolutional Neural Networks (CNN) use tensors as *kernels* to exploit local connectivity pattern between neurons of adjacent layers and achieve state-of-the-art performance in multiple domains, including image classification [6–8], object detection [9–12], speech recognition [13], etc.

However, although tensors are efficient in modeling the multi-dimensional data, algorithms related with multi-dimensional data are unfortunately expensive in computation and storage. For example, a standard 100 frames color video with hight and width of 256 has approximate 20 million pixels, and a conventional convolution layer of shape $3 \times 3 \times 512 \times 512$ has approximate 2 million parameters. The problems associated with the massive cost in computation and storage become more severe on resources limited devices, such as mobile devices, smart watches, etc. Tensor factorization approaches have been investigated to represent the tensor data via low-rank tensor factors, thus reducing the computational complexity, while selecting a suitable latent structure to capture the low-rank property of tensor data and providing a better trade-off among computation, storage, and performance have not been well studied.

To this end, we investigate a particular class of hieratical Tensor Networks (TN), specifically Tensor Train (TT) and Tensor Ring (TR) representations, apply them into multi-dimensional data analytics and deep neural networks, and achieve superior performance of trade-off in multiple disciplines. We summarize our main contribution as follows:

- For efficient dimensionality reduction, we propose a new subspace embedded with tensor train structures, denoted as Tensor Train Subspace (TTS), which is useful in extracting the discriminative features from noise multi-dimensional tensor data. With tensor train subspace, we further investigate two efficient dimensionality reduction methods, Tensor Train Principle Component Analysis (TTPCA) and Tensor Train Neighbor Preserving Embedding (TT-NPE), to efficiently reduce the dimensionality and computational complexity for the tensor data.

- For efficient multi-dimensional data representation, we investigate an efficient low-rank Tensor Ring (TR) completion algorithm to recover the multi-dimensional data using only a small portion of partially observed entries. We demonstrate tensor ring completion is better than the benchmark tensor completion algorithm in various multi-dimensional data sets, including image, image sets, and videos.

- For the implementation of multi-dimensional data computation on resources limited devices, we introduce a new Tensor Ring Network (TRN) that can layer-wise compress deep neural networks significantly. We further proposed a more efficient convolutional operation between input data and the tensor ring layers. TRN outperforms than the other benchmark tensor compression approaches and demonstrates promising capability in model compressing.

Given the motivation of the thesis, the remaining of the thesis is organized as follows: We first introduce the background about tensor decomposition in Section 1.2 and tensor networks in Section 1.3, followed by the introduction on notation in Section 1.4. In Chapter 2 we introduce the concept of the tensor train subspace, and its application in dimensionality reduction. In partial, we investigate a Tensor Train Principle Component Analysis (TT-PCA) algorithm that reduces dimensionality of a tensor data by projecting data onto a tensor train subspace [14]. We then investigate a non-linear tensor train embedding algorithm that reduces the dimensionality of a tensor data by preserving its neighborhood information, namely Tensor Train Neighborhood Preserving Embedding (TT-NPE) [15]. In Chapter 3 we investigate a more general tensor ring structured and proposed a tensor ring completion algorithm [16]. In Chapter 4 we develop a tensor ring nets that layer-wise compress neural networks and maintain a good trade-off in computational resources and performance [17]. Chapter 5 concludes the thesis.

We highlight that in Chapter 2, TT-PCA algorithm is based on [14] [1] *and TT-NPE is based on [15]* [2]*. Chapter 3* [3] *and Chapter 4* [4] *are based on [16] and [17] respectively.*

1.2 Tensor Decomposition

1.2.1 CANDECOMP/PARAFAC (CP) Decomposition

Definition 1.2.1 *(CANDECOMP/PARAFAC(CP) Decomposition [18]) Each element of a n-mode tensor $\mathcal{Y} \in \mathbb{R}^{I_1\times\cdots\times I_n}$ is generated by*

$$\mathcal{Y}(i_1, \cdots, i_n) = \mathop{}_{r=1}^{R} \mathbf{A}_1(i_1, r)\mathbf{A}_2(i_2, r)\cdots\mathbf{A}_n(i_n, r), \tag{1.1}$$

where $\mathbf{A}_i \in \mathbb{R}^{I_i\times R}$ are the tensor factors of CP decomposition, and R is the CP-rank.

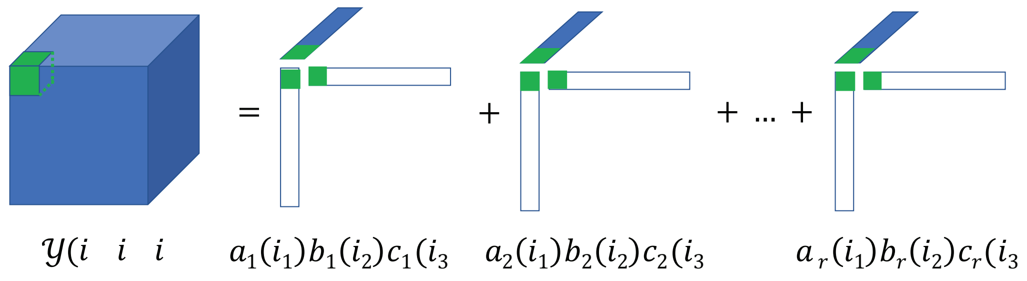

=
+
+ ... +
$\mathcal{Y}(i \quad i \quad i$
$a_1(i_1)b_1(i_2)c_1(i_3$
$a_2(i_1)b_2(i_2)c_2(i_3$
$a_r(i_1)b_r(i_2)c_r(i_3$

Fig. 1.1.: CP decomposition for a 3-mode tensor.

Fig 1.1 provides an example of a 3-order tensor that with CP rank r, and the highlighted green entry is the summation of r entries that generated by the outer-product of CP factors.

[1]A version of this Chapter is under review by Pattern Recognition Letters.

[2] c 2018 IEEE. Reprinted with permission from Wenqi Wang, Tensor Train Neighborhood Preserving Embedding, IEEE Transactions on Signal Processing, 2018

[3] c 2018 IEEE. Reprinted with permission from Wenqi Wang, Efficient Low Rank Tensor Ring Completion, IEEE International Conference on Computer Vision, 2017

[4] c 2018 IEEE. Reprinted with permission from Wenqi Wang, Wide Compression: Tensor Ring Nets, IEEE Conference on Computer Vision and Pattern Recognition, 2018

1.2.2 Tucker Decomposition

Definition 1.2.2 *(Tucker Decomposition [19,20]) Element in a n-mode tensor tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ is generated by*

$$\mathcal{Y}(i_1, \cdots, i_n) = \sum_{r_1=1}^{R_1} \cdots \sum_{r_n=1}^{R_n} \mathcal{C}(r_1, \cdots, r_n)\mathbf{A}_1(i_1, r_1)\mathbf{A}_2(i_2, r_2) \cdots \mathbf{A}_n(i_n, r_n), \tag{1.2}$$

where the n-order tensor $\mathcal{C} \in \mathbb{R}^{R_1 \times \cdots \times R_n}$ is the core *tensor of Tucker decomposition, and the array $[R_1, R_2, \cdots, R_n]$ is the Tucker-rank.*

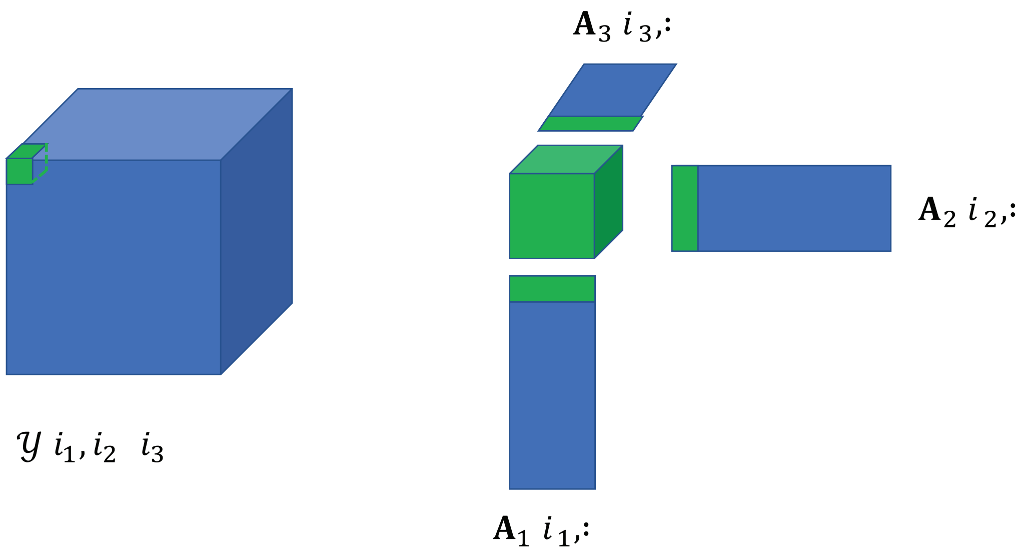


Fig. 1.2.: Tucker decomposition for a 3-mode tensor.

In Fig 1.2 a Tucker decomposition for a 3-order tensor is provided, and a larger tensor $\mathcal{Y}$ is projected into a small core tensor $\mathcal{C}$ through a sequence of linear transformation $\mathbf{A}_1$,$\mathbf{A}_2$, and $\mathbf{A}_3$.

It is worthy to point out that with the constraint that when $\mathcal{C}$ is a *super-symmetric* diagonal tensor, which satisfies $R_1 = R_2 = \cdots = R_n$ and

$$\mathcal{C}(i_1, \cdots, i_n) \begin{cases} = 0, & i_1 = i_2 = \cdots = i_n, \\ = 0, & \text{otherwise.} \end{cases} \tag{1.3}$$

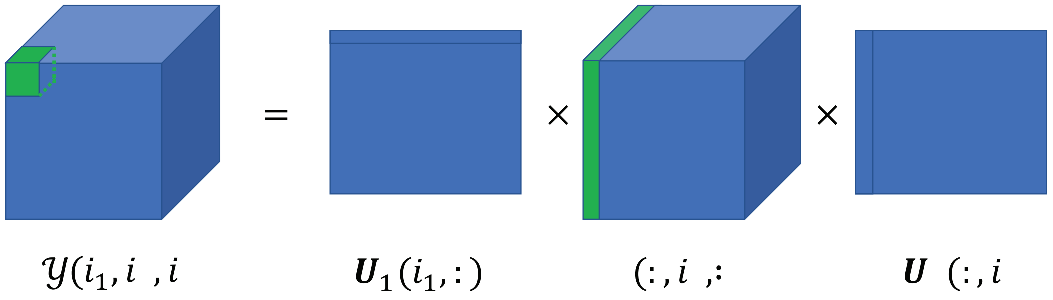
(1.3) is the same as (1.1), thus *CP decomposition is a Tucker decomposition with diagonal core tensor.*

1.2.3 Tensor Train (TT) Decomposition

Definition 1.2.3 *(Tensor Train Decomposition [21, 22]) Each element of a n-mode tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ in tensor train representation is generated by*

$$\mathcal{Y}(i_1, \cdots, i_n) = \mathbf{U}_1(i_1, :)\mathcal{U}_2(:, i_2, :) \cdots \mathcal{U}_{n-1}(:, i_{n-1}, :)\mathbf{U}_n(:, i_n), \tag{1.4}$$

where $\mathbf{U}_1 \in \mathbb{R}^{I_1 \times R_1}$ and $\mathbf{U}_n \in \mathbb{R}^{R_{n-1} \times I_n}$ are the boundary matrices and $\mathcal{U}_i \in \mathbb{R}^{R_{i-1} \times I_i \times R_i}$, $i =$ $2, \cdots, n-1$ are the decomposed tensors. The tensor train rank is the array $[R_1, R_2, \cdots, R_{n-1}]$.



=
×
×
$\mathcal{Y}(i_1, i\ \ , i$
$\boldsymbol{U}_1(i_1, :)$
$(:, i\ \ , :$
$\boldsymbol{U}\ \ (:, i$

Fig. 1.3.: Tensor train decomposition for a 3-mode tensor.

Tensor train decomposition for a 3-mode tensor $\mathcal{Y}$ is illustrated in Fig. 1.3, where $\mathcal{Y}(i_1, i_2.i_3)$ is the sequential matrix product of vector $\mathbf{U}_1(i_1, :)$, matrix $\mathcal{U}_2(:, i_2, :)$, and vector $\mathbf{U}_3(:, i_3)$.

1.2.4 Tensor Ring (TR) Decomposition

Definition 1.2.4 *(Tensor Ring Decomposition [23]) Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ be an n-order tensor with I_i-dimension along the i_{th} mode, then any entry inside the tensor, denoted as $\mathcal{X}(i_1, \cdots, i_n)$, is represented by*

$$\mathcal{X}(i_1, \cdots, i_n) = \overset{R_1}{\underset{r_1=1}{}} \cdots \overset{R_n}{\underset{r_n=1}{}} \mathcal{U}_1(r_n, i_1, r_1) \cdots \mathcal{U}_n(r_{n-1}, i_n, r_n), \tag{1.5}$$

where $\mathcal{U}_i \in \mathbb{R}^{R_{i-1} \times I_i \times R_i}$ *is a set of 3-order tensors, also named matrix product states (MPS), which consist the bases of the tensor ring structures. Note that* $\mathcal{U}_j(:, i_j, :) \in \mathbb{R}^{R_{j-1} \times 1 \times R_j}$ *can be regarded as a matrix of size* $\mathbb{R}^{R_{j-1} \times R_j}$*, thus* (1.5) *is equivalent to*

$$\mathcal{X}(i_1, \cdots, i_n) = tr(\mathcal{U}_1(:, i_1, :) \times \cdots \times \mathcal{U}_n(:, i_n, :)). \tag{1.6}$$

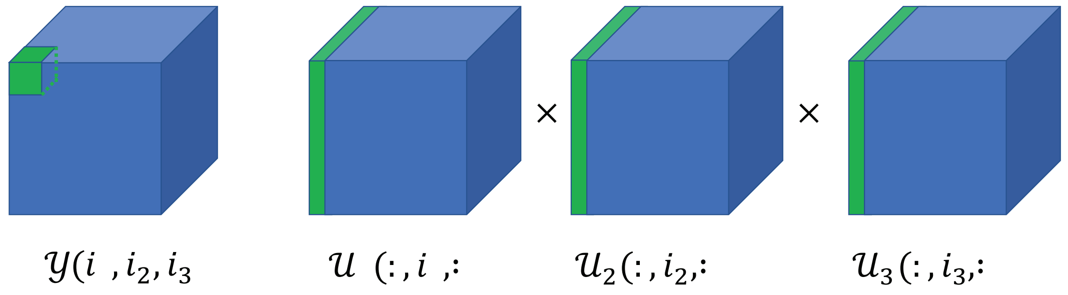


Fig. 1.4.: Tensor ring decomposition for a 3-mode tensor.

Tensor ring decomposition for a 3-mode tensor $\mathcal{Y}$ is illustrated in Fig. 1.4, where $\mathcal{Y}(i_1, i_2.i_3)$ is the trace of a matrix that is the sequential matrix product of matrix $\mathcal{U}_1(:, i_1, :)$, matrix $\mathcal{U}_2(:, i_2, :)$, and matrix $\mathcal{U}_3(:, i_3, :)$.

Remark 1 *(Tensor Ring Rank (TR-Rank)) In the formulation of tensor ring, we note that tensor ring rank is the vector* $[R_1, \cdots, R_n]$*. In general,* R_i *are not necessary to be the same. In our set-up, we set* $R_i = R\ \forall i = 1, \cdots, n$*, and* **the scalar** R **is referred as the tensor ring rank***.*

Remark 2 *(Tensor Train [21]) Tensor train is a special case of tensor ring when* $R_n = 1$*.*

1.3 Tensor Networks (TN)

Unlike single dimensional vectors or two dimensional matrices that operations between two objects can be pair-wise depicted efficiently, the operation among tensors construct a network structure and pair-wise relationship is not sufficient to capture

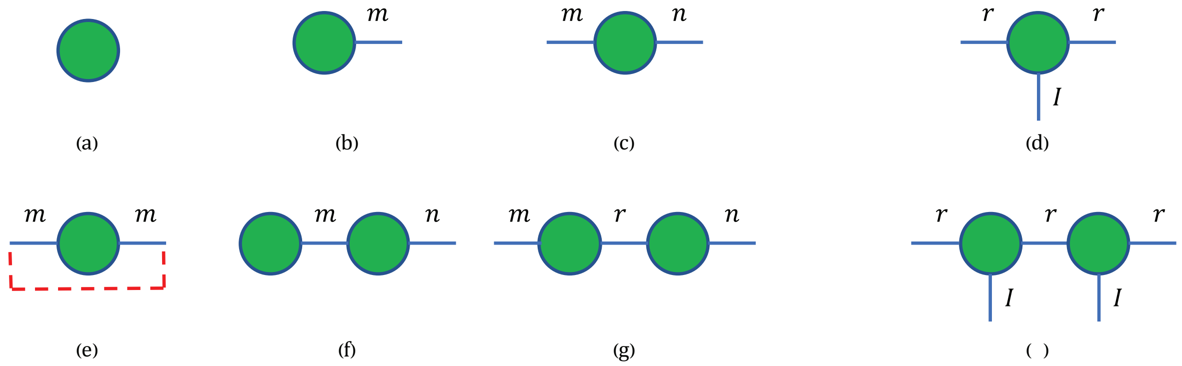
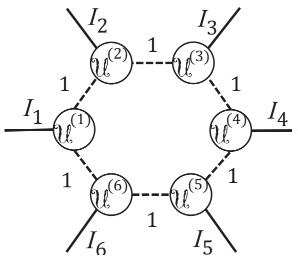
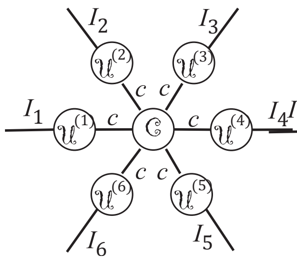


Fig. 1.5.: Tensor network notations. (a) a scalar $s \in \mathbb{R}^0$, (b) a vector $\mathbf{v} \in \mathbb{R}^m$, (c) a matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$, (d) a tensor $\mathcal{T} \in \mathbb{R}^{r\times I\times r}$, (e) trace operation $\mathrm{tr}(\mathbf{M})$, (f) vector to matrix product between $\mathbf{v} \in \mathbb{R}^m$ and $\mathbf{M} \in \mathbb{R}^{m\times n}$, (g) matrix to matrix product between $\mathbf{M}_1 \in \mathbb{R}^{m\times r}$ and $\mathbf{M}_2 \in \mathbb{R}^{r\times n}$, (h) tensor to tensor product between $\mathcal{U}_1 \in \mathbb{R}^{r\times I\times r}$ and $\mathcal{U}_2 \in \mathbb{R}^{r\times I\times r}$.

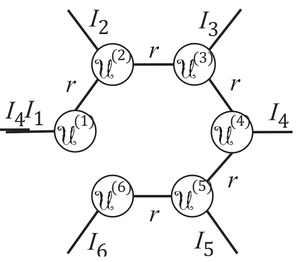
the structure within the tensor data. Tensor network is a graphical representation of multi-dimensional data and multi-dimensional data operations, and it enables the multi-dimensional data analytics via a network analysis. In tensor network notations [24, 25], each node represents a tensor and the number of edges determines the mode of a tensor. For instance, a scaler is a tensor of mode 0, which is thus a tensor without any edges as illustrated in Fig 1.5(a), and a vector is a tensor of mode 1, which is consistent with the node with only one edge in Fig 1.5(b). The same logic applies to high order tensors, thus a matrix of shape $m\times n$, and a 3-order tensor of shape $r\times I\times r$ are equivalent to the graphical representation of the tensor network notations in Fig 1.5 (c) and Fig 1.5 (d) respectively. The edge connecting two nodes is the operation of tensor merging product, which is the summation between the particular dimensions of the two connected tensors. Fig 1.5 (e-h) are the graphical representation of popular multi-dimensional data operations, including matrix trace, vector inner product $\mathbf{x}\ \ \mathbf{y}$, matrix to matrix product $\mathbf{AB}$ and tensor to tensor production $\mathcal{U}_1\mathcal{U}_2$ [16]. Using the tensor network representation, the common tensor factorization can be graphically described in 1.6, which gives a better comparison among the different models.



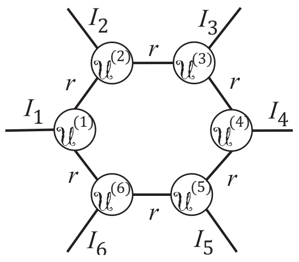
(a) CP Decomposition



(b) Tucker



(c) Tensor Train (TT)



(d) Tensor Ring(TR)

Fig. 1.6.: **Tensor decompositions.** Tensor diagrams for four popular tensor factorization methods: (a) the CP decomposition (unnormalized), (b) the Tucker decomposition, (c) the Tensor Train (TT) decomposition, and (d) the Tensor Ring (TR) decomposition. As shown, TR can be viewed as a generalization of both CP (with $r > 1$) and TT (with an added edge connecting the first and last tensors). We also compare against Tucker decomposition compression schemes.

1.4 Notations

We now introduce the set of notation we uses throughout the paper. Vectors and matrices are represented by boldface lower letters (e.g. $\mathbf{x}$) and boldface capital letters (e.g. $\mathbf{X}$), respectively. An n-order tensor is denoted by calligraphic letters $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_n}$, where I_i, $i = 1, 2, \ldots, n$ denotes the dimensionality along the i_{th} order. An element of a tensor $\mathcal{X}$ is represented as $\mathcal{X}(i_1, i_2, \cdots, i_n)$, where i_k, $k = 1, 2, .., n$ denotes the location index along the k_{th} order. A colon is applied to represent all the elements of an order in a tensor, e.g. $\mathcal{X}(:, i_2, \cdots, i_n)$ represents the fiber along order 1 and $\mathcal{X}[:, :, i_3, i_4, \cdots, i_n]$ represents the slice along order 1 and order 2 and so forth. $\mathbf{V}(\cdot)$ is a tensor vectorization operator such that $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ is mapped to a vector $\mathbf{V}(\mathcal{X}) \in \mathbb{R}^{I_1 \cdots I_n}$. $\times$ and $\otimes$ represent matrix product and kronecker product respectively. Let tr_i^j be a tensor trace operation, which reduces 2 tensor orders by getting the trace along the slices formed by the i_{th} and j_{th} order (assuming $I_i = I_j$). As an example, let $\mathcal{U} \in \mathbb{R}^{I_1 \times I_2 \times I_1}$ be a 3-mode tensor, then $\mathbf{v} = \text{tr}_1^3(\mathcal{U}) \in \mathbb{R}^{I_2}$ is given as $\mathbf{v}(i_2) = \text{trace}(\mathcal{U}(:, i_2, :)), i_2 = 1, \cdots, I_2$.

2. DIMENTIONALITY REDUCTION

2.1 Introduction

Robust feature extraction and dimensionality reduction are among the most fundamental problems in machine learning and computer vision. Assuming that the data is embedded in a low-dimensional subspace, popular and effective methods for feature extraction and dimensionality reduction are the Principal Component Analysis (PCA) [26, 27], and the Laplacian eigenmaps [28]. However, simply projecting data to a low dimensional subspace may not efficiently extract discriminative features. Motivated by recent works [15, 29, 30] that demonstrate applying tensor factorization (after reshaping matrices to multidimensional arrays or tensors) improves data representation, we consider reshaping vision data into tensors and embedding the tensors into Kronecker structured subspaces, i.e. tensor subspaces, to further refine these subspace based approaches with significant gains. In this context, a very popular representation format namely Tucker format has shown to be useful for a variety of applications [31–34]. However, Tucker representation is exponential in storage requirements [35]. In [22], it was shown that hierarchical Tucker representation, and in particular Tensor Train (TT) representation is a promising format for the approximation of solutions in high dimensional data and can alleviate the curse of dimensionality under fixed rank, which inspires us to investigate its application in efficient dimensionality reduction and embedding. Tensor train representation has also been shown to be useful for dimensionality reduction in [17, 36, 37].

In this section, we begin by noting that TT decompositions are associated with a structured subspace model, namely the Tensor Train subspace [38]. Using this notion, we extend a popular approach, namely the Neighborhood Preserving Embedding (NPE) [39] for unsupervised classification of data. In the past, the NPE approach has been extended to exploit the Tucker subspace structure on the data [40, 41]. Here, we embed the data into a Tensor Train subspace and propose a computationally efficient Tensor Train Neighbor Preserving Embedding (TTNPE) algorithm. We show that this approach achieves significant improvement in the storage of embedding and computation complexity for classification after embedding as compared to the embedding based on the Tucker representation in [40, 41]. An approximation method for TTNPE, called TTNPE-ATN (TTNPE- Approximated Tensor Networks) is provided to decrease the computational time for embedding the data. We validate the approach on classification of MNIST handwritten digits data set [42], Weizmann Facebase [43], and financial market dataset.

The key contributions of this paper are as follows. (i) We formulate the problem of embedding the data into a low-rank Tensor Train subspace, and propose a TTNPE algorithm for embedding the data. (ii) We give an approximation method to the embedding algorithm, TTNPE-ATN, to achieve faster computational time. (iii) We show that embedding based on TTNPE-ATN achieves significant improvement in the storage of embedding and computation complexity for classification after embedding as compared to the embedding based on the Tucker representation. Finally, the results on the different datasets show significant improvement in classification accuracy, computation and storage complexities for a given compression ratio, as compared to the baselines.

2.2 Related Work

We consider a tensor train decomposition for a tensor data set, which is an $n+1$ mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times R_n}$, where each element is represented as

$$\mathcal{X}(i_1, \cdots, i_n, r_n) = \mathbf{U}_1(i_1, :)\mathcal{U}_2(:, i_2, :) \cdots \mathcal{U}_{n-1}(:, i_{n-1}, :)\mathcal{U}_n(:, i_n, r_n). \tag{2.1}$$

Without loss of generality, we let $R_0 = 1$ and define $\mathcal{U}_1 \in \mathbb{R}^{R_0 \times I_1 \times R_1}$ as the tensor representation of $\mathbf{U}_1$. Thus, the tensor train decomposition for $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times R_n}$ is

$$\mathcal{X}(i_1, \cdots, i_n, r_n) = \mathcal{U}_1(1, i_1, :) \cdots \mathcal{U}_{n-1}(:, i_{n-1}, :)\mathcal{U}_n(:, i_n, r_n).$$

The TT-Rank of a tensor is denoted by a vector of ranks $(R_1, \cdots, R_n)$ in the tensor train decomposition.

Left and right unfoldings reshape tensors into matrix, and are defined as follows.

Definition 2.2.1 *(Left and Right Unfolding) Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times R_n}$ be a $n+1$ mode tensor. The left unfolding operation is the matrix obtained by taking the first n mode as row indices and the last mode as column indices such that $\mathbf{L}(\mathcal{X}) \in \mathbb{R}^{(I_1 \cdots I_n) \times R_n}$. Similarly, the right unfolding operation produces the matrix obtained by taking the $1st$ mode as row indices and the remaining n mode as column indices such that $\mathbf{R}(\mathcal{X}) \in \mathbb{R}^{I_1 \times (I_2 \cdots I_n R_n)}$.*

We further introduce a tensor operation and show the equivalence of tensor operations to matrix product.

Definition 2.2.2 *(Tensor Merging Product) Tensor merging product is an operation to merge the two tensors along the given sets of mode indices. Let $\mathcal{U}_1 \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ and $\mathcal{U}_2 \in \mathbb{R}^{J_1 \times \cdots \times J_m}$ be two tensors. Let $\mathbf{g}_i$, $i \in \{1, 2\}$, be a $k-$dimensional vector such that $\mathbf{g}_i(p) \in \{1, \cdots, n\}, 1 \leq p \leq k$ and $I_{\mathbf{g}_1(p)} = J_{\mathbf{g}_2(p)}$. Then, the tensor merging product is*

$$\mathcal{U}_3 = \mathcal{U}_1 \times_{\mathbf{g}_1}^{\mathbf{g}_2} \mathcal{U}_2 \in \mathbb{R}^{\{\times_{p \notin \mathbf{g}_1} I_p\} \times \{\times_{p \notin \mathbf{g}_2} J_p\}}, \tag{2.2}$$

which is a $m+n-2k$ mode tensor, given as

$$\mathcal{U}_3(i_t \forall t \notin \mathbf{g}_1, j_q \forall q \notin \mathbf{g}_2) = \underset{d_1, \cdots, d_k}{} \mathcal{U}_1(a_1, \cdots, a_n) \mathcal{U}_2(b_1, \cdots, b_m), \tag{2.3}$$

where $a_r = i_r$ for $r \notin \mathbf{g}_1$, $b_r = j_r$ for $r \notin \mathbf{g}_2$, $a_{g_1(p)} = d_p$ for $p = 1, \cdots, k$, and $b_{g_2(p)} = d_p$ for $p = 1, \cdots, k$.

Based on tensor merging product, we note that recovering a tensor from tensor train decomposition is a process of applying tensor merging product on tensor train factorizations. Let $R_0 = 1$, $\mathcal{U}_i \in \mathbb{R}^{R_{i-1} \times I_i \times R_i}, i = 1, \cdots, n$, be n 3 order tensors. The recovery of the $n+1$ order tensor is equivalent to find a set of 3 order tensors such that

$$\mathcal{U} = \mathcal{U}_1 \times_3^1 \mathcal{U}_2 \times \cdots \times_3^1 \mathcal{U}_n \in \mathbb{R}^{I_1 \times \cdots \times I_n \times R_n}. \tag{2.4}$$

Tensor merging product also applies in the matrix product. For instance, the matrix product between $\mathbf{A} \in \mathbb{R}^{m\times r}$ and $\mathbf{B} \in \mathbb{R}^{r\times n}$ is equivalent to $\mathbf{A} \times_2^1 \mathbf{B}$. This is because if $\mathbf{C} = \mathbf{A} \times \mathbf{B}$, then $\mathbf{C}(i,j) = \quad {}_{k}\,\mathbf{A}(i,k)\mathbf{B}(k,j)$. Similarly, $\mathbf{A} \times \mathbf{B} = \mathbf{B} \times_1^2 \mathbf{A}$.

Lemma 1 *Let $\mathcal{A} \in \mathbb{R}^{M\times R_1\times\cdots\times R_k}$ and $\mathcal{B} \in \mathbb{R}^{R_1\times\cdots\times R_k\times N}$ be two $k+1$ mode tensors, and let $\mathbf{A} \in \mathbb{R}^{M\times(R_1\cdots R_k)}$ and $\mathbf{B} \in \mathbb{R}^{(R_1\cdots R_k)\times N}$ be the right and left unfolding of $\mathcal{A}$ and $\mathcal{B}$. Tensor merging product, $\mathcal{A} \times_{2,\cdots,k+1}^{1,\cdots,k} \mathcal{B}$, is the same as $\mathbf{A} \times \mathbf{B}$.*

Proof The $(m,n)^{\text{th}}$ entry in the result gives

$$(\mathcal{A} \times_{2,\cdots,n+1}^{1,\cdots,n} \mathcal{B})_{m,n} = \underset{r_1,\cdots,r_n}{\quad} \mathcal{A}(m,r_1,\cdots,r_n)\mathcal{B}(r_1,\cdots,r_n,n), \tag{2.5}$$

which is the same as the (m,n)th entry given by $\mathbf{A} \times \mathbf{B}$. ■

2.3 Tensor Train PCA (TT-PCA)

2.3.1 TT-PCA Algorithm

Definition 2.3.1 *(Tensor Train Subspace (TTS)) A tensor train subspace, $\mathcal{S}_{TT} \subseteq \mathbb{R}^{I_1\times I_2\times\cdots\times I_n}$, is defined as the span of a n-order tensor that is generated by the tensor merging product of a sequence of 3-order tensors. Specifically,*

$$\mathcal{S}_{TT} \triangleq \{\mathcal{U}_1 \times_3^1 \mathcal{U}_2 \times \cdots \times_3^1 \mathcal{U}_n \times_3^1 \mathbf{a} | \forall \mathbf{a} \in \mathbb{R}^{R_n}\}. \tag{2.6}$$

For comparison with vector subspace model, tensors can be vectorized into vectors and the tensor train subspace expressed under matrix form gives

$$\mathbf{S}_{\text{TT}} = \{\mathbf{L}(\mathcal{U}_1 \times_3^1 \mathcal{U}_2 \times \cdots \times_3^1 \mathcal{U}_n)\mathbf{a} | \forall \mathbf{a} \in \mathbb{R}^{R_n}\}. \tag{2.7}$$

We note that a tensor train subspace is determined by $\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n$, where $\mathcal{U}_i \in \mathbb{R}^{R_{i-1}\times I_i\times R_i}$, $R_0 = 1$. When $n = 1$, the proposed tensor train subspace reduces to the linear subspace model under matrix case.

Lemma 2 *(Subspace Property) $\mathbf{S}_{TT}$ is a R_n dimensional subspace of $\mathbb{R}^{I_1\cdots I_n}$ for a given set of decomposed tensors., $\{\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n\}$.*

We next briefly outline some useful properties of the TT decomposition that will be used in this paper.

Lemma 3 *(Left-Orthogonality Property [22, Theorem 3.1]) For any tensor* $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times R_n}$ *of TT-rank* $\mathbf{R} = [R_1, \cdots, R_{n-1}]$*, the TT decomposition can be chosen such that* $\mathbf{L}(\mathcal{U}_i)$ *is left-orthogonal for all* $i = 1, \cdots n$*, or* $\mathbf{L}(\mathcal{U}_i)\ \ \mathbf{L}(\mathcal{U}_i) = \mathbf{I}_{R_i} \in \mathbb{R}^{R_i \times R_i}$. As a consequence of this result we have the following Lemma.

Lemma 4 *(Left-Orthogonality of Tensor Merging Product) If* $\mathbf{L}(\mathcal{U}_i)$ *is left-orthogonal for all* $i = 1, \cdots, n$*, then* $\mathbf{L}(\mathcal{U}_1 \times_3^1 \cdots \times_3^1 \mathcal{U}_j)$ *is left-orthogonal for all* $1 \leq j \leq n$*.*

Proof Let $\mathbf{B}_j = \mathbf{L}(\mathcal{U}_1 \times_3^1 \cdots \times_3^1 \mathcal{U}_j)$. We first show $\mathbf{B}_{j+1} = (\mathbf{I}^{I_{j+1}} \otimes \mathbf{B}_j) \times \mathbf{L}(\mathcal{U}_{j+1})$. Using this, and induction (since the result holds for $j = 1$), the result follows. $\mathbf{B}_j$ is a matrix of shape $(I_1 I_2 \cdots I_j) \times R_j$. When $I_{j+1} = 1$, $\mathcal{U}_{j+1}$ is a 3_{rd} order tensor of shape $R_j \times 1 \times R_{j+1}$, which is equivalent to a matrix of shape $R_j \times R_{j+1}$, thus $\mathbf{B}_{j+1} = \mathbf{B}_j \times \mathcal{U}_{j+1}$ becomes standard matrix multiplication. When $I_{j+1} > 1$, the tensor merging product is equivalent to the concatenation of I_{j+1} matrix multiplications, which thus is $\mathbf{B}_{j+1} = (\mathbf{I}^{I_{j+1}} \otimes \mathbf{B}_j) \times \mathbf{L}(\mathcal{U}_{j+1})$. ∎

Thus, we can without loss of generality, assume that $\mathbf{L}(\mathcal{U}_i)$ are left-orthogonal for all i. Then, the projection of a data point $\mathbf{y} \in \mathbb{R}^{R_n}$ on the subspace $\mathbf{S}_{\mathrm{TT}}$ is given by $\mathbf{L}(\mathcal{U}_1 \times_3^1 \mathcal{U}_2 \times \cdots \times \mathcal{U}_n)\ \ \mathbf{y}$.

Tensor Train Principle Component Analysis (TT-PCA) Given a set of tensor data $\mathcal{X}_i \in \mathbb{R}^{I_1 \times \cdots \times I_n}$, $i = 1, \cdots, N$, we intend to find r_n principal vectors that convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. The r_n principal vectors can be stacked as a matrix $\mathbf{L}(\mathcal{U}_1 \mathcal{U}_2 \cdots \mathcal{U}_n) \in \mathbb{R}^{I_1 \cdots I_n \times r_n}$ such that $\mathcal{U}_i \in \mathbb{R}^{r_{i-1} \times I_i \times \times r_i}$, with $r_0 = 1$. The objective of Tensor Train PCA (TT-PCA) is to find such $\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n$ such that the distance of the points from the TTS formed by $\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n$ is minimized. We note that for $n = 1$, this is the same objective as that for standard PCA [44]. In Fig 2.1, we provide a TT-PCA example for a set of N tensor data of dimension $I_1 \times I_2 \times I_3$ and tensor

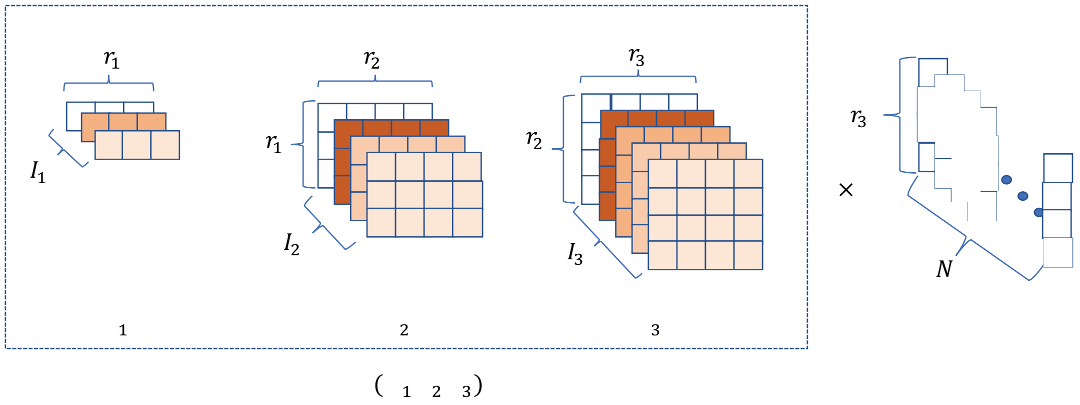


Fig. 2.1.: TT-PCA: Tensor Train subspace $\mathbf{L}(\mathcal{U}_1\mathcal{U}_2\mathcal{U}_3)$.

train rank $[r_1, r_2, r_3]$. The tensor train subspace is given by the left unfolding after the tensor connect product of $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$. Intuitively, tensor train subspace seeks a more efficient and compressible structure within the PCA subspace. Let $\mathbf{D} \in \mathbb{R}^{I_1 \cdots I_n \times N}$ be the matrix that concatenates the N vectorizations such that the i^{th} column of $\mathbf{D}$ is $\mathbf{V}(\mathcal{X}_i)$. The goal then is to find $\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n$ such that the distance of points from the subspace is minimized. More formally, we wish to solve the following problem,

$$\min_{\mathcal{U}_i, i=1,\cdots,n,\mathbf{A}} \quad \mathbf{L}(\mathcal{U}_1 \cdots \mathcal{U}_n)\mathbf{A} - \mathbf{D} \ {}_F^2. \tag{2.8}$$

This optimization problem in (2.8) is a non-convex problem. We however note that the problem is convex w.r.t. each of the variables $(\mathcal{U}_i, i = 1, \cdots, n, \mathbf{A})$ when the rest are fixed. Thus, one approach to solve the problem is to alternatively minimize over the variables when the rest are fixed. We propose an alternate approach that is based on successive SVD-algorithm for computing TT Decomposition in [22]. It is worthy to point that the proposed TT-PCA algorithm is a one-pass successive SVD-algorithm, rather than an iteration algorithm. The theoretical analysis in [22] shows that the successive SVD-algorithm is able to efficiently find the low rank tensor train decomposition that is close to the optimal. The algorithm steps are given in Algorithm 1. The algorithm steps assume that rank vector is not known, and estimates the ranks based on thresholding singular values. However, if the ranks are known, the threshold will be at the r_i number of singular values rather than at τ fraction of the maximum

Algorithm 1 Tensor Train Principle Component Analysis (TT-PCA) Algorithm

```
Input: $N$ tensors $\mathcal{X}_i \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n}$, $i = 1, \cdots, N$, threshold parameter $\tau$
Output: Decomposition for tensor train subspace $\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n$ and the representation $\mathbf{A}$
1: Form $\mathcal{Y}$ as an order $n+1$ tensor s.t. $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n \times N}$, which is formed by concatenating all data points $\mathcal{X}_i$ in the last mode.
2: Set $\mathbf{X}_1$ to be the $\mathcal{Y}_{[1]} \in \mathbb{R}^{I_1 \times (I_2 \cdots I_n N)}$ and apply SVD to $\mathbf{Y}_1$ such that $\mathbf{Y}_1 = \mathbf{U}_1 \mathbf{S}_1 \mathbf{V}_1$. Threshold singular values in $\mathbf{S}_1$ by maintaining the singular value larger than $\tau\sigma_{\max_1}$, where $\sigma_{\max_1}$ is the largest singular value of $\mathbf{S}_1$, to get $\tilde{\mathbf{S}}_1$ and the number of non-zero singular values in $\tilde{\mathbf{S}}_1$ as $r_1$, calculate $\mathbf{X}_2 = \tilde{\mathbf{S}}_1 \mathbf{V}$ and set $\mathcal{U}_1 = \mathbf{L}^{-1}(\mathbf{U}_1) \in \mathbb{R}^{1 \times I_1 \times r_1}$.
3: for $i = 2$ to $n$ do
4:     Reshape $\mathbf{X}_i \in \mathbb{R}^{r_{i-1} \times (I_i \cdots I_n N)}$ to $\mathbf{Y}_i \in \mathbb{R}^{(r_{i-1} I_i) \times (I_i \cdots I_n N)}$ and apply SVD to $\mathbf{Y}_i$ such that $\mathbf{Y}_i = \mathbf{U}_i \mathbf{S}_i \mathbf{V}_i$
5:     Threshold singular values in $\mathbf{S}_i$ by maintaining the singular value larger than $\tau\sigma_{\max_i}$ to get $\tilde{\mathbf{S}}_i$ and the number of non-zero singular values in $\tilde{\mathbf{S}}_i$ as $r_i$.
6:     Set $\mathcal{U}_i = \mathbf{L}^{-1}(\mathbf{U}_i) \in \mathbb{R}^{r_{i-1} \times I_i \times r_i}$ and $\mathbf{X}_{i+1} = \tilde{\mathbf{S}}_i \mathbf{V}_i$
7: end for
8: Set $\mathbf{A} = \mathbf{X}_{n+1}$
```

singular value. The proposed algorithm goes from left to right and find the different $\mathcal{U}_i$s. We note that this algorithm extends computing TT Decomposition in [22] by thresholding over the singular values, which tries to find the low rank approximation since the data is not exactly low rank. Such approaches for thresholding singular values for data approximation to low rank have been widely used for matrices [45,46].

The advantage of the approach include the following: (i) There are no iterations like in Alternating Minimization based approach, and the complexity is low. (ii) The obtained $\mathbf{L}(\mathcal{U}_i)$ is left-orthogonal for all $i = 1, \cdots, N$. Due to this property, we have by Lemma 4 that $\mathbf{L}(\mathcal{U}_1 \cdots \mathcal{U}_n)$ is left-orthogonal. Thus, the projection of a data point $\mathcal{D} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ onto the TT subspace formed is $(\mathbf{L}(\mathcal{U}_1 \cdots \mathcal{U}_n))^T \mathbf{V}(\mathcal{D})$.

2.3.2 Classification Using TT-PCA

In order to use TT-PCA for classification, we assume that we have N_{tr} data points $\mathcal{X}_i \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n}$, $i = 1, \cdots, N_{\text{tr}}$ for training, each having label $l_i \in \{1, \cdots, C\}$ that identify the association of the data points to the C classes, and let N_{te} data points as test data points that we wish to classify into the C classes. The first step is to perform TT-PCA for each of the C classes based on the data points that have that particular label among the N training data points. Let the corresponding $\mathcal{U}_{i:i=1,\cdots,n}$ for class j be denoted as $\mathcal{U}^{(j)}_{i:i=1,\cdots,n}$. Further, let $\mathbf{U}^{(j)} = \mathbf{L}(\mathcal{U}^{(j)}_1 \cdots \mathcal{U}^{(j)}_n)$. For a data point in the testing set $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots I_n}$, we wish to decide its label based on its distance to the subspace. Thus, the assigned label is given by

$$\text{Label}(\mathcal{Y}) = \underset{j=1,\cdots,C}{\arg\min} \;\; \mathbf{U}^{(j)}\mathbf{U}^{(j)} \;\; \mathbf{V}(\mathcal{Y}) - \mathbf{V}(\mathcal{Y}) \;\; {}^2_2. \tag{2.9}$$

2.3.3 Storage and Computation Complexity

In this subsection, we will give the amount of storage needed to store the subspace, and complexity for doing TT-PCA and classification based on TT-PCA. For comparisons, we consider the standard PCA, CANDECOMP/PARAFAC baed PCA

(CP-PCA) [47], and Tucker based PCA (T-PCA) algorithm [31]. We let $d = I_1 \cdots I_n$ be the dimension of each vectorized n^{th} order tensor data. Suppose we have N data points. We assume that $I_1 = \cdots = I_n$. Further, rank for PCA is chosen to be r, rank in each unfold for T-PCA is assumed to be r, and the ranks $r_i = r$ for $i \geq 1$ are chosen for TT-PCA. We note that ranks in each decomposition have a different interpretation and not directly comparable.

Storage of subspace: Under PCA model, the storage needed is for a $d \times r$ matrix which is left-orthogonal, and thus

$$\dim(\text{PCA}) = dr - r(1+r)/2, \tag{2.10}$$

where the $r(1+r)/2$ component is saved in storage as a result of orthonormal property of the PCA bases.

Under T-PCA model, n linear transformations and r core tensors need to be stored, and thus

$$\dim(\text{T-PCA}) = r^{n+1} + n \quad d^{\frac{1}{n}} r - r(1+r)/2 \quad , \tag{2.11}$$

where r^{n+1} is the storage for r cores, each $\in \mathbb{R}^{r \times \cdots \times r}$, and $n(d^{\frac{1}{n}} r - r(1+r)/2)$ is the storage for n linear transformations. $nr(1+r)/2$ amount of storage is saved due to the orthonormal property of the linear transformation matrices.

Under CP-PCA model, the reconstruction of the tensor data requires to store r vector sets, and each vector set is a best rank-1 CP approximation of the tensor data, thus the overall storage cost for the CP-PCA is

$$\dim(\text{CP-PCA}) = rnd^{\frac{1}{n}}. \tag{2.12}$$

It is worthy to hight that CP-PCA is actually a tensor reconstruction algorithm without specifying the tensor subspace. Thus although CP-PCA provides a way to reduce the dimensionality of tensor data via searching a CP structure, it can not provide a subspace like PCA subspace where data can be projected on.

Under TT-PCA model, we need to store $\mathcal{U}_1, \cdots, \mathcal{U}_n$ which are all left-orthogonal, and thus

$$\dim(\text{TT-PCA}) = d^{\frac{1}{n}} r(r(n-1)+1) - r(1+r)n/2, \tag{2.13}$$

where $\mathcal{U}_1$ takes $d^{\frac{1}{n}}r - r(1+r)/2$ and the remain $n-1$ MPS takes $(n-1)(d^{\frac{1}{n}}r^2 - r(1+r)/2)$.

We also consider a metric of normalized storage, *compression ratio*, which is the ratio of subspace storage to the entire Nd amount of data storage, or equivalently $\rho_{\text{ST}} = \frac{\dim(\text{ST})}{N_{\text{tr}}d}$, where ST can be any of PCA, T-PCA, or TT-PCA.

Computation Complexity of finding reduced subspace: We will now find the complexity of the three PCA algorithms (standard PCA, T-PCA, and TT-PCA). We assume that there are C classes, N_{tr} is the total number of training data points, and N_{te} be the total number of test data points. To compute standard PCA, we first compute the covariance matrix of the data, whose complexity is $O(d^2N_{\text{tr}})$. This is followed by eigenvalue decomposition of the covariance matrix, whose complexity is $O(d^3)$. Thus, the overall complexity is $O(d^2\max(N_{\text{tr}}, d))$. To compute the subspace corresponding to T-PCA, we first compute n orthonormal linear transformations using SVD, which takes $O(nd^{\frac{1}{n}}r^2)$ [21] time. This is followed by finding the subspace basis for the dimensional reduced tensor by PCA, which takes $O(r^{2n}\max(N_{\text{tr}}, r^n))$ time. Thus, the total computation complexity is $O(r^{2n}\max(N_{\text{tr}}, r^n) + nd^{\frac{1}{n}}r^2)$. The computation complexity for finding the tensor train subspace needs the recovery of the n components $(\mathcal{U}_1, \cdots, \mathcal{U}_n)$, which takes $O(nd^{\frac{1}{n}}r^3)$ time for calculation based on Algorithm 1.

Classification Complexity: Prediction under standard PCA model is equivalent to solving (2.9), whose computation complexity is $O(N_{\text{te}}Cdr)$. For T-PCA, we need additional step to make $\mathbf{U}$ for each class, which required an additional complexity of $O(dCr^2)$. Thus, the overall complexity for prediction based on T-PCA is $O(Cdr\max(N_{\text{te}}, r))$. TT-PCA needs the same steps as T-PCA where first a conversion to $\mathbf{U}$ is needed which has a complexity of $O(dCr^2)$ for each class giving an overall complexity of $O(Cdr\max(N_{\text{te}}, r))$.

These results for PCA, T-PCA and TT-PCA are summarized in Table 2.1, where the lowest complexity entries in each column are bold-faced. We can see that TT-PCA has advantages in both storage and subspace computation. Although TT-PCA

Table 2.1.: Storage and computation complexity for PCA algorithm. The bold entry in each column depicts the lowest order.

	Storage	Subspace Computation	Classification
PCA	$dr - \frac{r(1+r)}{2}$	$O(d^2 \max(N_{\text{tr}}, d))$	$\mathbf{O(CdrN_{te})}$
T-PCA	$r^{n+1} + n(d^{\frac{1}{n}} r - \frac{r(r+1)}{2})$	$O(r^{2n} \max(N_{\text{tr}} + r^n) + nd^{\frac{1}{n}} r^2)$	$O(Cdr \max(N_{\text{te}}, r))$
TT-PCA	$\mathbf{d^{\frac{1}{n}} r(r(n-1)+1) - \frac{r(r+1)n}{2}}$	$\mathbf{O(nd^{\frac{1}{n}} r^3)}$	$O(Cdr \max(N_{\text{te}}, r))$
CP-PCA	$rnd^{\frac{1}{n}}$	N/A	N/A

degrades in computation complexity compared with PCA in making prediction, the extra complexity is dependent on amount of testing data and is negligible for $N_{\text{te}} > r$.

2.3.4 Results

In this section, we compare the proposed TT-PCA algorithm with the T-PCA [48–50], CP-PCA [47], and the standard PCA algorithms. T-PCA is a Tucker decomposition based PCA that has been shown to be effective in face recognition, and CP-PCA is a CP decomposition based PCA that has been shown to be efficient in dimensionality reduction for tensor data.

We first compare the low rank reconstructed data using PCA, CP-PCA, T-PCA, and TT-PCA algorithms, and the algorithm performance is evaluated by the reconstruction error at different compression. Specifically, efficient PCA algorithms give low reconstruction error at the same compression ratio. We then compare the classification error among all algorithms, and the label is selected by the label of the subspace that gives the minimal residual errors, as depicted in (2.9). However, since CP-PCA [47] does not have a subspace structure, the classification is only compared among PCA, T-PCA, and TT-PCA.

The evaluation is conducted in the Extended YaleFace Dateset B [51, 52], which consists of 38 persons with 64 faces each that are taken under different illumination. Extended YaleFace Dateset B has been shown to satisfy subspace structure [53],

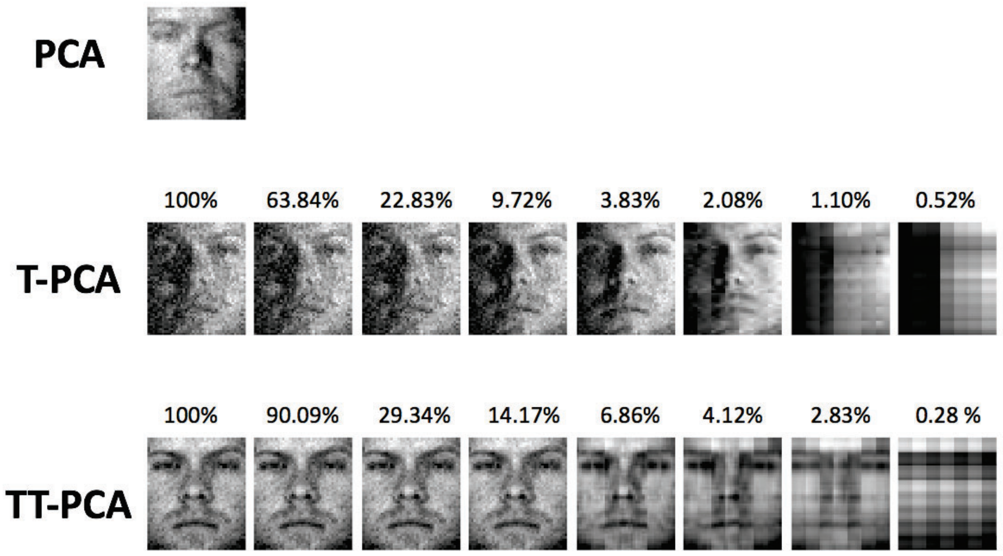


Fig. 2.2.: First Eigen Face for PCA and First Tensor Face for T-PCA and TT-PCA under different compression ratios. The number at top are the compression ratios.

which motivates our choice for exploring multi-dimensional subspace structures in this dataset. *For the experiments each image of a person is reshaped as* $\mathcal{X}_i \in \mathbb{R}^{6\times 8\times 6\times 7}$ *to validate the approach using tensor subspaces.* In addition, the TT-PCA algorithm is further evaluated in MNIST Dataset and Cifar10 Dataset.

Extended YaleFace Dataset B

We first compare the first dominant eigen-face for PCA and the first dominant tensor-face for T-PCA and TT-PCA by sampling $N_{\text{tr}} = 20$ images from one randomly selected person, and reshape each of the images into a 4^{th} order tensor for tensor PCA analysis. We add a Gaussian noise $\mathcal{N}(0, 900)$ to each pixel of the image. TT-PCA and T-PCA have the flexibility in controlling compression ratio by switching τ, where larger τ gives high compression ratio and less accuracy in approximation and vice versa. Figure 2.2 shows the tensor-face for T-PCA and TT-PCA under different compression ratio, and the eigen-face for PCA, where the compression ratio (marked on top) is decreasing from left-right (implying increasing data compression) for the tensor PCA algorithms. TT-PCA shows a better performance in constructing

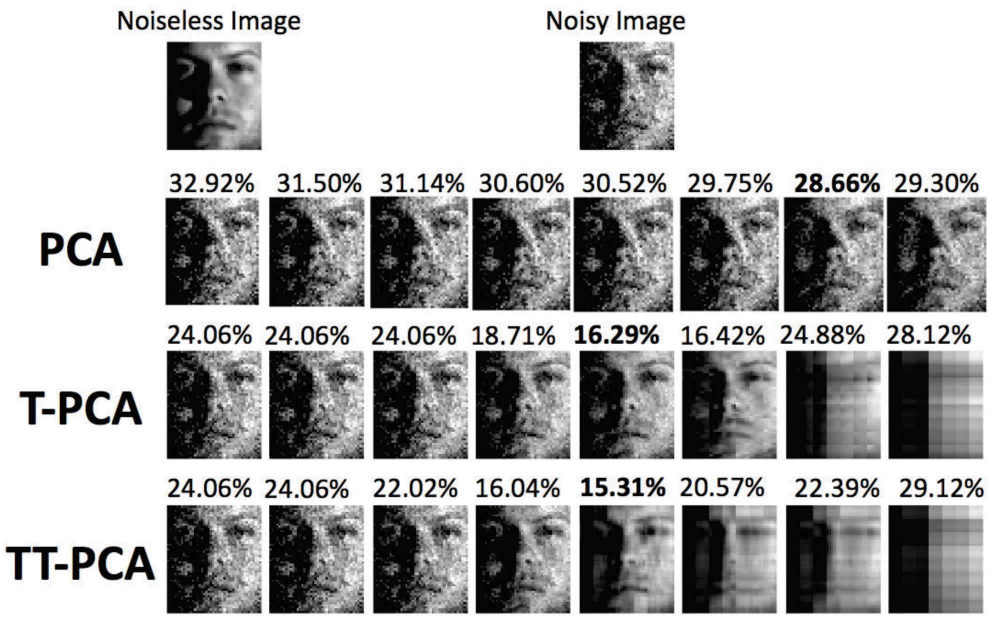


Fig. 2.3.: Face denoising under PCA, T-PCA and TT-PCA. The reconstruction errors are marked on top of each image. Different images in each row correspond to decreasing compression ratios (increasing compression, increasing τ) from left to right. The compression ratios for T-PCA and TT-PCA are the same (left-right) as that in Fig. 2.2.

tensor-face than both the T-PCA and PCA algorithms since the dominant eigen-face pictorially takes more features of the noiseless image of the person. As τ increases (which changes compression, from left to right), tensor rank becomes lower and tensor-face degrades to more blurry images. Under similar compression ratio, such as 6.86% for TT-PCA and 9.72% for T-PCA, TT-PCA performs better than T-PCA since the tensor face is less affected by noise.

We further illustrate one image sampled from the 20 noisy images and its projection onto (a) the linear subspace given by PCA with ranks being $16, 14, 12, 10, 8, 6, 4, 2$ (from left-to-right), which gives compression ratios of $0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1$, (b) the multi-linear subspace given by T-PCA with compression ratio $1, 0.638.0.228$, $0.972, 0.038, 0.021, 0.011, 0.005$, and (c) tensor train subspace given by TT-PCA with compression ratio $1, 0.901, 0.293, 0.142, 0.069, 0.041, 0.028, 0.003$. The reconstruction

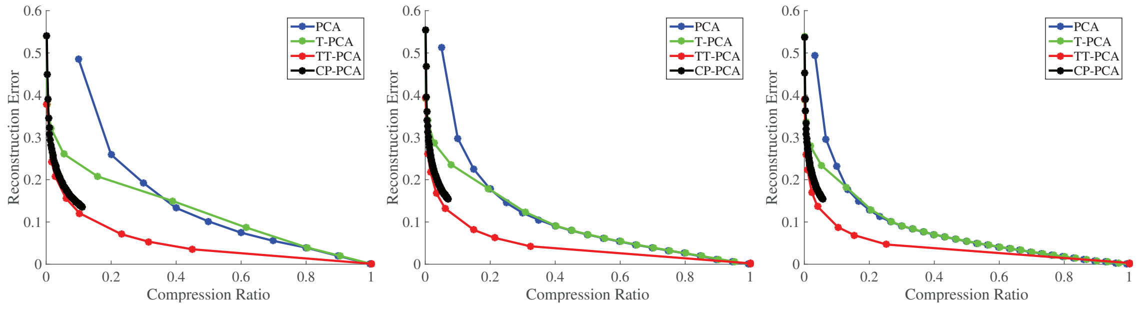


Fig. 2.4.: Reconstruction error versus Compressed Ratio for Extended YaleFace Dataset B Dataset. 38 faces with noise are selected from the data set and the training sample size f is 10, 20, 30 (from left to right) respectively.

error, defined as the distance between the original image (without noise) and the projection of the noisy image to the subspace, is depicted at the top of images in Figure 2.3. As seen from the figure the reconstruction errors of T-PCA and TT-PCA are significantly lower than that of PCA, and TT-PCA gives the lowest 15.31% reconstruction error under 0.069 compression ratio.

Reconstruction using TT-PCA-Next, we evaluate the reconstruction error of PCA, T-PCA, CP-PCA and TT-PCA. Since PCA is a dimensionality reduction algorithm that minimizes the reconstruction error under limited parameter budget, we test the reconstruction error, defined as $\|\mathcal{X}_{\text{reconstruct}} - \mathcal{X}\ _F/\|\mathcal{X}\ _F$, versus the compression ratio for the four algorithms in Fig 2.4. We also note that unlike the other PCA algorithm where the maximal rank is known, it is unclear the maximal rank for CP-PCA algorithm. Meanwhile, CP-PCA is a sequential PCA algorithm where a rank r approximation of a tensor depends on the rank $r-1$ approximation. Considering the time budget for the computation, we chose the maximum rank of CP-PCA to be 100 that leads the maximum compression ratio for CP-PCA to be 0.1. The numerical results on YaleFace Dataset demonstrate that disregard the training sample size, TT-PCA gives lower reconstruction error than PCA, T-PCA, and CP-PCA under any compression ratios.

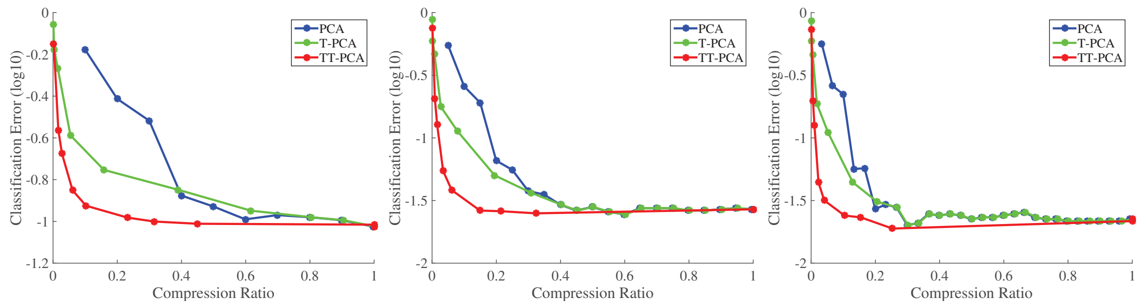


Fig. 2.5.: Classification Error in $\log 10$ scale versus Compressed Ratio for Extended Yale-Face Dataset B Dataset. 38 faces with noise are selected from the data set and the training sample size f is 10, 20, 30 (from left to right) respectively.

Classification using TT-PCA - Next, we test the performance of PCA, T-PCA, and TT-PCA for classification. For classification, we choose f training data points (at random) from each of the 38 people, and thus the amount of training data points is $N_{\text{tr}} = 38f$. The remaining data of each person is used for testing, and thus $N_{\text{te}} = 38(64 - f)$. For training sizes $f = 10, 20, 30$, Figure 2.5 compares the classification error of the different algorithms as a function of compression ratio for each f. We note that as f increases, the classification performance becomes better for all algorithms. We further see that TT-PCA performs better at low compression ratios, and the classification error increases after first decreasing. This is because with higher compression ratios (low compression), the approaches will try to over-fit noise leading to lower classification accuracy. *This indicates that human face data under different illumination conditions lies not only close to the subspace models, but are better approximated by tensor train subspace models. Further we note that TT-PCA requires far less training sample size compared to other approaches.*

MNIST Dataset MNIST dataset includes 60,000 hand written digits from 0 to 9, and each image is of size 28×28. To evaluate the TT-PCA algorithms, we randomly select 1000 images for each digit, which are further split to 100 for training and 900 for testing. Reconstruction error and classification error versus compression ratio is shown in Fig 2.6. TT-PCA outperforms than PCA, T-PCA, and CP-PCA

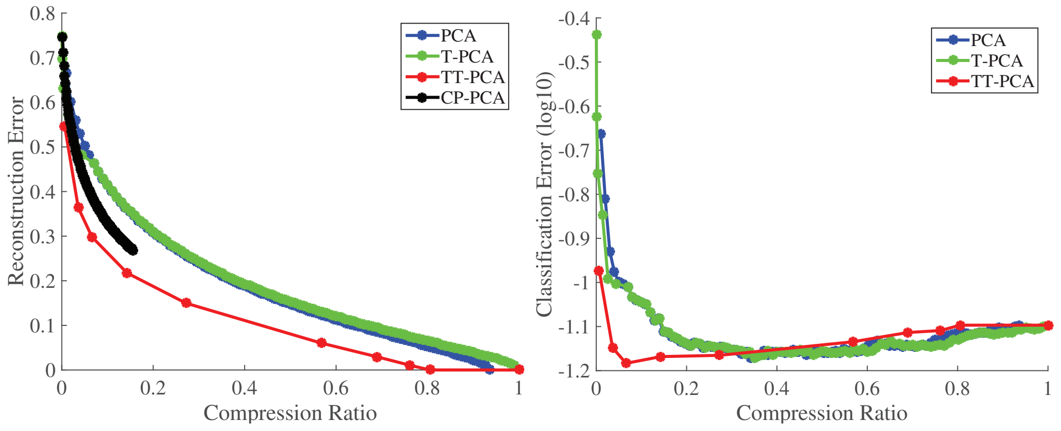


Fig. 2.6.: Reconstruction Error versus Compression Ratio (left) and Classification Error in log 10 scale versus Compression Ratio (right) for MNIST dataset.

by a large margin for the reconstruction error, and at the compression ratio 0.8, TT-PCA gives very small amount of reconstruction error, demonstrating the efficiency in capturing the low-dimensional structure in the image data. Even though TT-PCA performs slightly worse than PCA and T-PCA in classification when the compression ratio is larger than 0.4, TT-PCA still shows improved classification performance at compression ratio 0.1 that is attributes to the efficient extraction of the discriminative features from the images. It is worthy to note that better reconstruction does not indicate better classification, and the better classification requires better feature extraction.

CIFAR-10 Dataset CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes. It is selected because CIFAR-10 is a colorful data with more noise, and is harder to reconstructed and classified. The TT-PCA algorithms are evaluated by randomly sampling 1000 images from each class, where 100 images are used for training and the remaining 900 images are used for testing. Each colorful image of size $32 \times 32 \times 3$ is heuristically reshaped into $4 \times 8 \times 4 \times 4 \times 3$ for tensor PCA analysis. Similar to the performance of TT-PCA for MNIST dataset, TT-PCA outperforms than the others in image reconstruction and classification by a larger margin, which in part is due to improved capability in de-noising of noisy data. However, the overall

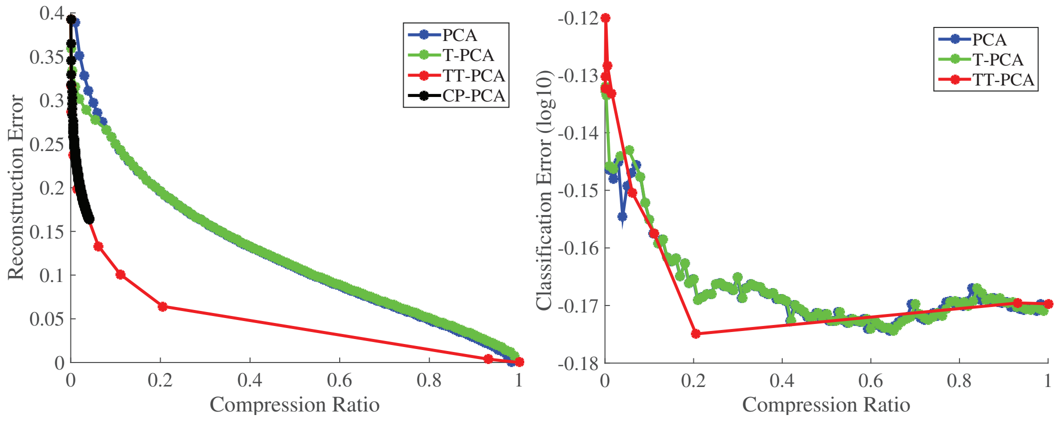


Fig. 2.7.: Reconstruction Error versus Compression Ratio (left) and Classification Error in log 10 scale versus Compression Ratio (right) for Cifar10 dataset.

classification performance for PCA, T-PCA, and TT-PCA down-grades significantly as compared to the classification results for MNIST, demonstrating the difficulty in classifying noisy dataset. Nevertheless, TT-PCA still gives the best classification results at the compression ratio 0.2.

2.4 Tensor Train Neighbor Preserving Embedding (TT-NPE)

Unlike TT-PCA that maintains the global information of the data distribution and reduces the dimensionality of tensor data by projecting data onto a tensor train subspace to minimize the reconstruction error, an tensor train neighbor preserving embedding algorithm seeks a projection with a tensor train structure that maintains the local information (in particular neighborhood information). Mathematically, given a set of tensor data $\mathcal{X}_i \in \mathbb{R}^{I_1\times\cdots\times I_n}$, $i = 1, \cdots, N$, we wish to project the data $\mathcal{X}_i$ to vector $\mathbf{t}_i \in \mathbb{R}^{R_n}$, satisfying $\mathbf{t}_i = \mathbf{L}(\mathcal{U}_1 \times_3^1 \mathcal{U}_2 \times \cdots \times \mathcal{U}_n)^T \mathbf{V}(\mathcal{X}_i)$ and preserving neighborhood among the projected data. We first construct a neighborhood graph to capture the neighborhood information in the given data and generate the affinity matrix $\mathbf{F}$ as

$$\mathbf{F}_{ij} = \begin{cases} \exp(-\|\mathcal{X}_i - \mathcal{X}_j \ {}_F^2/\epsilon), & \text{if } \mathcal{X}_{j:j=i} \in O(K, \mathcal{X}_i) \\ 0, & \text{otherwise,} \end{cases} \tag{2.14}$$

where $O(K, \mathcal{X}_i)$ denotes the subset of data excluding $\mathcal{X}_i$ that are within the K-nearest neighbors of $\mathcal{X}_i$, and   is the scaling factor. By definition, $\mathbf{F}_{ii} = 0$. We also note that this is an unsupervised tensor embedding method since the label information is not used in the embedding procedure. Without loss of generality, we set $\mathbf{S} = \mathbf{F} + \mathbf{F}$ and $\mathbf{S}$ is further normalized by dividing entries of each row by the row sum such that each row sums to one.

The goal is to find the decomposition $\mathcal{U}_1, \cdots, \mathcal{U}_n$ that minimizes the average distance between all the points and their weighted combination of remaining points, weighted by the symmetrized affinity matrix in the projection, i.e.

$$\min_{\substack{\mathcal{U}_k:\forall k=1,\cdots,n \\ \mathbf{L}(\mathcal{U}_k) \text{ is Unitary}}} \ {}_i \quad \mathbf{L}\ (\mathcal{U}_1 \times \cdots \times \mathcal{U}_n)\mathbf{V}(\mathcal{X}_i) - \ {}_j \ \mathbf{S}_{ij}\mathbf{L}\ (\mathcal{U}_1 \times \cdots \times \mathcal{U}_n)\mathbf{V}(\mathcal{X}_j)\ {}_2^2. \tag{2.15}$$

Let $\mathbf{D} \in \mathbb{R}^{I_1 \cdots I_n \times N}$ be the matrix that concatenates the N vectorized tensor data such that the i^{th} column of $\mathbf{D}$ is $\mathbf{V}(\mathcal{X}_i)$, and let $\mathbf{E} = \mathbf{L}(\mathcal{U}_1 \times \cdots \times \mathcal{U}_n)$. Then, (2.15) is equivalent to

$$\min_{\substack{\mathcal{U}_k:\forall k=1,\cdots,n \\ \mathbf{L}(\mathcal{U}_k) \text{ is Unitary}}} \quad \mathbf{E}\ (\mathbf{D} - \mathbf{DS}\ )\ {}_F^2. \tag{2.16}$$

Since $\mathbf{D} - \mathbf{DS} \quad \in \mathbb{R}^{(I_1 \cdots I_n) \times N}$ is determined, we set $\mathbf{Y} = \mathbf{D} - \mathbf{DS}$. Thus the Frobenius norm in (2.16) can be further expressed in the form of matrix trace to reduce the problem to

$$\min_{\substack{\mathcal{U}_k:\forall k=1,\cdots,n \\ \mathbf{L}(\mathcal{U}_k) \text{ is Unitary}}} \operatorname{tr}(\mathbf{Y}\ \ \mathbf{EE}\ \ \mathbf{Y}). \tag{2.17}$$

Based on the cyclic permutation property of the trace operator, (2.17) is equivalent to

$$\min_{\substack{\mathcal{U}_k:\forall k=1,\cdots,n \\ \mathbf{L}(\mathcal{U}_k) \text{ is Unitary}}} \operatorname{tr}(\mathbf{E}\ \ \mathbf{YY}\ \ \mathbf{E}). \tag{2.18}$$

Let $\mathbf{Z} = \mathbf{Y}\mathbf{Y} \ \ \in \mathbb{R}^{(I_1 \cdots I_n) \times (I_1 \cdots I_n)}$ be the constant matrix. Then, the problem (2.18) becomes

$$\min_{\substack{\mathcal{U}_k : \forall k = 1, \cdots, n \\ \mathbf{L}(\mathcal{U}_k) \text{ is Unitary}}} \operatorname{tr}(\mathbf{E} \ \ \mathbf{Z}\mathbf{E}). \tag{2.19}$$

We will use the alternating minimization method [54] to solve (2.19) such that each $\mathcal{U}_k$ is updated by solving

$$\min_{\mathcal{U}_k : \mathbf{L}(\mathcal{U}_k) \text{ is unitary}} \operatorname{tr}(\mathbf{E} \ \ \mathbf{Z}\mathbf{E}). \tag{2.20}$$

In order to solve (2.20), we use an iterative algorithm. Each $\mathcal{U}_{k:k=1,\cdots,n}$ is initialized by tensor train decomposition [21] with a thresholding parameter τ, which zeros out the singular values which are smaller than τ times the maximum singular value, such that tensor train ranks $(R_1, \cdots, R_n)$ are determined. The larger the thresholding parameter τ, the smaller the tensor train ranks. Typically, τ could be chosen via cross validation such that the classification error in the validation set is minimized.

2.4.1 Tensor Train Neighbor Preserving Embedding using Tensor Network (TTNPE-TN)

Let $\mathcal{Z} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times I_1 \times \cdots \times I_n}$ be the reshaped tensor of $\mathbf{Z}$, and

$$\begin{aligned} \mathcal{T}_1 &= \mathcal{U}_1 \times \cdots \times \mathcal{U}_{k-1} \in \mathbb{R}^{I_1 \times \cdots \times I_{k-1} \times R_{k-1}}, \\ \mathcal{T}_n &= \mathcal{U}_{k+1} \times \cdots \times \mathcal{U}_n \in \mathbb{R}^{R_k \times I_{k+1} \times \cdots \times I_n \times R_n}. \end{aligned} \tag{2.21}$$

For **updating** $\mathcal{U}_{k:k=1,\cdots,n-1}$, based on Lemma 1, we note that (2.20) can be written as

$$\min_{\substack{\mathcal{U}_k \\ \mathbf{L}(\mathcal{U}_k) \text{ is unitary}}} \quad \mathcal{U}_k \quad \times_{1,2,3}^{1,2,3} \operatorname{tr}_4^8 \ \ \mathcal{Z} \times_{n+k+1,\cdots,2n}^{2,\cdots,n-k+1} \mathcal{T}_n \times_{n+1,\cdots,n+k-1}^{1,\cdots,k-1} \mathcal{T}_1 \times_{k+1,\cdots,n}^{2,\cdots,n-k+1} \mathcal{T}_n \times_{1,\cdots,k-1}^{1,\cdots,k-1} \mathcal{T}_1 \\ \times_{1,2,3}^{1,2,3} \mathcal{U}_k. \tag{2.22}$$

Let $\mathcal{A} \in \mathbb{R}^{R_{k-1} \times I_k \times R_k \times R_{k-1} \times I_k \times R_k}$ be the 6-order tensor, given as $\operatorname{tr}_4^8 \ \ \mathcal{Z} \times_{n+k+1,\cdots,2n}^{2,\cdots,n-k+1} \mathcal{T}_n \times_{n+1,\cdots,n+k-1}^{1,\cdots,k-1} \mathcal{T}_1 \times_{k+1,\cdots,n}^{2,\cdots,n-k+1} \mathcal{T}_n \times_{1,\cdots,k-1}^{1,\cdots,k-1} \mathcal{T}_1$, where the details to compute $\mathcal{A}$ via tensor merging product is given in Appendix A.

Thus (2.22) becomes

$$\min_{\mathcal{U}_k:\mathbf{L}(\mathcal{U}_k)\text{ is unitary}} \mathcal{U}_k \times^{1,2,3}_{1,2,3} \mathcal{A} \times^{1,2,3}_{1,2,3} \mathcal{U}_k. \tag{2.23}$$

Based on Lemma 1, the tensor merging product (2.23) can be transformed into matrix product. Thus, (2.23) becomes

$$\min_{\mathcal{U}_k:\mathbf{L}(\mathcal{U}_k)\text{ is unitary}} \mathbf{V}(\mathcal{U}_k)\ \ \mathbf{A}\mathbf{V}(\mathcal{U}_k), \tag{2.24}$$

where $\mathbf{A} \in \mathbb{R}^{(R_{k-1}I_kR_k)\times(R_{k-1}I_kR_k)}$ is the reshaped form of $\mathcal{A}$. A differentiable function under unitary constraint can be solved by the algorithm proposed in [55]. In problem (2.24), the gradient of objective function to $\mathbf{V}(\mathcal{U}_k)$ is $2\mathbf{A}\mathbf{V}(\mathcal{U}_k)$.

Updating $\mathcal{U}_n$ is different from solving $\mathcal{U}_{k:k=1,\cdots,n-1}$ since the trace operation merges the tensor $\mathcal{U}_n$ with itself, thus (2.23) does not apply for solving $\mathcal{U}_n$. Instead, updating $\mathcal{U}_n$ in (2.20) is equivalent to solving

$$\min_{\mathcal{U}_n:\mathbf{L}(\mathcal{U}_n)\text{ is unitary}} \operatorname{tr}_1^2\ \ \mathcal{U}_n \times^{1,2}_{1,2} (\mathcal{Z} \times^{1,\cdots,n-1}_{n+1,\cdots,2n-1} \mathcal{T}_1 \times^{1,\cdots,n-1}_{1,\cdots,n-1} \mathcal{T}_1) \times^{1,2}_{1,2} \mathcal{U}_n\ \ .$$

Let $\mathcal{B} \in \mathbb{R}^{R_{n-1}\times I_n \times R_{n-1}\times I_n}$ be the 4-th order tensor formed by $(\mathcal{Z} \times^{1,\cdots,n-1}_{n+1,\cdots,2n-1} \mathcal{T}_1 \times^{1,\cdots,n-1}_{1,\cdots,n-1} \mathcal{T}_1)$, where the details to compute $\mathcal{B}$ via tensor merging product is given in Appendix B. Thus updating $\mathcal{U}_n$ is equivalent to solving

$$\min_{\mathcal{U}_n:\mathbf{L}(\mathcal{U}_n)\text{ is unitary}} \operatorname{tr}_1^2\ \ \mathcal{U}_n \times^{1,2}_{1,2} \mathcal{B} \times^{1,2}_{2,3} \mathcal{U}_n\ \ , \tag{2.25}$$

which by Lemma 1, can be transformed into the matrix form

$$\min_{\substack{\mathcal{U}_n \in \mathbb{R}^{R_{n-1}\times I_n \times R_n} \\ \mathbf{L}(\mathcal{U}_n)\text{ is unitary}}} \operatorname{trace}(\mathbf{L}\ \ \mathcal{U}_n)\ \ \mathbf{B}\mathbf{L}(\mathcal{U}_n)\ \ , \tag{2.26}$$

where $\mathbf{B} \in \mathbb{R}^{(R_{n-1}I_n)\times(R_{n-1}I_n)}$ is reshaped from $\mathcal{B}$. The gradient of the objective function to $\mathbf{L}(\mathcal{U}_n)$ is $2\mathbf{B}\mathbf{L}(\mathcal{U}_n)$.

We now analyze the **computation and memory complexity** of TTNPE-TN algorithm, where the memory complexity indicates the memory required to store all the intermediate variables. For $\mathcal{U}_{k:k=1,\cdots n-1}$, the generation of $\mathbf{A}$ requires merging the tensor networks, which has a computation complexity of

$$O \quad (I_1 \cdots I_n)^2 R_{k-1} + (I_1 \cdots I_n)^2 (\frac{R_{k-1}}{I_1 \cdots I_{k-1}})^2 R_k R_n \quad ,$$

and solving (2.24) takes $O\left(R_{k-1} I_k R_k^2\right)$ time. Thus, the computation of $\mathbf{A}$ dominates the complexity. The memory requirement for generating $\mathbf{A}$ is $O\left((R_{k-1} I_k R_k R_n)^2\right)$, which is large when the tensor train ranks are high. Similarly, the generation of $\mathbf{B}$ to solve $\mathcal{U}_n$ takes $O\left((I_1 \cdots I_n)^2 R_{n-1}\right)$ time and solving (2.26) takes $O(R_{n-1} I_n R_n^2)$, and the memory for generating $\mathbf{B}$ is $O\left((I_1 \cdots I_n)^2\right)$, indicating solving for $\mathcal{U}_n$ is less expensive than that for solving for $\mathcal{U}_k$ in terms of both memory and computation complexity.

Although TTNPE-TN algorithm gives an exact solution for updating $\mathcal{U}_i$ in each alternating minimization step, the memory and computation cost prohibits its application when the tensor train ranks are large. In order to address this, we propose a Tensor Train Neighbor Preserving Embedding using Approximate Tensor Network (TTNPE-ATN) algorithm in the next section, to approximate (2.20), aiming to reduce computation and memory cost.

2.4.2 Tensor Train Neighbor Preserving Embedding using Approximated Tensor Network (TTNPE-ATN)

Our main intuition is as follows. Without the TT decomposition constraint, the solution to minimize the quadratic form tr($\mathbf{E}$ $\mathbf{ZE}$) where $\mathbf{E}$ is unitary is given by $\mathbf{E}$ being the matrix formed by eigenvectors corresponding to the lowest eigenvalues of $\mathbf{Z}$ and the value of the objective is the sum of the lowest eigenvalues of $\mathbf{Z}$ [56]. Let the matrix corresponding to the eigenvectors corresponding to r_n smallest eigenvalues of $\mathbf{Z}$ be $\mathbf{V}_{r_n}$. With the additional constraint that $\mathbf{E}$ has TT decomposition, the above choice of $\mathbf{E}$ may not be optimal. Thus, we relax the original problem to minimize the distance between $\mathbf{E}$ and $\mathbf{V}_{r_n}$. Thus, the relaxed problem of (2.20) is

$$\min_{\substack{\mathcal{U}_k \\ \mathbf{L}(\mathcal{U}_k) \text{ is unitary}}} \quad \mathbf{L}(\mathcal{U}_1 \times \cdots \times \mathcal{U}_n) - \mathbf{V}_{r_n} \ {}_F^2, \tag{2.27}$$

where $\mathbf{L}(\mathcal{U}_1 \times \cdots \times \mathcal{U}_n), \mathbf{V}_{r_n} \in \mathbb{R}^{(I_1 \cdots I_n) \times r_n}$.

Let $\mathbf{T}_k$ be a reshaping operator that change the dimension of a matrix from $\mathbb{R}^{(I_1 \cdots I_n) \times r_n}$ to $\mathbb{R}^{(I_1 \cdots I_k) \times (I_{k+1} \cdots I_n r_n)}$, thus (2.27) is equivalent to

$$\min_{\mathcal{U}_k : \mathbf{L}(\mathcal{U}_k) \text{ is unitary}} \quad \mathbf{T}_k(\mathbf{L}(\mathcal{T}_1 \times_k^1 \mathcal{U}_k \times_3^1 \mathcal{T}_n)) - \mathbf{T}_k(\mathbf{V}_{r_n}) \ {}_F^2, \tag{2.28}$$

which is equivalent to

$$\min_{\mathcal{U}_k : \mathbf{L}(\mathcal{U}_k) \text{ is unitary}} \quad (\mathbf{I}_{I_k} \otimes \mathbf{L}(\mathcal{T}_1))\, \mathbf{L}(\mathcal{U}_k) \mathbf{R}(\mathcal{T}_n) - \mathbf{T}_k(\mathbf{V}_{r_n}) \ {}_F^2, \tag{2.29}$$

which has the same format as minimizing $\ \mathbf{PXQ} - \mathbf{C} \ {}_F^2$ under unitary constraint. Since the gradient is $\mathbf{P} \ (\mathbf{PXQ} - \mathbf{C})\mathbf{Q} \ $, (2.29) can be solved by the algorithm proposed in [55].

After the relaxation, the computation complexity is $O\left(R_{k-1} I_k I_1 \cdots I_n R_n\right)$ for calculating the gradient, $O\left((I_1 \cdots I_n)^2\right)$ for generating $\mathbf{V}_{r_n}$, and $O(R_{k-1} I_k R_k^2)$ for solving (2.29). Thus the eigenvalue decomposition for generating $\mathbf{V}_{r_n}$ dominates the computational complexity. The memory for computing $\mathbf{P}$ and $\mathbf{Q}$ is

$$\max(I_1 \cdots I_{k-1} R_{k-1}, R_k I_{k+1} \cdots I_n R_n).$$

Thus both memory and computation cost of TTNPE-ATN are much less than those in TTNPE-TN algorithm. Therefore in the simulation section, we will only consider the TTNPE-ATN algorithm. We validated for a small experiment that the embedding performance for the two are similar, where the validation results are omitted in this paper. The two algorithms (TTNPE-TN and TTNPE-ATN) are described in Algorithm 2.

Algorithm 2 TTNPE-TN and TTNPE-ATN Algorithms

```
Input: A set of $N$ tensors $\mathcal{X}_{i=1,\cdots,N} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n}$, denoted as $\mathcal{X}$, threshold parameter
    $\tau$, kernel scaling parameter , number of neighbors $K$, thresholding parameter
    $\tau$, and max iterations $maxIter$
Output: Tensor train subspace factors $\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n$
1: Compute affinity matrix $\mathbf{F}$ by
       $\mathbf{F}_{ij} = \begin{cases} \exp(-\|\mathcal{X}_i - \mathcal{X}_j \;^2_F/\epsilon), & \text{if } \mathcal{X}_{j:j=i} \in O(K, \mathcal{X}_i), \\ 0, & \text{otherwise,} \end{cases}$
       $\mathbf{S} = \mathbf{F} + \mathbf{F}$ and normalize $\mathbf{S}$ such that each row sums to 1.
2: Form $\mathbf{D} \in \mathbb{R}^{I_1 \cdots I_n \times N}$ as a reshape of the input data, compute $\mathbf{Y} = \mathbf{D} - \mathbf{DS}$ , and
   compute $\mathbf{Z} = \mathbf{YY}$ .
3: Apply tensor train decomposition [21] on $\mathcal{X}$ to initialize $\mathcal{U}_{i=1,\cdots,n}$ with thresholding
   parameter $\tau$, and the tensor train ranks are determined based on selection of $\tau$.
4: Solve $\mathbf{V}_{R_n}$ by applying eigenvalue decomposition on $\mathbf{Z}$.
5: Set $iter = 1$
6: while $iter \leq maxIter$ or convergence of $\mathcal{U}_1, \cdots, \mathcal{U}_n$ do
7:     for $i = 1$ to $n$ do
8:         (TTNPE-TN) Update $\mathcal{U}_i$ in (2.24) for $i < n$ and in (2.26) for $i = n$,
   using the algorithm proposed in [55].
9:         (TTNPE-ATN) Update $\mathcal{U}_i$ in (2.29) by algorithm proposed in [55].
10:    end for
11:    $iter = iter + 1$
12: end while
```

Table 2.2.: Storage and Computation Complexity Analysis for Embedding Methods. The bold entry in each column depicts the lowest order.

	Storage	Subspace Computation	Classification
KNN	dN_{tr}	$\mathbf{0}$	$O(N_{te}N_{tr}d)$
TNPE	$r^n N_{tr} + nd^{\frac{1}{n}}r.$	$O(n(N_{tr}rd + N_{tr}r^{2n} + r^{3n}))$	$O(N_{te}r^2d + N_{te}N_{tr}r^n)$
TTNPE-ATN	$\boldsymbol{(n-1)(d^{\frac{1}{n}}r^2 - r^2) + (d^{\frac{1}{n}}r - r^2) + rN_{tr}}$	$O(nd^{\frac{1}{n}}r^3 + dN_{tr}^2 + d^2N_{tr})$	$\boldsymbol{O(N_{te}r^2d + N_{te}N_{tr}r)}$

2.4.3 Classification Using TTNPE-TN and TTNPE-ATN

The classification is conducted by first solving a set of tensor train factors $\mathcal{U}_1, \cdots, \mathcal{U}_n$. Then, the training data and testing data is projected onto the tensor train subspace bases as follows:

$$\mathbf{t}_i = \mathbf{L}(\mathcal{U}_1 \times \cdots \times \mathcal{U}_n) \;\; \mathbf{V}(\mathcal{X}_i) \in \mathbb{R}^{R_n}. \tag{2.30}$$

Any data point in the testing set is labeled by applying k-nearest neighbors(KNN) [57] classification with K neighbors in the embedded space $\mathbb{R}^{R_n}$.

2.4.4 Storage and Computation Complexity

In this section, we will analyze the amount of storage to store the high dimensional data, complexity for finding the embedding using TTNPE-ATN and the cost of projection onto the TT subspace for classification. KNN and TNPE [41] algorithms are considered for comparison. For the computational complexity analysis, let d be the data dimension, n and r be the reshaped tensor order and rank in TTNPE-ATN model, K be the number of neighbors, and N_{tr} (N_{te}) be the total training (testing) data. We assume the dimension along each tensor mode is the same, thus each tensor mode is $d^{\frac{1}{n}}$ in dimension.

Storage of data Under KNN model, the storage required for N_{tr} training data is Storage(KNN) $= dN_{\text{tr}}$. Under TNPE model, the storage for the N_{tr} training data needs the space for n linear transformation which is $n(d^{\frac{1}{n}}r)$, and the space for N_{tr} embedded training data of size $N_{\text{tr}}r^n$, requiring the total storage Storage(TNPE) $=$

$r^n N_{\text{tr}} + nd^{\frac{1}{n}} r$. Under TTNPE-ATN model, we need space $(n-1)(d^{\frac{1}{n}} r^2 - r^2) + (d^{\frac{1}{n}} r - r^2)$ [22] to store the projection bases $\mathcal{U}_1, \cdots, \mathcal{U}_n$, and $N_{\text{tr}} r$ to store the embedded training data. Thus the total storage is $\text{Storage(TTNPE-ATN)} = (n-1)(d^{\frac{1}{n}} r^2 - r^2) + (d^{\frac{1}{n}} r - r^2) + rN_{\text{tr}}$. We consider a metric of normalized storage, *compression ratio*, which is the ratio of storage required under the embedding method and storage for the entire data, calculated by $\rho_{\text{ST}} = \frac{\text{Storage(ST)}}{N_{\text{tr}} d}$, where ST can be any of KNN, TNPE, TTNPE-ATN.

Computation Complexity for estimating the embedding subspace The computation complexity includes computation for both the addition and multiplication operations. Under KNN model, data is directly used for classification and there is no embedding process. Under TNPE model, the embedding needs 3 steps, where solving n linear transformations takes $O(N_{\text{tr}} rd)$ for embedding raw data, matrix generation for an eigenvalue problem takes $O(N_{\text{tr}} r^{2n})$, and eigenvalue decomposition for updating each linear transformation takes r^{3n}, giving a total computational complexity $O(n(N_{\text{tr}} rd + N_{\text{tr}} r^{2n} + r^{3n}))$. Under TTNPE-ATN model, the embedding takes 3 steps, where the initialization by tensor train decomposition algorithm takes $O(nd^{\frac{1}{n}} r^3)$, the generation of $\mathbf{Z}$ takes $O(dN_{\text{tr}}^2 + d^2 N_{\text{tr}})$, and updating $\mathcal{U}_k$, which includes a gradient calculation by merging a tensor network, takes $O(nd^{\frac{1}{n}} r^3))$, thus giving a total computational complexity $O(nd^{\frac{1}{n}} r^3 + dN_{\text{tr}}^2 + d^2 N_{\text{tr}})$.

Classification Complexity Under KNN model, classification is conducted by pair-wise computations of the distance between a testing point with all training points, which has a computational complexity of $O(N_{\text{te}} N_{\text{tr}} d)$. Under TNPE model, an extra time is required for embedding the testing data, which is $O(r^2 d N_{\text{te}})$. However, less time is needed in classification by applying KNN in a reduced dimension, which is $O(N_{\text{tr}} N_{\text{te}} r^n)$. Thus the total complexity is $O(r^2 d N_{\text{te}} + N_{\text{tr}} N_{\text{te}} r^n)$. Similarly, under TTNPE-ATN algorithm, embedding takes an extra computation time of $O(N_{\text{te}} r^2 d)$, but a significantly less time used in classification, which is $O(N_{\text{te}} N_{\text{tr}} r)$. Thus the total complexity is $O(N_{\text{te}} r^2 d + N_{\text{te}} N_{\text{tr}} r)$.

The comparison of the three algorithms is shown in Table 2.2, where TTNPE-ATN exhibates a great advantage in storage and computation for classification after embedding.

2.4.5 Results

In this section, we test our proposed tensor embedding on image datasets, where the 2D images are reshaped into multi-mode tensors. Reshaping images to tensors is a common practice to compare tensor algebraic approaches [58] since it captures the low rank property from the data and exhibits improved data representation. The embedding is evaluated based on KNN classification, where an effective embedding that preserves neighbor information would give classification results close to that of KNN classification at lower compression ratios. We compare the proposed TTNPE-ATN algorithm with Tucker decomposition based neighbor preserving embedding (TNPE) algorithm as proposed in [41]. We further note that the authors of [41] compared their approach with different approaches based on vectorization of data, including Neighborhood Preserving Embedding (NPE), Locality Preserving Projection (LPP), Principal Component Analysis (PCA), and Local Discriminant Embedding (LDE). Since the approach in [41] was shown to outperform these approaches, we do not consider these vectorized data approaches in our comparison. Note that the tensor train rank, which determines the compression ratio, is learnt from the Algorithm 2 based upon the selection of $\tau \in (0, 1]$.

Weizmann Face Database Weizmann Face Database [43] is a dataset that includes 26 human faces with different expressions and lighting conditions. 66 images from each of the 10 randomly selected people are used for multi-class classification, where 20 images from each person are selected for training and the remaining images are used for testing. The experiment is repeated 10 times (for the same 10 people, but random choices of the 20 training images per person) and the averaged classification errors are shown in Fig. 2.8. Each image is down sampled to 64×44 for ease of

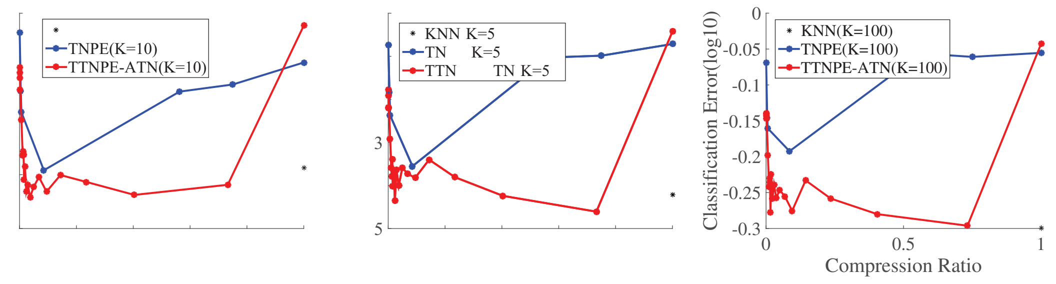


Fig. 2.8.: Classification Error in log 10 scale for Weizmann Face Database for the three models.

computation and is further reshaped to a 5-mode tensor of dimension $4 \times 4 \times 4 \times 4 \times 11$ to apply the TNPE and TTNPE-ATN algorithms. 10, 50, and 100 neighbors are considered to build the graph (from left to right) and the KNN from the same number of neighbors in the embedded space are used for classification. Since KNN does not compress the data, it results in a single point at a compression ratio of 1.

We show that TTNPE-ATN performs better than TNPE when the compression ratio is lower than 0.9, indicating TTNPE-ATN better captures the localized features in the dataset thus yielding better embedding under low compression ratios. With the increase of compression ratio, the classification error for TTNPE-ATN algorithm first decreases, which is because the data structure can be better captured with increasing compression ratio (lower compression). The classification error then increases with compression ratio since the embedding overfits the background noise in the images. Similar trend happens for TNPE algorithm. We note that for a compression ratio of 1, the result for TTNPE-ATN do not match that of KNN since we are learning at-most 200-rank space (due to 20 training images for each of 10 people) while the overall data dimension is 64×44, thus giving an approximation at the compression ratio of 1. Increasing K helps preserve more neighbors for embedding, and the neighbor structure is preserved better. Further, the best classification results given by TTNPE-ATN are even better than the classification results given by KNN algorithm, indicating TTNPE-ATN gives better neighborhood preserving embedding as compared to the TNPE algorithm.

Reshaping is investigated to verify if the performance of the embedding is subject to the empirically selected reshaping dimension ($4 \times 4 \times 4 \times 4 \times 11$). The optimal reshaping dimension has been empirically investigated in [15], where a moderate reshaping gives the best data representation of the multi-dimensional data. Fig. 2.9 considers two of the possible reshapings, $4 \times 4 \times 4 \times 4 \times 11$ and $8 \times 8 \times 4 \times 11$, and illustrates that both TTNPE-ATN and TNPE are not very sensitive to the reshaping method. Further, TTNPE-ATN performs better than TNPE in both the considered reshaping scenarios.

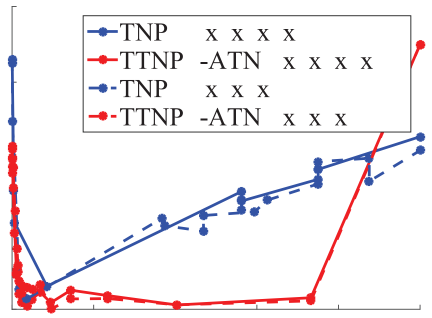


Fig. 2.9.: Classification Error in log 10 scale for Weizmann dataset under reshaping $4 \times 4 \times 4 \times 4 \times 11$ and $8 \times 8 \times 4 \times 11$.

Noise perturbation has been investigated for TTNPE-ATN algorithm in Fig. 2.10, where 20dB, 15dB, 10dB, and 5dB Gaussian noise is added to the data. The performance of TTNPE-ATN algorithm downgrades when the noise increases, while TTNPE-ATN still out-performs than TNPE on clean data when noise is less then 10dB.

Execution time for tensor embedding on Weizmann dataset is illustrated in Fig. 2.11, where we see that the proposed TTNPE-ATN is faster than TNPE in all of subspace learning, multi-dimensional data embedding, and embedded data classification operations. We also note the time for subspace learning dominates the computation

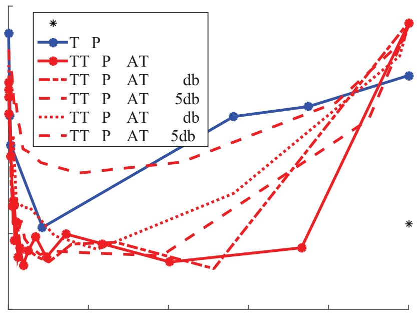


Fig. 2.10.: Classification Error in log 10 scale for Weizmann dataset under noise level 20dB, 15dB, 10dB, and 5dB.

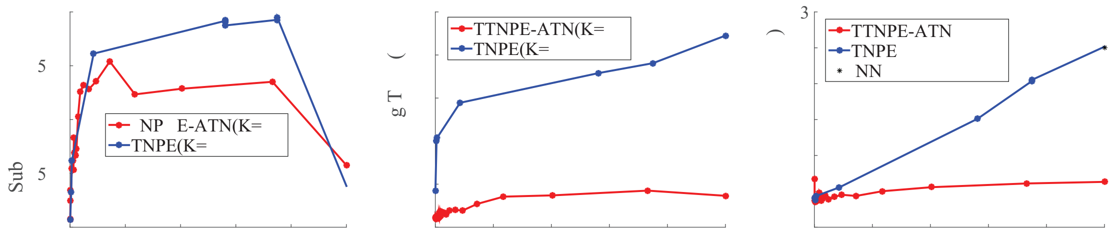


Fig. 2.11.: CPU time for TNPE, TTNPE-ATN and KNN on Weizmann dataset. From left to right are cpu time for subspace learning, multi-dimensional data embedding, and embedded data classification. The execution time is analyzed using Weizmann dataset when K is chosen to be 10

time. Further, the summation of embedding time and classification time is also lower for TTNPE-ATN as compared to KNN.

MNIST Dataset We use the MNIST dataset [42], which consists 60000 handwritten digits of size 28×28 from 0 to 9, to further investigate the embedding performance when the number of training samples is large. Each image is reshaped to $4 \times 7 \times 4 \times 7$ tensor. We perform binary classification for digits 1 and 2 by using 600 training samples from each digit. Figure 2.12 shows the classification performance of the three algorithms (KNN on data directly, TNPE, and TTNPE-ATN) when different values

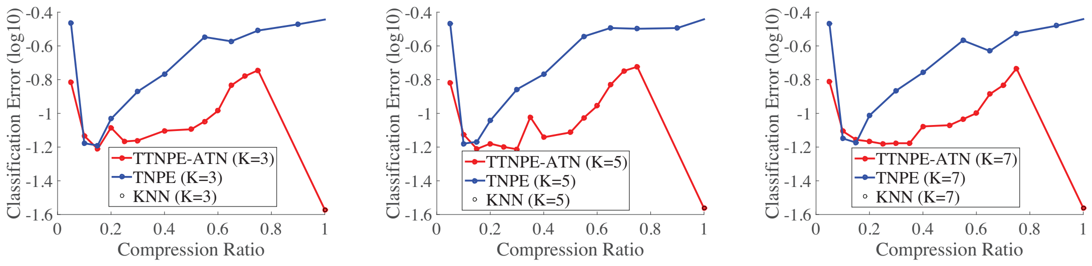


Fig. 2.12.: Classification Error in $\log 10$ scale for MNIST for the three models.

of $K = 3, 5, 7$ neighbors are used to construct the graph (from left to right). The same value of K is used for classification in the embedded space. 1000 out of sample images from each digit are selected for testing.

The results in Fig. 2.12 are averaged over 10 independent experiments (over the choice of 600 training and 1000 test samples).

We first note that the proposed TTNPE-ATN is the same as the standard KNN for that point when the training sample size is sufficient large (since the number of training samples do not limit the performance). Further, as the compression ratio increases, the classification error of the proposed TTNPE-ATN decreases first, since TTNPE-ATN model can effectively capture the embedded data structure. The classification error then increases since it fits the inherent noise as compared to the low TT-rank approximation of the data. *Overall, TTNPE-ATN algorithm shows comparable embedding performance as TNPE algorithm in the compression ratio region around 0.1, outperforms TNPE for higher compression ratios (lesser compression), and converges to KNN results at compression ratio of 1.*

We note that TTNPE-ATN shows a different behavior for compression ratios close to 1 in Fig. 2.12 as compared to Fig. 2.8. This is in part since the number of training samples are lower than the dimension of the data in Fig. 2.8 which implies there is

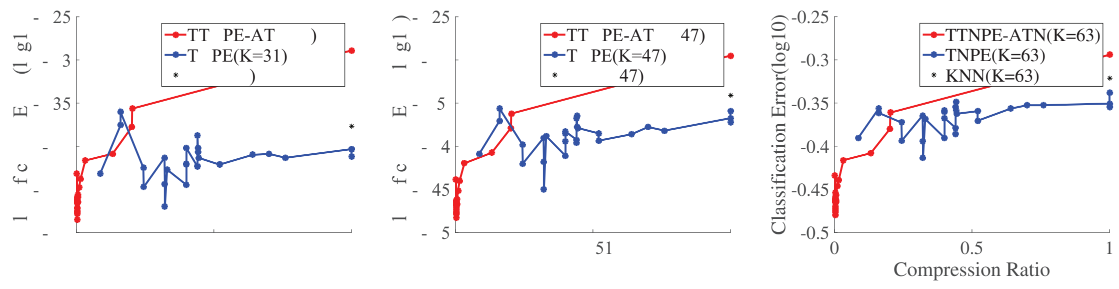


Fig. 2.13.: Classification Error in log 10 scale for Finance for the three models.

an overfitting of noise, while the number of training samples are higher than the data dimension for the results in Fig. 2.12.

Financial Market Dataset In this section, tensor embedding method is applied to four year stock price data to determine whether the stock belongs to financial or technology sector. The stock prices used in this section are the daily adjusted closing prices for the top 400 companies, ranked by the market capital as of the end of 2017, from financial and technology sectors, respectively. The data is collected from 01/10/2014 to 12/29/2017 using [59], and the daily return of each stock is computed to be used as data. We did not use the absolute stock prices, but the return rates over these days to avoid the information in the absolute value of the stock price. The time-range mentioned above had 1001 business days, thus giving us 1000 data points for stock returns. 300 stocks from each sector (out of 400) are randomly sampled for training and the remaining data are used for testing. Each time series is reshaped to a 3rd mode tensor $10 \times 10 \times 10$ for tensor embedding analysis. In the TTNPE-ATN, TNPE, and KNN algorithms, 31, 47, and 63 neighbors are selected for implementing the algorithm, respectively. Large number of neighbors empirically gives better and stable performance. Fig. 2.13 illustrates the average results of 10 independent experiments over random choice of 300 training data for each of the two sectors.

Financial data is known to be noisy. However, we note that both the TNPE and TTNPE-ATN embedding algorithms outperform KNN, thus the low dimensional tensor embedding is able to better reduce noise from the data. TTNPE-ATN algorithm classifies data more accurately in the low compression ratio regime while starts to degrade for compression ratio greater than 0.1, which is mainly due to over-fitting the noise. However, TTNPE-ATN still outperforms TNPE when the compression ratio is smaller than 0.3.

2.5 Conclusion

This paper proposes a novel algorithm for non-linear Tensor Train Neighborhood Preserving Embedding (TTNPE-ATN) for tensor data classification. We investigate the tradeoffs between error, storage, and computation and evaluate the method on several vision datasets. We further show that TTNPE-ATN algorithm exhibits improved classification performance and better dimensionality reduction among the baseline approaches, and has lower computational complexity as compared to Tucker neighborhood preserving embedding method. In the future, we will investigate the convergence of tensor network optimization and provide the theoretical gap between TTNPE-ATN and TTNPE-TN. While there has been work on parameter selection for matrix-based approaches [60, 61], finding the thresholding parameter for TTNPE is an interesting future research direction.

3. MISSING DATA COMPLETION

3.1 Introduction

Using the matrix product state (MPS) representation of the recently proposed tensor ring (TR) decompositions, we propose a TR completion algorithm, which is an alternating minimization algorithm that alternates over the factors in the MPS representation. This development is motivated in part by the success of matrix completion algorithms that alternate over the (low-rank) factors. We propose a novel initialization method and analyze the computational complexity of the TR completion algorithm. The numerical comparison between the TR completion algorithm and the existing algorithms that employ a low rank tensor train (TT) approximation for data completion shows that our method outperforms the existing ones for a variety of real computer vision settings, and thus demonstrates the improved expressive power of tensor ring as compared to tensor train.

3.2 Related Work

Tensor decompositions for representing and storing data have recently attracted considerable attention due to their effectiveness in compressing data for statistical signal processing [35, 36, 62–64]. We focus on Tensor Ring (TR) decomposition [23] and in particular its relation to Matrix Product States (MPS) [65] representation for tensor and use it for completing data from missing entries. In this context our algorithm is motivated by recent work in matrix completion where under a suitable initialization an alternating minimization algorithm [66, 67] over the low rank factors is able to accurately predict the missing data.

Recently, tensor networks, considered as the generalization of tensor decompositions, have emerged as the potentially powerful tools for analysis of large-scale tensor data [65]. The most popular tensor network is the Tensor Train (TT) representation, which for an order-d tensor with each dimension of size n requires $O(dnr^2)$ parameters, where r is the rank of each of the factors, and thus allows for the efficient data representation [21]. Tensor train decompositions have been recently considered in [29, 68] and the authors in [29, 68] considered the completion of data via an alternating least square method.

Although TT format has been widely applied in numerical analysis, its applications to image classification and completion are rather limited [29, 36, 68]. As outlined in [23], TT decomposition suffers from the following limitations. Namely, (i) TT model requires rank-1 constraints to the border factors, (ii) TT ranks are typically small for near-border factors and large for the middle factors, and (iii) the multiplications of the TT factors are not permutation invariant. In order to alleviate those drawbacks, a tensor ring (TR) decomposition has been proposed in [23]. TR decomposition removes the unit rank constraints for the boundary tensor factors and utilizes a trace operation in the decomposition. The multilinear products between factors also have no strict ordering and the factors can be circularly shifted due to the properties of the trace operation. This paper provides novel algorithms for data completion when the data is modeled as a TR decomposition.

For data completion using tensor decompositions, one of the key attributes is the notion of the rank. Even though the rank in TR is a vector, we can assume all ranks to be the same, unlike that in TT case where the intermediate ranks are higher, thus providing a single parameter that can be tuned based on the data and the number of samples available. The use of trace operation in the tensor ring structure brings challenges for completion as compared to that for tensor train decomposition. The tensor ring structure is equivalent to a cyclic structure in tensor networks [66], and understanding this structure can help understand completion for more general tensor networks. We propose an alternating minimization algorithm for the tensor ring completion. The initialization of the algorithm is an extension of TT approximation algorithm in [21] after zero-filling the missing data. Further, all the sub-problems in alternating minimization are converted to efficient least square problems, thus significantly improving the complexity of each sub-problem. We also analyze the storage and computational complexity of the proposed algorithm.

We note that, to the best of our knowledge, tensor ring completion has never been investigated for tensor completion, even though tensor ring factorization has been proposed in [23]. The different novelties as compared to [23] include the initialization algorithm, exclusion of tensor factor normalization, conversion of tensor completion problem into different least square sub-problems, and analysis of complexity in storage and computation.

The proposed algorithm is evaluated on a variety of datasets, including Einstein's image, Extended YaleFace Dataset B, and high speed video. The results are compared with the tensor train completion algorithms in [29, 68], and the additional structure in the tensor ring is shown to significantly improve the performance as compared to using the TT structure.

3.3 Background

We first provide the fundamental background knowledge on tensor operation.

Definition 3.3.1 *(Mode-i unfolding [69]) Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ be a n-mode tensor. Mode-i unfolding of $\mathcal{X}$, denoted as $\mathcal{X}_{[i]}$, matrized the tensor $\mathcal{X}$ by putting the i^{th} mode in the matrix rows and remaining modes with the original order in the columns such that*

$$\mathcal{X}_{[i]} \in \mathbb{R}^{I_i \times (I_1 \cdots I_{i-1} I_{i+1} \cdots I_n)}. \tag{3.1}$$

Definition 3.3.2 *(Mode-i canonical matrization [69]) Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ be an n^{th} order tensor, the mode-i canonical matrization gives*

$$\mathcal{X}_{<i>} \in \mathbb{R}^{(\prod_{t=1}^{i} I_t) \times (\prod_{t=i+1}^{n} I_t)}, \tag{3.2}$$

such that any entry in $\mathcal{X}_{<i>}$ *satisfies*

$$\begin{aligned}
&\mathcal{X}_{<i>}(i_1 + (i_2 - 1)I_1 + \cdots + (i_k - 1) \mathop{}_{t=1}^{k-1} I_t, \\
&i_{k+1} + (i_{k+2} - 1)I_{k+1} + \cdots + (i_n - 1) \mathop{}_{t=k+1}^{n-1} I_t) \\
=&\mathcal{X}(i_1, \cdots, i_n).
\end{aligned} \tag{3.3}$$

Definition 3.3.3 *(Tensor Connect Product) Let* $\mathcal{U}_i \in \mathbb{R}^{R_{i-1} \times I_i \times R_i}, i = 1, \cdots, n$ *be* n *3rd-order tensors, the tensor connect product between* $\mathcal{U}_j$ *and* $\mathcal{U}_{j+1}$ *is defined as,*

$$\mathcal{U}_j \mathcal{U}_{j+1} \in \mathbb{R}^{R_{j-1} \times (I_j I_{j+1}) \times R_{j+1}} = \textit{reshape}\left(\mathbf{L}(\mathcal{U}_j) \times \mathbf{R}(\mathcal{U}_{j+1})\right). \tag{3.4}$$

Thus, the tensor connect product n *MPSs is*

$$\mathcal{U} = \mathcal{U}_1 \cdots \mathcal{U}_n \in \mathbb{R}^{R_0 \times (I_1 \cdots I_n) \times R_n}. \tag{3.5}$$

Tensor connect product gives the product rule for the production between 3-order tensors, just like the matrix product as for 2-order tensor. Under matrix case, $\mathcal{U}_j \in \mathbb{R}^{1 \times I_j \times R_j}$, $\mathcal{U}_{j+1} \in \mathbb{R}^{R_j \times I_{j+1} \times 1}$. Thus tensor connect product gives the vectorized solution of matrix product.

We then define an operator f that applies on $\mathcal{U}$. Let $\mathcal{U} \in \mathbb{R}^{R_0 \times (I_1 \cdots I_n) \times R_n}$ be the 3-order tensor, $R_0 = R_n$, and let f be a reshaping operator function that reshapes a 3-order tensor $\mathcal{U}$ to a tensor of dimension $\mathcal{X}$ of dimension $\mathbb{R}^{I_1 \times \cdots \times I_n}$, denoted as

$$\mathcal{X} = f(\mathcal{U}), \tag{3.6}$$

where $\mathcal{X}(i_1, \cdots, i_n)$ is generated by

$$\mathcal{X}(i_1, \cdots, i_n) = \operatorname{tr}(\mathcal{U}(:, i_1 + (i_2 - 1)I_1 + \cdots + (i_n - 1)I_{n-1}, :)). \tag{3.7}$$

Thus a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ with tensor ring structure is equivalent to

$$\mathcal{X} = f(\mathcal{U}_1 \cdots \mathcal{U}_n). \tag{3.8}$$

Similar to matrix transpose, which can be regarded as an operation that cyclic swaps the two modes for a 2-order tensor, we define a 'tensor permutation' to describe the cyclic permutation of the tensor modes for a higher order tensor.

Definition 3.3.4 *(Tensor Permutation) For any n-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$, the i^{th} tensor permutation is defined as $\mathcal{X}^{P_i} \in \mathbb{R}^{I_i \times I_{i+1} \times \cdots \times I_n \times I_1 \times I_2 \times \cdots \times I_{i-1}}$ such that $\forall_i, j_i \in [1, I_i]$*

$$\mathcal{X}^{P_i}(j_i, \cdots, j_n, j_1, \cdots, j_{i-1}) = \mathcal{X}(j_1, \cdots, j_n). \tag{3.9}$$

Then we have the following result.

Lemma 5 *If $\mathcal{X} = f(\mathcal{U}_1 \cdots \mathcal{U}_n)$, then $\mathcal{X}^{P_i} = f(\mathcal{U}_i \mathcal{U}_{i+1} \cdots \mathcal{U}_n \mathcal{U}_1 \cdots \mathcal{U}_{i-1})$.*

Proof Based on definition of tensor permutation in Definition 3.3.4, on the left hand side, the $(j_1, ..., j_n)$ entry of the tensor is

$$\mathcal{X}^{P_i}(j_1, ..., j_n) = \mathcal{X}(j_{n-i+2}, ..., j_n, j_1, ..., j_{n-i+1}). \tag{3.10}$$

On the right hand side, the $(j_1, ..., j_n)$ entry of the tensor gives

$$\begin{aligned} & f(\mathcal{U}_i \cdots \mathcal{U}_{i-1})(j_1, \cdots, j_n) \\ =& \text{Trace}(\mathcal{U}_i(:, j_1, :)\mathcal{U}_{i+1}(:, j_2, :)...\mathcal{U}_n(:, j_{n-i+1}, :)\mathcal{U}_1(:, j_{n-i+2}, :) \cdots \mathcal{U}_{i-1}(:, j_n, 1)). \end{aligned} \tag{3.11}$$

Since trace is invariant under cyclic permutations, we have

$$\begin{aligned} & \text{Trace}(\mathcal{U}_i(:, j_1, :)\mathcal{U}_{i+1}(:, j_2, :)...\mathcal{U}_n(:, j_{n-i+1}, :)\mathcal{U}_1(:, j_{n-i+2}, :) \cdots \mathcal{U}_{i-1}(:, j_n, :)) \\ =& \text{Trace}(\mathcal{U}_1(:, j_{n-i+2}, :) \cdots \mathcal{U}_{i-1}(:, j_n, :)\mathcal{U}_i(:, j_1, :)\mathcal{U}_{i+1}(:, j_2, :)...\mathcal{U}_n(:, j_{n-i+1}, :)) \\ =& f(\mathcal{U}_1 \cdots \mathcal{U}_n)(j_{n-i+2}, \cdots, j_n, j_1, \cdots, j_{n-i+1}), \end{aligned} \tag{3.12}$$

which equals to the right hand side of equation (3.10). Since any entries in $\mathcal{X}^{P_i}$ are the same as those in $\mathcal{U}_i \mathcal{U}_{i+1} \cdots \mathcal{U}_n \mathcal{U}_1 \cdots \mathcal{U}_{i-1}$, the claim is proved. ■

With this background and basic constructs, we now outline the main problem setup.

3.4 Tensor Ring Completion Algorithm

3.4.1 Problem Formulation

Given a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ that is partially observed at locations Ω, let $\mathcal{P}_\Omega \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ be the corresponding binary tensor in which 1 represents an observed entry and 0 represents a missing entry. The problem is to find a low tensor ring rank (TR-Rank) approximation of the tensor $\mathcal{X}$, denoted as $f(\mathcal{U}_1 \cdots \mathcal{U}_n)$, such that the recovered tensor matches $\mathcal{X}$ at $\mathcal{P}_\Omega$. This problem is referred as the tensor completion problem under tensor ring model, which is equivalent to the following problem

$$\min_{\mathcal{U}_{i:i=1,\cdots,n}} \quad \mathcal{P}_\Omega \circ (f(\mathcal{U}_1 \cdots \mathcal{U}_n) - \mathcal{X}) \;{}_F^2. \tag{3.13}$$

Note that the rank of the tensor ring R is predefined and the dimension of $\mathcal{U}_{i:i=1,\cdots,n}$ is $\mathbb{R}^{R \times I_i \times R}$.

To solve this problem, we propose an algorithm, referred as Tensor Ring completion by Alternating Least Square (TR-ALS) to solve the problem in two steps.

- Choose an initial starting point by using Tensor Ring Approximation (TRA). This initialization algorithm is detailed in Section 3.4.2.
- Update the solution by applying Alternating Least Square (ALS) that alternatively (in a cyclic order) estimates a factor say $\mathcal{U}_i$ keeping the other factors fixed. This algorithm is detailed in Section 3.4.3.

3.4.2 Tensor Ring Approximation (TRA)

A heuristic initialization algorithm, namely TRA, for solving (3.13) is proposed in this section. The proposed algorithm is a modified version of tensor train decomposition as proposed in [21]. We first perform a tensor train decomposition on the zero-filled data, where the rank is constrained by Singular Value Decomposition (SVD). Then, an approximation for the tensor ring is formed by extending the obtained factors to the desired dimensions by filling the remaining entries with small random numbers. We note that the small entries show faster convergence as compared to zero entries based on our considered small examples, and thus motivates the choice in the algorithm. Further, non-zero random entries help the algorithm initialize with larger ranks since the TT decomposition has the corner ranks as 1. Having non-zero

Algorithm 3 Tensor Ring Approximation (TRA)

```
Input: Missing entry zero filled tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n}$, TR-Rank $R$, small random variable depicting the standard deviation of the added normal random variable $\sigma$
Output: Tensor train decomposition $\mathcal{U}_{i:i=1,\cdots,n} \in \mathbb{R}^{R \times I_i \times R}$
1: Apply mode-1 canonical matricization for $\mathcal{X}$ and get matrix $\mathbf{X}_1 = \mathcal{X}_{<1>} \in \mathbb{R}^{I_1 \times (I_2 I_3 \cdots I_n)}$
2: Apply SVD and threshold the number of singular values to be $T_1 = \min(R, I_1, I_2 \cdots I_n)$, such that $\mathbf{X}_1 = \mathbf{U}_1 \mathbf{S}_1 \mathbf{V}_1$ , $\mathbf{U}_1 \in \mathbb{R}^{I_1 \times T_1}$, $\mathbf{S}_1 \in \mathbb{R}^{T_1 \times T_1}$, $\mathbf{V}_1 \in \mathbb{R}^{T_1 \times (I_2 I_3 \cdots I_n)}$. Reshape $\mathbf{U}_1$ to $\mathbb{R}^{1 \times I_1 \times T_1}$ and extend it to $\mathcal{U}_1 \in \mathbb{R}^{R \times I_1 \times R}$ by filling the extended entries by random normal distributed values sampled from $\mathcal{N}(0, \sigma^2)$.
3: Let $\mathbf{M}_1 = \mathbf{S}_1 \mathbf{V}_1 \in \mathbb{R}^{T_1 \times (I_2 I_3 \cdots I_n)}$.
4: for $i = 2$ to $n-1$ do
5:     Reshape $\mathbf{M}_{i-1}$ to $\mathbf{X}_i \in \mathbb{R}^{(T_{i-1} I_i) \times (I_{i+1} I_{i+2} \cdots I_n)}$.
6:     Compute SVD and threshold the number of singular values to be $T_i = min(R, T_{i-1} I_i, I_{i+1} \cdots I_n)$, such that $\mathbf{X}_i = \mathbf{U}_i \mathbf{S}_i \mathbf{V}_i$ , $\mathbf{U}_i \in \mathbb{R}^{(T_{i-1} I_i) \times T_i}$, $\mathbf{S}_i \in \mathbb{R}^{T_i \times T_i}$, $\mathbf{V} \in \mathbb{R}^{T_i \times (I_{i+1} I_{i+2} \cdots I_n)}$. Reshape $\mathbf{U}_i$ to $\mathbb{R}^{T_{i-1} \times I_i \times T_i}$ and extend it to $\mathcal{U}_i \in \mathbb{R}^{R \times I_i \times R}$ by filling the extended entries by random normal distributed values sampled from $\mathcal{N}(0, \sigma^2)$.
7:     Set $\mathbf{M}_i = \mathbf{S}_i \mathbf{V}_i \in \mathbb{R}^{T_i \times (I_{i+1} I_{i+2} \cdots I_n)}$
8: end for
9: Reshape $\mathbf{M}_{n-1} \in \mathbb{R}^{T_{n-1} \times I_n}$ to $\mathbb{R}^{T_{n-1} \times I_n \times 1}$, and extend it to $\mathcal{U}_n \in \mathbb{R}^{R \times I_n \times R}$ by filling the extended entries by random normal distributed values sampled from $\mathcal{N}(0, \sigma^2)$ to get $\mathcal{U}_n$
10: Return $\mathcal{U}_1, \cdots, \mathcal{U}_n$
```

entries can help the algorithm not getting stuck in a local optima of low corner rank. The TRA algorithm is given in Algorithm 3.

3.4.3 Alternating Least Square

The proposed tensor ring completion by alternating least square method (TR-ALS) solves (3.13) by solving the following problem for each i iteratively. The factors are initialized from the TRA algorithm presented in the previous section.

$$\mathcal{U}_i = \arg\min_{\mathcal{Y}} \ \mathcal{P}_\Omega \circ f(\mathcal{U}_1 \cdots \mathcal{U}_{i-1}\mathcal{Y}\mathcal{U}_{i+1} \cdots \mathcal{U}_n) - \mathcal{X}_\Omega) \ {}_F^2. \tag{3.14}$$

Lemma 6 *When $i = 1$, solving*

$$\mathcal{U}_i = \arg\min_{\mathcal{Y}} \ \mathcal{P}_\Omega \circ f(\mathcal{U}_1 \cdots \mathcal{U}_{i-1}\mathcal{Y}\mathcal{U}_{i+1} \cdots \mathcal{U}_n) - \mathcal{X}_\Omega) \ {}_F^2 \tag{3.15}$$

is equivalent to

$$\mathcal{U}_i = \arg\min_{\mathcal{Y}} \ \mathcal{P}_\Omega^{P_i} \circ f(\mathcal{Y}\mathcal{U}_{i+1} \cdots \mathcal{U}_n\mathcal{U}_1 \cdots \mathcal{U}_{i-1}) - \mathcal{X}_\Omega^{P_i} \ {}_F^2. \tag{3.16}$$

Proof First we note that tensor permutation does not change tensor Frobenius norm as all the entries remain the same as those before the permutation. In Lemma 2, we have

$$\mathcal{U}_i = \arg\min_{\mathcal{Y}} \ \mathcal{P}_\Omega \circ f(\mathcal{U}_1 \cdots \mathcal{U}_{i-1}\mathcal{Y}\mathcal{U}_{i+1} \cdots \mathcal{U}_n) - \mathcal{X}_\Omega) \ {}_F^2. \tag{3.17}$$

Since the permutation operation does not change the Frobenius norm, equivalently we have

$$\mathcal{U}_i = \arg\min_{\mathcal{Y}} \ \mathcal{P}_\Omega^{P_i} \circ (f(\mathcal{U}_1 \cdots \mathcal{U}_{i-1}\mathcal{Y}\mathcal{U}_{i+1} \cdots \mathcal{U}_n))^{P_i} - \mathcal{X}_\Omega^{P_i} \ {}_F^2. \tag{3.18}$$

Based on Lemma 1, we have

$$(f(\mathcal{U}_1 \cdots \mathcal{U}_{i-1}\mathcal{Y}\mathcal{U}_{i+1} \cdots \mathcal{U}_n))^{P_i} = f(\mathcal{Y}\mathcal{U}_{i+1} \cdots \mathcal{U}_n\mathcal{U}_1 \cdots \mathcal{U}_{i-1}), \tag{3.19}$$

thus equation (3.18) becomes

$$\mathcal{U}_i = \arg\min_{\mathcal{Y}} \ \mathcal{P}_\Omega^{P_i} \circ f(\mathcal{Y}\mathcal{U}_{i+1} \cdots \mathcal{U}_n\mathcal{U}_1 \cdots \mathcal{U}_{i-1}) - \mathcal{X}_\Omega^{P_i} \ {}_F^2. \tag{3.20}$$

Thus we prove our claim. ■

Since the format of (3.16) is exactly the same for each i when the other factors are known, it is enough to describe solving a single $\mathcal{U}_k$ without loss of generality. Based on Lemma 6, we need to solve the following problem.

$$\mathcal{U}_k = \arg\min_{\mathcal{Y}} \ \mathcal{P}_{\Omega}^{P_k} \circ f(\mathcal{Y}\mathcal{U}_{k+1} \cdots \mathcal{U}_n \mathcal{U}_1 \cdots \mathcal{U}_{k-1}) - \mathcal{X}_{\Omega}^{P_k} \ {}_F^2. \tag{3.21}$$

We further apply mode-k unfolding, which gives the equivalent problem

$$\mathcal{U}_k = \arg\min_{\mathcal{Y}} \ \mathcal{P}_{\Omega\ [k]}^{P_k} \circ f(\mathcal{Y}\mathcal{U}_{k+1} \cdots \mathcal{U}_n \mathcal{U}_1 \cdots \mathcal{U}_{k-1})_{[k]} - \mathcal{X}_{\Omega\ [k]}^{P_k} \ {}_F^2, \tag{3.22}$$

where $\mathcal{P}_{\Omega\ [k]}^{P_k}$, $f(\mathcal{Y}\mathcal{U}_{k+1} \cdots \mathcal{U}_n \mathcal{U}_1 \cdots \mathcal{U}_{k-1})_{[k]}$ and $\mathcal{X}_{\Omega\ [k]}^{P_k}$ are matrices with dimension $\mathbb{R}^{I_k \times (I_{k+1} \cdots I_n I_1 \cdots I_{k-1})}$.

The trick in solving (3.22) is that each slice of tensor $\mathcal{Y}$, denoted as $\mathcal{Y}(:, i_k, :), i_k \in \{1, \cdots, I_k\}$ which corresponds to each row of $\mathcal{P}_{\Omega\ [k]}^{P_k}$, $f(\mathcal{Y}\mathcal{U}_{k+1} \cdots \mathcal{U}_n \mathcal{U}_1 \cdots \mathcal{U}_{k-1})_{[k]}$ and $\mathcal{X}_{\Omega\ [k]}^{P_k}$, can be solved independently, thus equation (3.22) can be solved by solving I_k equivalent subproblems

$$\mathcal{U}_k(:, i_k, :) = \arg\min_{\mathcal{Z} \in \mathbb{R}^{R \times 1 \times R}} \ \mathcal{P}_{\Omega\ [k]}^{P_k}(i_k, :) \circ f(\mathcal{Z}\mathcal{U}_{k+1} \cdots \mathcal{U}_{k-1}) - \mathcal{X}_{\Omega\ [k]}^{P_k}(i_k, :) \ {}_F^2. \tag{3.23}$$

Let $\mathcal{B}^{(k)} = \mathcal{U}_{k+1} \cdots \mathcal{U}_n \mathcal{U}_1 \cdots \mathcal{U}_{k-1} \in \mathbb{R}^{R \times (I_{k+1} \cdots I_n I_1 \cdots I_{k-1}) \times R}$, Ω_{i_k} be the observed entries in vector $\mathcal{X}_{[k]}(i_k, :)$, thus $\mathcal{B}_{\Omega_{i_k}}^{(k)} \in \mathbb{R}^{R \times (I_{k+1} \cdots I_n I_1 \cdots I_{k-1})_{\Omega_{i_k}} \times R}$ are the components in $\mathcal{B}^{(k)}$ such that $\mathcal{P}_{\Omega\ [k]}^{P_k}(i_k, (I_{k+1} \cdots I_n I_1 \cdots I_{k-1})_{\Omega_{i_k}})$ are observed. Thus equation (3.23) is equivalent to

$$\mathcal{U}_k(:, i_k, :) = \arg\min_{\mathcal{Z}} \ f(\mathcal{Z}\mathcal{B}_{\Omega_{i_k}}^{(k)}) - \mathcal{X}_{\Omega\ [k]}^{P_k}(i_k, (I_{k+1} \cdots I_n I_1 \cdots I_{k-1})_{\Omega_{i_k}})) \ {}_F^2. \tag{3.24}$$

We regard $\mathcal{Z} \in \mathbb{R}^{R \times 1 \times R}$ as a matrix $\mathbf{Z} \in \mathbb{R}^{R \times R}$. Since the Frobenius norm of a vector in (3.24) is equivalent to entry-wise square summation of all entries, we rewrite (3.24) as

$$\mathcal{U}_k(:, i_k, :) = \arg\min_{\mathbf{Z} \in \mathbb{R}^{R \times R}} \ {}_{j \in \Omega_{i_k}} \ \mathrm{tr}(\mathbf{Z} \times \mathcal{B}_{\Omega_{i_k}}^{(k)}(:, j, :)) - \mathcal{X}_{\Omega\ [k]}^{P_k}(i_k, j) \ {}_F^2. \tag{3.25}$$

Lemma 7 *Let* $\mathbf{A} \in \mathbb{R}^{r_1 \times r_2}$ *and* $\mathbf{B} \in \mathbb{R}^{r_2 \times r_1}$ *be any two matrices, then*

$$\mathit{Trace}(\mathbf{A} \times \mathbf{B}) = \mathit{vec}(\mathbf{B} \ \ ) \ \ \mathit{vec}(\mathbf{A}). \tag{3.26}$$

Based on Lemma 7, (3.25) becomes

$$\mathcal{U}_k(:, i_k, :) = \arg\min_{\mathbf{Z}} \underset{j\in\Omega^{(k)}_{i_k}}{} \ \mathrm{vec}((\mathcal{B}^{(k)}_{\Omega_{i_k}}(:, j, :))\ \ )\ \mathrm{vec}(\mathbf{Z}) - \mathcal{X}^{P_k}_{\Omega\ [k]}(i_k, j)\ {}^{2}_{F}. \tag{3.27}$$

Then the problem for solving $\mathcal{U}_k[:, i_k, :]$ becomes a least square problem. Solving I_k least square problem would give the optimal solution for $\mathcal{U}_k$. Since each $\mathcal{U}_{i:i=1,\cdots,n}$ can solved by a least square method, tensor completion under tensor ring model can be solved by taking orders to update $\mathcal{U}_{i:i=1,\cdots,n}$ until convergence. We note the completion algorithm does not require normalization on each MPS, unlike the decomposition algorithm [23] that normalizes all the MPSs to seek a unique factorization. The stopping criteria in TR-ALS is measured via the changes of the last tensor factors $\mathcal{U}_n$ since if the last factor does not change, the other factors are less likely to change. Details of the algorithm are given in Algorithm 3.

3.4.4 Complexity Analysis

Storage Complexity Given an n-order tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_n}$, the total amount of parameters to store is $\ {}^{n}_{i=1} I_i$, which increases exponentially with order. Under tensor ring model, we can reduce the storage space by converting each factor (except the last) one by one to being orthonormal and multiply the product with the next factor. Thus, the number of parameters to store the MPSs $\mathcal{U}_{i:i=1,\cdots,n-1}$ with orthonormal property requires storage $\ {}^{n-1}_{i=1}(R^2 I_i - R^2)$, and $\mathcal{U}_n$ with parameter $R^2 I_n$. Thus, the total amount of storage is $R^2(\ {}^{n}_{i} I_i - n + 1)$, where the tensor ring rank R can be adjusted to fit the tensor data at the desired accuracy.

Computational Complexity For each $\mathcal{U}_i$, the least square problem in (3.22) solved by pseudo-inverse gives a computational complexity $\max(O(PR^4), O(R^6))$, where P is the total number of observations. Within one iteration when n MPSs need to be updated, the overall complexity is $\max(O(nPR^4), O(nR^6))$.

We note that tensor train completion [68] gives the similar complexity as tensor ring completion. However, tensor train rank is a vector and it is hard for tuning to achieve the optimal completion. The intermediate ranks in tensor train are large in general, leading to significantly higher computational complexity of tensor train. This is alleviated in part by the tensor ring structure which can be parametrized by the tensor ring rank which can be smaller than the intermediate ranks of the tensor train in general. In addition, the single parameter in the tensor ring structure leads to an ease in characterizing the performance for different ranks and can be easily tuned for practical applications. The lower ranks lead to lower computational complexity of data completion under the tensor ring structure as compared to the tensor train structure.

3.4.5 Reshaping

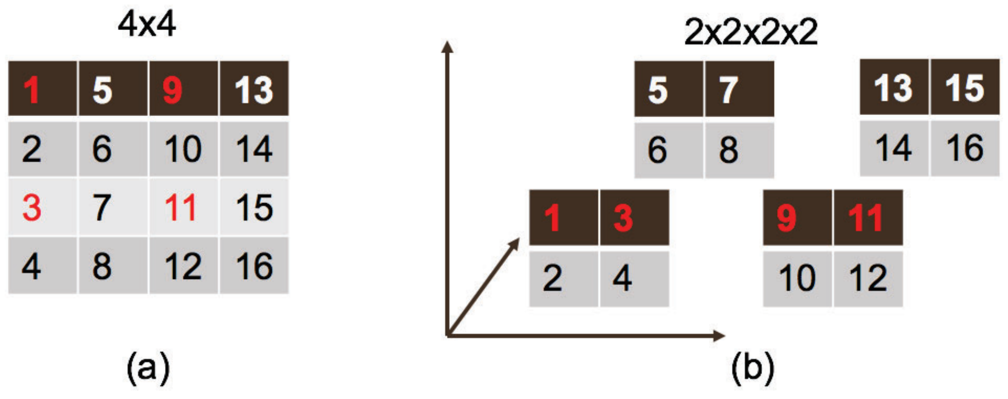


Fig. 3.1.: Reshaping a 4×4 matrix into a $2 \times 2 \times 2 \times 2$ tensor.

In the real-settings, images are reshaped into higher order tensors by re-arranging the pixels into high-dimensional array, which although is a common practice in many literatures [29, 36, 58], fundamental analysis on the reasoning of the reshaping has never been provided. As shown in Fig 3.1, reshaping a 4×4 matrix into a $2 \times 2 \times 2 \times 2$ captures the low rank structure of the image data in the sense that the sub-image by pixels $(1, 3, 9, 11)$ would be similar to that by pixels $(5, 7, 13, 15)$, $(6, 8, 14, 16)$ and pixels $(2, 4, 10, 12)$. This is attributes the spatial property of vision datasets, where the downsampled sub-image is the original image data at low resolution. In the remaining of the work, the tensor ring completion algorithm is applied to the image after reshaping into high-dimensional tensors.

3.5 Results

In this section, we compare our proposed TR-ALS algorithm with tensor train completion under alternating least square (TT-ALS) algorithm [68], which solves the tensor completion by alternating least squares under tensor train format. SiLRTC algorithm is another tensor train completion algorithm proposed in [29] and the tensor train rank is tuned based on the dimensionality of the tensor. It is selected for comparison as it shows good recovery in image completion [29]. The evaluation merit we consider is Recovery Error (RE). Let $\hat{\mathcal{X}}$ be the recovered tensor and $\mathcal{X}$ be the ground truth of the tensor. Thus, the recovery error is defined as

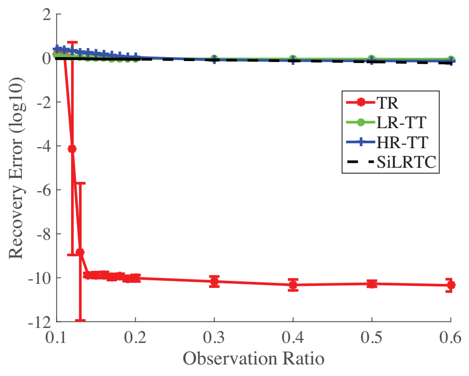
$$RE = \frac{\hat{\mathcal{X}} - \mathcal{X} \ _F}{\mathcal{X} \ _F}.$$

Tensor ring completion by alternating least square (TR-ALS) algorithm is an iterative algorithm and the maximum iteration, $maxiter$, is set to be 300. The convergence is captured by the change of the last factorization term $\mathcal{U}_n$, where the error tolerance is set to be 10^{-10}.

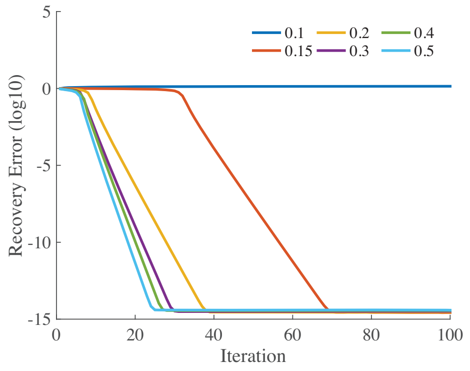
In the remaining of the section, we first evaluate the completion results for synthetic data. Then we validate the proposed TR-ALS algorithm on image completion, YaleFace image-sets completion, and video completion.

3.5.1 Synthetic Data

In this section, we consider a completion problem of a 4-order tensor $\mathcal{X} \in \mathbb{R}^{20\times20\times20\times20}$ with TR-Rank being 8 without loss of generality. The tensor is generated by a se-



(a) Recovery error (log 10) versus observation Ratio. The plots are the average of 10 experiments and the error bar are marked using one standard deviation.



(b) Convergence plot for TR-ALS under observation ratio from 0.1 to 0.5

Fig. 3.2.: Completion for synthetic data. Synthetic data is a 4_{th} order tensor of dimension $20 \times 20 \times 20 \times 20$ with TR-Rank being 8.

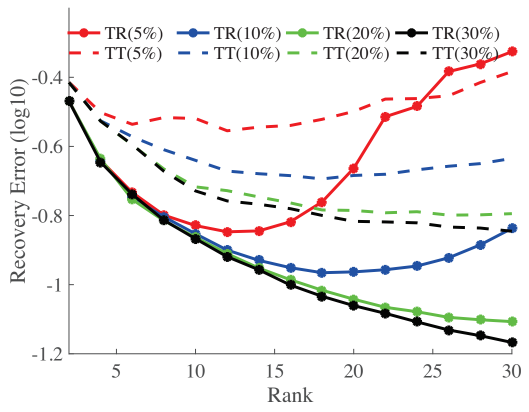
quence of connected 3-rd order tensor $\mathcal{U}_{i:i=1,\cdots,4} \in \mathbb{R}^{8\times 20\times 8}$ and every entry in $\mathcal{U}_i$ are sampled independently from a standard normal distribution.

TT-ALS is considered as a comparable to show the difference between tensor train model and tensor ring model. Two different tensor train ranks are chosen for the comparisons. The first tensor-train ranks are chosen as $[8, 8, 8]$, and the completion with these ranks is called Low rank tensor train (LR-TT) completion. The second tensor-train ranks are chosen as the double of the first ($[16, 16, 16]$), and the completion with these ranks is called High rank tensor train (HR-TT) completion. Another comparable used is the SiLRTC algorithm proposed in [29], where the rank is adjusted according to the dimensionality of the tensor data, and a heuristic factor of $f = 1$ in the proposed algorithm of [29] is selected for testing.

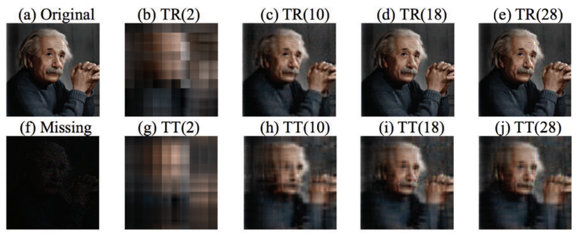
Fig.3.2a shows the completion error of TR-ALS, LR-TT, HR-TT, and SiLRTC for observation ratio from 10% to 60%. TR-ALS shows the lowest recovery error compared with other algorithms and the recovery error drops to 10^{-10} for observation ratio larger than 14%. The large completion errors of all tensor train algorithm at every observation ratio show that tensor train algorithm can not effectively complete the tensor data generated under tensor ring model. Fig. 3.2b shows the convergence of TR-ALS under sampling ratios $10\%, 15\%, 20\%, 30\%, 40\%,$ and 50%, and the plot indicates the higher the observation ratios, the faster the algorithm converges. When the observation ratio is lower than 10%, the tensor with missing data can not be completed under the proposed set-up. The fast convergence of the proposed TR-ALS algorithm indicates that alternating least square is effective in tensor ring completion.

3.5.2 Image Completion

In this section, we consider the completion of RGB Einstein Image [70], treated as a 3-order tensor $\mathcal{X} \in \mathbb{R}^{600\times600\times3}$. A reshaping operation is applied to transform the image into a 7-order tensor of size $\mathbb{R}^{6\times10\times10\times6\times10\times10\times3}$. Reshaping low order tensors into high order tensors is a common practice in literature and has shown improved performance in classification [36] and completion [29]. Fig. 3.3a shows the recovery error versus rank for TR-ALS and TT-ALS when the percentage of data observed are $5\%, 10\%, 20\%, 30\%$. At any considered ranks, TR-ALS completes the image with a better accuracy than TT-ALS. For any given percentage of observations, the recovery error first decreases as the rank increases which is caused by the increased information being captured by the increased number of parameters in the tensor structure. The recovery error then starts to increase after a thresholding rank, which can be ascribed to over-fitting. *The plot also indicates that higher the observation ratio, larger the thresholding rank, which to the best of our knowledge is reported for the first time.* Fig. 3.3b shows the recovered image of Einstein image when 10% pixels are randomly observed. TR-ALS with rank 28 gives the best recovery accuracy in the considered ranks.



(a) The recovery error versus rank for TR-ALS and TT-ALS under observation ratio $5\%, 10\%, 20\%, 30\%$.



(b) Einstein image completion when 10% of pixels are randomly observed. (a) and (f) are the original Einstein image and the Einstein image with 10% randomly observed entries. (b)-(e) are the completed images via TR-ALS with TR-Rank $2, 10, 18, 28$ and completion errors $33.97\%, 14.03\%, \mathbf{10.83}\%, 14.55\%$ respectively. (g)- (j) are the completed images via TT-ALS with TT-Rank $2, 10, 18, 28$ and completion errors $38.51\%, 22.89\%, \mathbf{20.70}\%, 23.19\%$ respectively.

Fig. 3.3.: Completion for Einstein image. Einstein image is of size $600 \times 600 \times 3$, and is reshaped into a 7-order tensor of size $6 \times 10 \times 10 \times 6 \times 10 \times 10 \times 3$ tensor for tensor ring completion

3.5.3 YaleFace Dataset Completion

In this section, we consider Extended YaleFace Dataset B [71] that includes 38 people with 9 poses under 64 illumination conditions. Each image has the size of 192×168, where we down-sample the size of each image to 48×42 for ease of computation. We consider the image subsets of 38 people under 64 illumination with 1 pose by formatting the data into a 4-order tensor in $\mathbb{R}^{48\times42\times64\times38}$, which is further reshaped into a 8-order tensor $\mathcal{X} \in \mathbb{R}^{6\times8\times6\times7\times8\times8\times19\times2}$. We consider the case

Table 3.1.: Completion error of 10% observed Extended YaleFace data via TT-ALS and TR-ALS under rank $5, 10, 15, 20, 25, 30$.

Rank	5	10	15	20	25	30
TT-ALS ($\mathbb{R}^{6\times8\times6\times7\times8\times8\times19\times2}$)	37.08%	29.65%	27.91%	26.84%	26.16%	25.55%
TR-ALS ($\mathbb{R}^{6\times8\times6\times7\times8\times8\times19\times2}$)	33.45%	**24.67%**	**20.72%**	**18.47%**	**16.92%**	**16.25%**
TR-ALS ($\mathbb{R}^{2\times3\times2\times4\times2\times3\times7\times8\times8\times19\times2}$)	33.73%	25.08%	21.20%	18.97%	17.34%	16.34%
TR-ALS ($\mathbb{R}^{48\times42\times64\times38}$)	**30.36%**	26.08%	23.74%	22.22%	21.48%	21.57%

when 10% of pixels are randomly observed. YaleFace sets completion is considered to be harder than an image completion since features under different illumination and across human features are harder to learn than information from the color channels of images. Table 3.1 shows that for any considered rank, TR-ALS recovers data better than TT-ALS and the best completion result in the given set-up is 16.25% for TR-ALS as compared with 25.55% given by TT-ALS. Further we reshape the data into an 11-order tensor and 4-order tensor to evaluate the effect of reshaped tensor size on tensor completion. The result in Table 3.1 shows that in the given reshaping set-up, reshaping tensor from 4-order tensor to 7-th order tensor significantly improve the performance of tensor completion by decreasing recovery error from 21.48% to 16.25%. However, further reshaping to 11-order tensor slightly degrades the performance of completion, resulting in an increased recovery error to 16.34%.

Fig. 3.4 shows the original image, missing images, and recovered images using TR-ALS and TT-ALS algorithms for ranks of $10, 20,$ and 30, where the completion results given by TR-ALS better captures the detail information given from the image and recovers the image with a better resolution.

3.5.4 Video completion

The video data we used in this section is high speed camera video for gun shooting [72]. It is downloaded from Youtube with 85 frames in total and each frame is

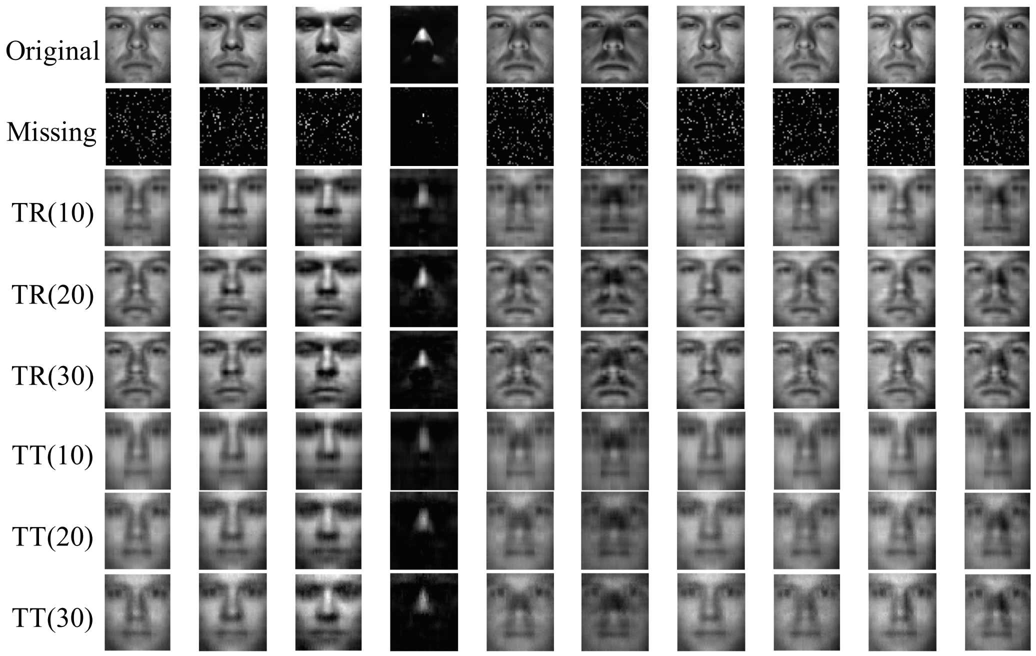


Fig. 3.4.: YaleFace dataset is sub-sampled to formulate into a tensor of size $48 \times 42 \times 64 \times 38$, which is reshaped into a 8-order tensor of size $6 \times 8 \times 6 \times 7 \times 8 \times 8 \times 19 \times 2$ for tensor ring completion. 90% of the pixels are assumed to be randomly missing. From top to bottom are original images, missing images, TR-ALS completed images with TR-Ranks $10, 20, 30$, and TT-ALS completed images with TT-Ranks $10, 20, 30$.

Table 3.2.: Completion error of 10% observed Video data via TT-ALS and TR-ALS under rank $10, 15, 20, 25, 30$.

Rank	10	15	20	25	30
TT-ALS	19.16%	14.83%	16.42%	16.86%	16.99%
TR-ALS	13.90%	10.12%	8.13%	6.88%	**6.25%**

consisted by a $100 \times 260 \times 3$ image. Thus the video is a 4-order tensor of size $100 \times 260 \times 3 \times 85$, which is further reshaped into a 11-order tensor of size $5 \times 2 \times 5 \times 2 \times 13 \times 2 \times 5 \times 2 \times 3 \times 5 \times 17$ for completion. Video is a multi-dimensional data with different color channel a time dimension in addition to the 2D image structure.

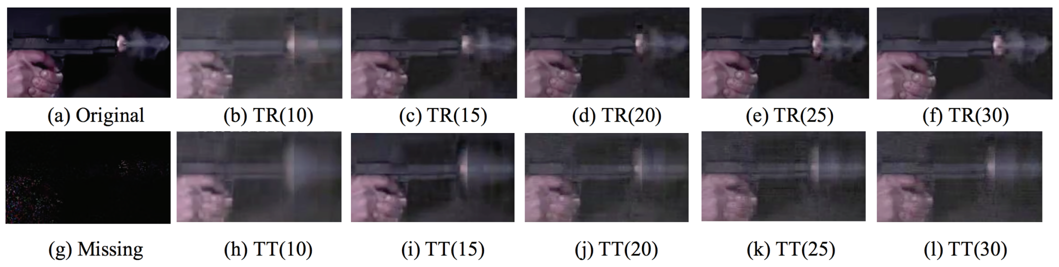


Fig. 3.5.: Gun Shot is a video of size $100 \times 260 \times 3 \times 80$ download from Youtube, which is reshaped into a 11-order tensor of size $5 \times 2 \times 5 \times 2 \times 13 \times 2 \times 5 \times 2 \times 3 \times 5 \times 17$ for tensor ring completion. 90% of the pixels are assumed to be randomly missing. (a) and (g) are the first frame of the original video and missing video. (b)-(f) are the completed frame via TR-ALS using TR-Rank $10, 15, 20, 25, 30$. (h)-(l) are the completed frame via TT-ALS using TR-Rank $10, 15, 20, 25, 30$.

In Table 3.2, we show that TR-ALS achieves 6.25% recovery error when 10% of the pixels are observed, which is much better than the best recovery error of 14.83% achieved by TT-ALS. The first frame of the video is shown in Fig. 3.5, where the first row shows the original frame and the completed frames by TR-ALS, and the second row shows the frame with missing entries and the frames completed by TT-ALS. The resolution, and the display of the bullets and the smoke depict that the proposed TR-ALS achieves better completion results as compared to the TT-ALS algorithm.

3.6 Conclusion

We propose a novel algorithm for data completion using tensor ring decomposition. This is the first paper on data completion exploiting this structure which is a non-trivial extension of the tensor train structure. Our algorithm exploits the matrix product state representation and uses alternating minimization over the low rank factors for completion. The evaluation of the proposed approach on a variety of datasets, including Einstein's image, Extended YaleFace Dataset B, and video completion demonstrates the significant improvement of tensor ring completion as compared to tensor train completion.

Deriving provable performance guarantees on tensor completion using the proposed algorithm is left as further work. In this context, the statistical machinery for proving analogous results for the matrix case [66, 67] can be used.

4. MODELL COMPRESSION

4.1 Introduction

Deep neural networks have demonstrated state-of-the-art performance in a variety of real-world applications. In order to obtain performance gains, these networks have grown larger and deeper, containing millions or even billions of parameters and over a thousand layers. The trade-off is that these large architectures require an enormous amount of memory, storage, and computation, thus limiting their usability. Inspired by the recent tensor ring factorization, we introduce Tensor Ring Networks (TR-Nets), which significantly compress both the fully connected layers and the convolutional layers of deep neural networks. Our results show that our TR-Nets approach is able to compress LeNet-5 by 11$\times$ without losing accuracy, and can compress the state-of-the-art Wide ResNet by 243$\times$ with only 2.3% degradation in Cifar10 image classification. Overall, this compression scheme shows promise in scientific computing and deep learning, especially for emerging resource-constrained devices such as smartphones, wearables, and IoT devices.

Deep neural networks have made significant improvements in a variety of applications, including recommender systems [73,74], time series classification [75], nature language processing [76–78], and image and video recognition [79]. These accuracy improvements require developing deeper and deeper networks, evolving from AlexNet [6] (with $P = 61$ M parameters), VGG19 [7] ($P = 114$ M), and GoogleNet ($P = 11$ M) [80], to 32-layer ResNet ($P = 0.46$ M) [8,81], 28-layer WideResNet [82] ($P = 36.5$ M), and DenseNets [83]. Unfortunately, with each evolution in architecture comes a significant increase in the number of model parameters. On the other hand, many modern use cases of deep neural networks are for resource-constrained devices, such as mobile phones [84], wearables and IoT devices [85], etc. In these applications, storage, memory, and test runtime complexity are extremely limited in resources, and compression in these areas is thus essential.

After prior work [86] observed redundancy in trained neural networks, a useful area of research has been compression of network layer parameters (*e.g.,* [87–90]). While a vast majority of this research has been focused on the compression of fully connected layer parameters, the latest deep learning architectures are almost entirely dominated by convolutional layers. For example, while only 5% of AlexNet parameters are from convolutional layers, over 99% of Wide ResNet parameters are from convolutional layers. This necessitates new techniques that can factorize and compress the multi-dimensional tensor parameters of convolutional layers.

We propose compressing deep neural networks using *Tensor Ring (TR) factorizations* [58],which can be viewed as a generalization of a single Canonical Polyadic (CP) decomposition [18, 19, 91], with two extensions:

1. the outer vector products are generalized to matrix products, and
2. the first and last matrix are additionally multiplied along their outer edges, forming a "ring" structure.

Note that this is also a generalization of the *Tensor Train factorization* [21], which only includes the first extension. This is inspired by previous results in image processing [16], which demonstrate that this general factorization technique is extremely expressive, especially in preserving spatial features.

Specifically, we introduce Tensor Ring Nets (TRN), in which layers of a deep neural network are compressed using tensor ring factorization. For fully connected layers, we compress the weight matrix, and investigate different merge/reshape orders to minimize real-time computation and memory needs. For convolutional layers, we carefully compress the filter weights such that we do not distort the spatial properties of the mask. Since the mask dimensions are usually very small (5×5, 3×3 or even 1×1) we do not compress along these dimensions at all, and instead compress along the input and output channel dimensions.

To verify the expressive power of this formulation, we train several compressed networks. First, we train LeNet-300-100 and LeNet-5 [42] on the MNIST dataset, compressing LeNet-5 by 11× without degradation and achiving 99.31% accuracy, and compressing LeNet-300-100 by 13× with a degrading of only 0.14% (obtaining overall accuracy of 97.36%). Additionally, we examine the state-of-the-art 28-layer Wide-ResNet [82] on Cifar10, and find that TRN can be used to effectively compress the Wide-ResNet by 243× with only 2.3% decay in performance, obtaining 92.7% accuracy. The compression results demonstrates the capability of TRN to compress state-of-the-art deep learning models for new resources constrained applications.

4.2 Related Work

Past deep neural network compression techniques have largely applied to fully connected layers, which previously have dominated the number of parameters of a model. However, since modern models like ResNet and WideResNet are moving toward wider convolutional layers and omitting fully connected layers altogether, it is important to consider compression schemes that work on both fronts.

Many modern compression schemes focus on post-processing techniques, such as hashing [87] and quantization [92]. A strength of these methods is that they can be applied in addition to any other compression scheme, and are thus orthogonal to other methods. More similar to our work are novel representations like circulant projections [93] and truncated SVD representations [90].

Low-rank tensor approximation of deep neural networks has been widely investigated in the literature for effective model compression, low generative error, and fast prediction speed [84,94,95]. Tensor Networks (TNs) [24,25] have recently drawn considerable attention in multi-dimensional data representation [15,16,30,41], and deep learning [96–99].

One of the most popular methods of tensor factorization is the Tucker factorization [20], and has been shown to exhibit good performance in data representation [35,41, 100] and in compressing fully connected layers in deep neural networks [99]. In [84], a Tucker decomposition approach is applied to compress both fully connected layers and convolution layers.

Tensor train (TT) representation [21] is another example of TNs that factorizes a tensor into boundary two matrices and a set of 3rd order tensors, and has demonstrated its capability in data representation [29, 30, 101] and deep learning [36, 79]. In [16], the TT model is compared against TR for multi-dimensional data completion, showing that for the same intermediate rank, TR can be far more expressive than TT, motivating the generalization. We investigate TR for deep neural network compression.

4.3 Tensor Ring Nets

$\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ is a d mode tensor with ${}_{i=1}^{d} I_i$ degrees of freedom. A *tensor ring decomposition* factors such an $\mathcal{X}$ into d independent 3-mode tensors, $\mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(d)}$ such that each entry inside the tensor $\mathcal{X}$ is represented as

$$\mathcal{X}_{i_1,\cdots,i_d} = \underset{r_1,\cdots,r_d}{} \mathcal{U}^{(1)}_{r_d,i_1,r_1} \mathcal{U}^{(2)}_{r_1,i_2,r_2} \cdots \mathcal{U}^{(d)}_{r_{d-1},i_d,r_d}, \tag{4.1}$$

where $\mathcal{U}^{(i)} \in \mathbb{R}^{R \times I_i \times R}$, and R is the *tensor ring rank*. More generally, $\mathcal{U}^{(i)} \in \mathbb{R}^{R_i \times I_i \times R_{i+1}}$ and each R_i may not be the same. For simplicity, we assume $R_1 = \cdots = R_d = R$. Under this low-rank factorization, the number of free parameters is reduced to $R^2 \; {}_{i=1}^{d} I_i$ in the tensor ring factor form, which is significantly less than ${}_{i=1}^{d} I_i$ in $\mathcal{X}$.

For notational ease, let $\mathcal{U} = \{\mathcal{U}^{(1)}, \cdots, \mathcal{U}^{(d)}\}$, and define $\mathbf{decomp}(\mathcal{X}; R, d)$ as the operation to obtain d factors $\mathcal{U}^{(i)}$ with tensor ring rank R from $\mathcal{X}$, and $\mathbf{construct}(\mathcal{U})$ as the operation to obtain $\mathcal{X}$ from $\mathcal{U}$.

Additionally, for $1 \leq k < j \leq d$, define the **merge** operation as $\mathcal{M} = \mathbf{merge}(\mathcal{U}, k, j)$ such that $\mathcal{U}_k, \mathcal{U}_{k+1}, \cdots, \mathcal{U}_j$ are merged into one single tensor $\mathcal{M}$ of dimension $R \times I_k \times I_{k+1} \times \cdots \times I_j \times R$, and each entry in $\mathcal{M}$ is

$$\begin{aligned} \mathcal{M}_{r_{k-1},i_k,i_{k+1},\cdots,i_j,r_j} &= \\ &\underset{r_k,\cdots,r_{j-1}}{} \mathcal{U}^{(k)}_{r_{k-1},i_k,r_k} \mathcal{U}^{(k+1)}_{r_k,i_{k+1},r_{k+1}} \cdots \mathcal{U}^{(j)}_{r_{j-1},i_j,r_j}. \end{aligned} \tag{4.2}$$

Note that construct operator is the merge operation $\mathbf{merge}(\mathcal{U}, 1, d)$, which results in a tensor of shape $R \times I_1 \times I_2 \times \cdots \times I_d \times R$, followed by summing along mode 1 and mode $d + 2$, resulting in a tensor of shape $I_1 \times I_2 \times \cdots \times I_d$; e.g.

$$\mathbf{construct}(\mathcal{U}) = \overset{R}{\underset{r=1}{}} \mathbf{merge}(\mathcal{U}, 1, d)_{r,:,r}.$$

Merge ordering The computation complexity is measured in flops (counting additions and multiplications). The number of flops for a **construct** depends on the sequence of merging $\mathcal{U}^{(i)}, i = 1, \cdots, d$. (See figure 4.1). A detailed analysis of the two schemes is given in appendix C, resulting in the following conclusions.

Theorem 4.3.1 *Suppose $I_1 = \cdots = I_d \geq 2$ and $I = {}_{i=1}^{d} I_i$. Then*

1. *any merge order costs between $2R^3I$ and $4R^3I$ flops,*
2. *any merge order costs requires storing between R^2I and $2R^2I$ floats, and*
3. *if d is a power of 2, then a hierarchical merge order achieves the minimum flop count.*

Proof See appendix C. ■

Several interpretations can be made from these observations. First, though different merge orderings give different flop counts, the worst choice is at most 2x more expensive than the best choice. However, since we have to make some kind of choice, we note that since every merge order is a combination of hierarchical and sequential

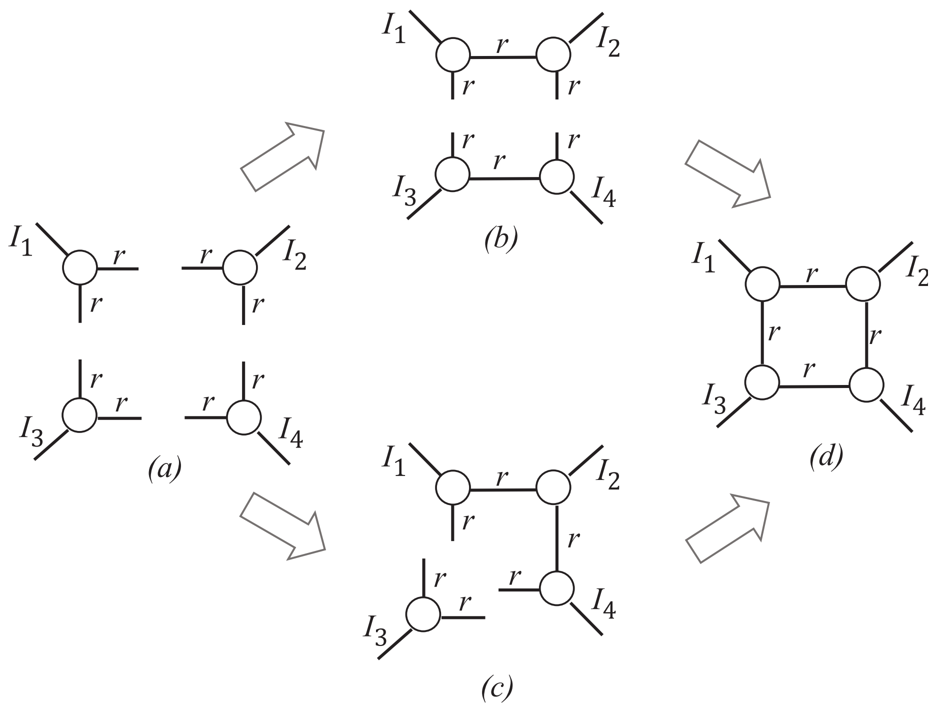


Fig. 4.1.: **Merge ordering.** A 4th order tensor is merged from its factored form, either hierarchically via (a)→(b)→(d), or sequentially via (a)→(c)→(d). Note that the computational complexity of forming (b) is $r^3(I_1I_2 + I_3I_4)$ and for (c) is $r^3(I_1I_2 + I_1I_2I_4)$, and (c) is generally more expensive (if $I_1 \approx I_2 \approx I_3 \approx I_4$). This is discussed in detail in Appendix C.

merges, striving toward a hierarchical merging is a good heuristic to minimize flop count. Thus, in our paper, we always use this strategy.

A *Tensor Ring Network (TRN)* is a tensor factorization of either fully connected layers (FCL) or convolutional layers (ConvL), trained via back propagation. If a pre-trained model is given, a good initialization can be obtained from the tensor ring decomposition of the layers in the pre-trained model.

4.3.1 Fully Connected Layer Compression

In feed-forward neural networks, an input feature vector $\mathbf{x} \in \mathbb{R}^I$ is mapped to an output feature vector $\mathbf{y} = \mathbf{A}\mathbf{x} \in \mathbb{R}^O$ via a fully connected layer $\mathbf{A} \in \mathbb{R}^{I \times O}$. Without loss of generality, $\mathbf{x}$, $\mathbf{A}$, and $\mathbf{y}$ can be reshaped into higher order tensors $\mathcal{X}$, $\mathcal{A}$, and $\mathcal{Y}$ with

$$\mathcal{Y}_{o_1,\ldots,o_{\hat{d}}} = \underset{i_1,\ldots,i_d}{\quad} \mathcal{A}_{i_1,\ldots,i_d,o_1,\ldots,o_{\hat{d}}} \mathcal{X}_{i_1,\ldots,i_d} \tag{4.3}$$

where d and $\hat{d}$ are the modes of $\mathcal{X}$ and $\mathcal{Y}$ respectively, and i_k's ad o_k's span from 1 to I_k and 1 to O_k respectively, and

$$\overset{d}{\underset{i=1}{\quad}} I_i = I, \qquad \overset{\hat{d}}{\underset{i=1}{\quad}} O_i = O.$$

To compress a feed-forward network, we decompose as $\mathcal{U} = \{\mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(d+\hat{d})}\} = \mathbf{decomp}(\mathcal{A}; R, d + \hat{d})$ and replace $\mathcal{A}$ with its decomposed version in (4.3). A tensor diagram for this operation is given in Figure 4.2, which shows how each multiplication is applied and the resulting dimensions.

Computational cost The computational cost again depends on the order of merging $\mathcal{X}$ and $\mathcal{U}$. Note that there is no need to fully construct the tensor $\mathcal{A}$, and a tensor representation of $\mathcal{A}$ is sufficient to obtain $\mathcal{Y}$ from $\mathcal{X}$. To reduce the computational

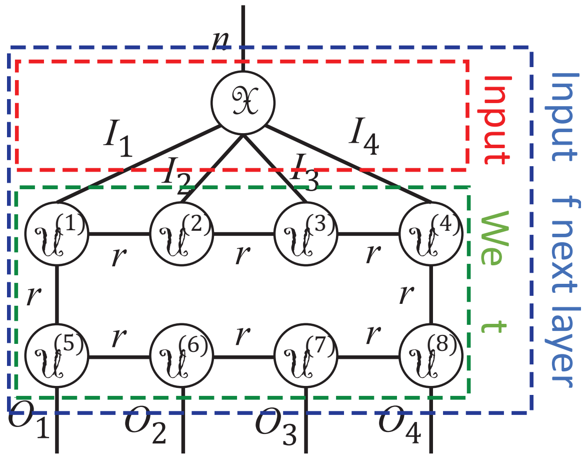


Fig. 4.2.: **Fully connected layer.** Tensor diagram of a fully connected TRN, divided into input and weights. The composite tensor is the input into the next layer.

cost, a *layer separation* approach is proposed by first using hierarchical merging to obtain

$$\begin{aligned} \mathcal{F}^{(1)} &= \mathbf{merge}(\mathcal{U}, 1, d) \in \mathbb{R}^{R \times I_1 \times \cdots \times I_d \times R} \\ \mathcal{F}^{(2)} &= \mathbf{merge}(\mathcal{U}, d+1, d+\hat{d}) \in \mathbb{R}^{R \times O_1 \times \cdots \times O_{\hat{d}} \times R}, \end{aligned} \tag{4.4}$$

which is upper bounded by $4R^3(I+O)$ flops. By replacing $\mathcal{A}$ in (4.3) with $\mathcal{F}^{(1)}$ and $\mathcal{F}^{(2)}$ and switching the order of summation, we obtain

$$\mathcal{Z}_{r_d, r_{d+\hat{d}}} = \quad\;_{i_1,\ldots,i_d} \mathcal{F}^{(1)}_{r_{d+\hat{d}}, i_1, \cdots, i_d, r_d} \mathcal{X}_{i_1,\ldots,i_d}, \tag{4.5}$$

$$\mathcal{Y}_{o_1,\ldots,o_{\hat{d}}} = \quad\;_{r_{d+\hat{d}}, r_d} \mathcal{Z}_{r_d, r_{d+\hat{d}}} \mathcal{F}^{(2)}_{r_d, o_1, \cdots, o_{\hat{d}}, r_{d+\hat{d}}}. \tag{4.6}$$

The summation (4.5) is equivalent to a feed-forward layer of shape $(I_1 \cdots I_d) \times R^2$, which takes $2R^2 I$ flops. Additionally, the summation over $r_{d+\hat{d}}$ and r_d is equivalent to another feed-forward layer of shape $R^2 \times (O_1 \cdots O_{\hat{d}})$, which takes $2R^2 O$ flops. Such analysis demonstrates that the *layer separation* approach to a FCL in a tensor ring net is equivalent to a low-rank matrix factorization to a fully-connected layer, thus reducing the computational complexity when R is relatively smaller than I and O.

Define P_{FC} and C_{FC} as the complexity saving in parameters and computation, respectively, for the tensor net decomposition over the typical fully connected layer forward propagation. Thus we have

$$P_{\text{FC}} = \frac{IO}{R^2 \quad\; {}_i^d I_i + \quad {}_j^{\hat{d}} O_j}. \tag{4.7}$$

and

$$C_{\text{FC}} \geq \frac{2BIO}{(4R^3 + 2BR^2)(I+O)}, \tag{4.8}$$

where B is the batch size of testing samples. Here, we see the compression benefit in computation; when B is very large, (4.8) converges to $IO/(R^2(I+O))$, which for large I, O and small R is significant. Additionally, though the expensive reshaping step grows cubically with R (as before), it does not grow with batch size; conversely, the multiplication itself (which grows linearly with batch size) is only quadratic in R. In the paper, the parameter is selected by picking small R and large d to achieve the optimal C since R needs to be small enough for computation saving.

4.3.2 Convolutional Layer Compression

In convolutional neural networks (CNNs), an input tensor $\mathcal{X} \in \mathbb{R}^{H\times W\times I}$ is convoluted with a 4th order kernel tensor $\mathcal{K} \in \mathbb{R}^{D\times D\times I\times O}$ and mapped to a 3rd order tensor $\mathcal{Y} \in \mathbb{R}^{H\times W\times O}$, as follows

$$\begin{aligned} \mathcal{Y}_{h,w,o} &= \sum_{d_1,d_2=1}^{D} \sum_{i=1}^{I} \mathcal{X}_{h\ ,w\ ,i} \mathcal{K}_{d_1,d_2,i,o}, \\ h\ &= (h-1)s + d_1 - p, \\ w\ &= (w-1)s + d_2 - p, \end{aligned} \tag{4.9}$$

where s is stride size, p is zero-padding size. Computed as in (4.9), the flop cost is $D^2 \cdot IO \cdot HW$. [1]

In TRN, tensor ring decomposition is applied onto the kernel tensor $\mathcal{K}$ and factorizes the 4th order tensor into four 3rd tensors. With the purpose to maintain the spatial information in the kernel tensor, we do not factorize the spatial dimension of $\mathcal{K}$ via merging the spatial dimension into one $4th$ order tensor $\mathcal{V}^{(1)}_{R_1,D_1,D_2,R_2}$, thus we have

$$\mathcal{K}_{d_1,d_2,i,o} = \sum_{r_1,r_2,r_3=1}^{R} \mathcal{V}_{r_1,d_1,d_2,r_2} \mathcal{U}_{r_2,i,r_3} \hat{\mathcal{U}}_{r_3,o,r_1}. \tag{4.10}$$

In the scenario when I and O are large, the tensors $\mathcal{U}$ and $\hat{\mathcal{U}}$ are further decomposed into $\mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(d)}$ and $\mathcal{U}^{(d+1)}, \ldots, \mathcal{U}^{(d+\hat{d})}$ respectively. (See also Figure 4.3.)

[1]For small filter sizes $D \quad \log(HW)$, as is often the case in deep neural networks for image processing, often direct multiplication to compute convolution is more efficient than using an FFT, which for this problem has order $IO(HW(\log(HW)))$ flops. Therefore we only consider direct multiplication as a baseline.

The kernel tensor factorization in (4.10) combined with the convolution operation in (4.9) can be equivalently solved in three steps:

$$\mathcal{P}_{h\,,w\,,r_2,r_3} = \sum_{i=1}^{I} \mathcal{X}_{h\,,w\,,i} \mathcal{U}^{(2)}_{r_2,i,r_3} \tag{4.11}$$

$$\mathcal{Q}_{h,w,r_3,r_1} = \sum_{d_1,d_2=1}^{D} \sum_{r_2}^{R} \mathcal{P}_{h\,,w\,,r_2,r_3} \mathcal{U}^{(1)}_{r_1,d_1,d_2,r_2} \tag{4.12}$$

$$\mathcal{Z}_{h,w,o} = \sum_{r_1,r_3} \mathcal{Q}_{h,w,r_3,r_1} \mathcal{U}^{(3)}_{r_3,o,r_1}. \tag{4.13}$$

where (4.11) is a tensor multiplication along one slice, with flop count HWR^2I, (4.12) is a 2-D convolution with flop count HWR^3D^2, and (4.13) is a tensor multiplication along 3 slices with flop count HWR^2O. This is also equivalent to a three-layer convolutional networks without non-linear transformations, where (4.11) is a convolutional layer from I feature maps to R^2 feature maps with a 1×1 patch, (4.12) contains R convolutional layers from R feature maps to R feature maps with a $D \times D$ patch, and (4.13) is a convolutional layer from R^2 feature maps to O feature maps with with a 1×1 patch. This is a common sub-architecture choice in other deep CNNs, like the inception module in GoogleNets [80], but without nonlinearities between 1×1 and $D \times D$ convolution layers.

Complexity: We employ the ratio between complexity in CNN layer and the complexity in tensor ring layer to quantify the capability of TRN in reducing computation (C_{conv}) and parameter (P_{conv}) costs,

$$\begin{aligned} P_{\text{conv}} &= \frac{D^2IO}{D^2R^2 + IR^2 + OR^2}, \\ C_{\text{conv}} &= \frac{IO \cdot D^2}{R^2I + R^3D^2 + R^2O}. \end{aligned} \tag{4.14}$$

If, additionally, the tensors $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(2)}$ are further decomposed to d and $\hat{d}$ tensors, respectively, then

$$\begin{aligned} P_{\text{conv}} &= \frac{D^2IO}{D^2R^2 + R^2(\sum_i^d I_i + \sum_j^{\hat{d}} O_j)}, \\ C_{\text{conv}} &= \frac{BIO \cdot D^2}{4R^3(I+O) + BR^2(I+O) + BR^3D^2}. \end{aligned} \tag{4.15}$$

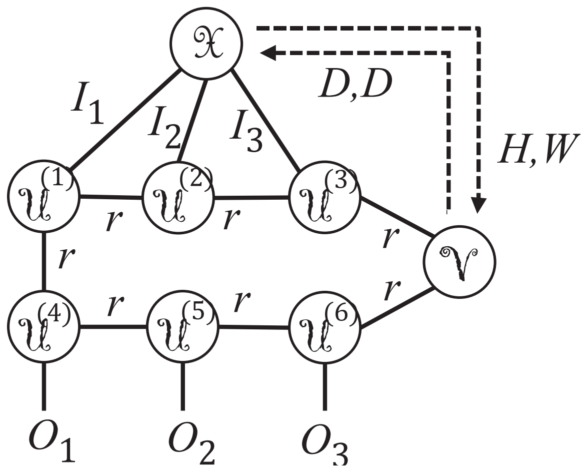


Fig. 4.3.: Tensor ring compressed convolutional layer.

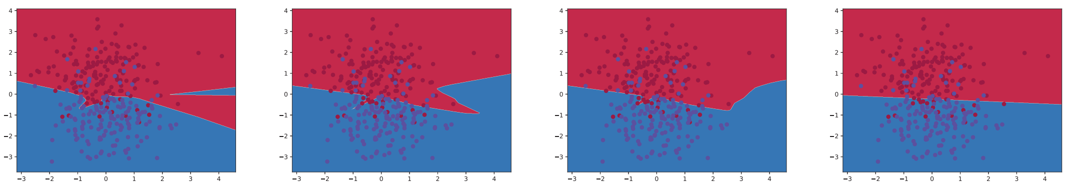


Fig. 4.4.: Decision boundary for the mixture of two gaussian distribution in 2D. From left to right are standard fully connected layer, tensor ring nets with rank 3, tensor ring nets with rank 2, and tensor ring nets with rank 1.

Note that in the second scenario, we have a further compression in storage requirements, but lose gain in computational complexity, which is a design tradeoff. In our experiments, we further factorize $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(3)}$ in to higher order tensors in order to achieve our gain in model compression.

Initialization In general nonconvex optimization (especially for deep learning), the choice of initial variables can dramatically effect the quality of the model training. In particular, we have found that initializing each parameter randomly from a Gaussian distribution is effective, with a carefully chosen variance. If we initialize all tensor factors as drawn i.i.d. from $\mathcal{N}(0, \sigma^2)$, then after merging d factors the merged tensor elements will have mean 0 and variance $R^d\sigma^{2d}$ (See appendix D). By picking $\sigma = \left(\frac{2}{N}\right)^{1/d}\frac{1}{\sqrt{R}}$, where N is the amount of parameters in the uncompressed layer, the merged tensor will have mean 0, variance $\sqrt{2/N}$, and in the limit will also be Gaussian. Since this latter distribution works well in training the uncompressed models, choosing this value of σ for initialization is well-motivated, and observed to be necessary for faster convergence.

4.4 Results

To evaluate the performance of TRN, we first evaluate the decision boundary of TRN on the synthetic data, including 2D Gaussian Mixture data where the data are generated from two random Gaussian distribution with mean $\mu_1 = [0, 1]$ and $\mu_2 = [0, -1]$, and standard deviation $\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $\Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. 150 samples drawn randomly from each class are used as the training data. The classifier is a two layer neural networks that takes relu as activation function. Without loss of generality, the hidden layer has a dimensionality 256, which is reshaped into $[16, 16]$ in the tensor ring nets. As shown in Fig 4.4, in the given set-up, the general fully connected layer is able to cut the space into two separate domain, where the decision boundary is smooth and flexible in its geometry. In contrast, tensor ring nets try to separate the two class with sharping lines rather than smooth curves. As the tensor ring rank decreases, the counter of the decision boundary becomes less free in shape. When the tensor ring rank becomes 1, the tensor ring factorized layer is equivalent to a rank-1 CP factorized layer, and the decision boundary becomes linear. In other words, low-rank tensor ring nets seek a sharper decision boundary with less freedom in the high dimensional space.

We now evaluate the effectiveness of TRN-based compression on several well-studied deep neural networks and datasets: LeNet-300-100 and LeNet-5 on MNIST, and ResNet and WideResNet on Cifar10 and Cifar100. These networks are trained using Tensorflow [102]. All the experiments on LeNet are implemented on Nvidia GTX 1070 GPUs, and all the experiments for ResNet and WideResNet are implemented on Nvidia GTX Titan X GPUs. In all cases, the same tensor ring rank r is used in the networks, and all the networks are trained from randomly initialization using the the proposed initialization method. Overall, we show that this compression scheme can give significant compression gains for small accuracy loss, and even negligible compression gains for no accuracy loss.

4.4.1 Fully connected layer compression

The goal of compressing the LeNet-300-100 network is to assess the effectiveness of compressing fully connected layers using TRNs; as the name suggests, LeNet-300-

Table 4.1.: **Fully connected compression.** Dimensions of the three-fully-connected layers in the uncompressed (left) and TRN-compressed (right) models. The computational complexity includes tensor product merging ($O(r^3)$) and feed-froward multiplication ($O(r^2)$).

	Uncompressed dims.			TRN dimensions		
layer	shape	# params	flops	shape of composite tensor	# params	flops
fc1	784×300	235K	470K	$(4 \times 7 \times 4 \times 7) \times (3 \times 4 \times 5 \times 5)$	$39r^2$	$1177r^3 + 1084r^2$
fc2	300×100	30K	60K	$(3 \times 4 \times 5 \times 5) \times (4 \times 5 \times 5)$	$31r^2$	$457r^3 + 400r^2$
fc3	100×10	1K	2K	$(4 \times 5 \times 5) \times (2 \times 5)$	$21r^2$	$127r^3 + 107r^2$
Total	-	266K	532K	-	$91r^2$	$1761r^3 + 1591r^2$

100 contains two hidden fully connected layers with output dimension 300 and 100, and an output layer with dimension 10 (= # classes). Table 4.1 gives the parameter settings for LeNet-300-100, both in its original form (uncompressed) and in its tensor factored form. A compression rate greater than 1 is achieved for all $r \leq 54$, and a reduction in computational complexity for all $r \leq 6$; both are typical choices.

Table 4.2 shows the performance results on MNIST classification for the original model (as reported in their paper), and compressed models using both matrix factorization and TRNs. For a 0.14% accuracy loss, TRN can compress up to 13×, and for no accuracy loss, can compress 1.2×. Note also that matrix factorization, at 16× compression, performs worse than TRN at 117× compression, suggesting that the high order structure is helpful. Note also that low rank Tucker approximation in [84] is equivalent to low rank matrix approximation when compressing fully connected layer.

4.4.2 Convolutional layer compression

We now investigate compression of convolutional layers in a small network. LeNet-5 is a (relatively small) convolutional neural networks with 2 convolution layers, fol-

Table 4.2.: **Fully connected results.** LeNet-300-100 on MNIST datase, trained to 40 epochs, using a minibatch size 50. Trained from random weight initialization. ADAM [103] is used for optimization. Testing time is per 10000 samples. CR = Compression ratio. LR = Learning rate.

Method	Params	CR	Err %	Test (s)	Train (s/epoch)	LR
LeNet-300-100 [42]	266K	1×	2.50	0.011 ± 0.002	3.5 ± 1.0	$2e^{-4}$
M-FC [84, 90]($r = 10$)	16.4K	16.3×	3.91	0.016 ± 0.010	6.4 ± 1.2	$1e^{-4}$
M-FC ($r = 20$)	31.2K	5.3×	3.0	0.014 ± 0.010	5.2 ± 1.2	$1e^{-4}$
M-FC ($r = 50$)	75.7K	3.5×	2.62	0.021 ± 0.012	8.1 ± 1.2	$1e^{-4}$
TRN ($r = 3$)	0.8K	325.5×	8.53	0.015 ± 0.007	7.9 ± 1.4	$1e^{-3}$
TRN ($r = 5$)	2.3K	117.2×	3.75	0.015 ± 0.007	7.8 ± 1.4	$2e^{-3}$
TRN ($r = 15$)	20.5K	13.0×	2.64	0.015 ± 0.007	8.1 ± 1.4	$5e^{-4}$
TRN ($r = 50$)	227.5K	1.2×	**2.31**	0.022 ± 0.008	11.1 ± 1.4	$5e^{-5}$

lowed by 2 fully connected layers, which achieves 0.79% error rate on MNIST. The dimensions before and after compression are given in Table 4.3. In this wider network we see a much greater potential for compression, with positive compression rate whenever $r \leq 57$. However, the reduction in complexity is more limited, and only occurs when $r \leq 4$.

However, the performance on this experiment is still positive. By setting $r = 20$, we compress LeNet-5 by 11× and a lower error rate than the original model as well as the Tucker factorization approach. If we also require a reduction in flop count, we incur an error of 2.24%, which is still quite reasonable in many real applications.

4.4.3 ResNet and Wide ResNet Compression

Finally, we evaluate the performance of tensor ring nets (TRN) on the Cifar10 and Cifar100 image classification tasks [104]. Here, the input images are colored, of

Table 4.3.: **Small convolution compression.** Dimensions of LeNet-5 layers in its original form (left) and TRN-compressed (right). The computational complexity includes tensor product merging and convolution operation in (4.12) of $O(r^3)$, and convolution in (4.11) (4.13) of $O(r^2)$.

	Uncompressed dims.			TRN dimensions		
layer	shape	# params	flops	shape	# params	flops
conv1	$5 \times 5 \times 1 \times 20$	0.5K	784K	$5 \times 5 \times 1 \times (4 \times 5)$	$19r^2$	$33408r^2 + 39245r^3$
conv2	$5 \times 5 \times 20 \times 50$	25K	5000K	$5 \times 5 \times (4 \times 5) \times (5 \times 10)$	$34r^2$	$17840r^2 + 5095r^3$
fc1	1250×320	400K	800K	$(5 \times 5 \times 5 \times 10) \times (5 \times 8 \times 8)$	$46r^2$	$1570r^2 + 1685r^3$
fc2	320×10	3K	6K	$(5 \times 8 \times 8) \times 10$	$31r^2$	$330r^2 + 360r^3$
Total	-	429K	6590K	-	$130r^2$	$53148r^2 + 46385r^3$

Table 4.4.: **Small convolution results.** LeNet-5 on MNIST dataset, trained to 20 epochs, using a minibatch size 128. ADAM [103] is used for optimization. Testing time is per 10000 samples. CR = Compression ratio. LR = Learning rate.

Method	Params	CR	Err %	Test (s)	Train (s/epoch)	LR
LeNet-5 [42]	429K	1×	0.79	0.038 ± 0.027	1.6 ± 1.9	$5e^{-4}$
Tucker [84]	189K	2×	0.85	0.066 ± 0.025	7.7 ± 3	$5e^{-4}$
TRN ($r = 3$)	1.5K	286×	2.24	0.058 ± 0.026	8.3 ± 4.5	$5e^{-4}$
TRN ($r = 5$)	3.6K	120×	1.64	0.072 ± 0.039	10.6 ± 7.1	$5e^{-4}$
TRN ($r = 10$)	11.0K	39×	1.39	0.080 ± 0.025	15.6 ± 4.6	$2e^{-4}$
TRN ($r = 15$)	23.4K	18×	0.81	0.039 ± 0.019	20.1 ± 16.0	$2e^{-4}$
TRN ($r = 20$)	40.7K	11×	**0.69**	0.052 ± 0.028	27.8 ± 7.4	$1e^{-5}$

size $32 \times 32 \times 3$, belonging to 10 and 100 object classes respectively. Overall there are 50000 images for training and 10000 images for testing.

Table 4.5 gives the dimensions of ResNet before and after compression. A similar reshaping scheme is used for WideResNet. Note that for ResNet, we have compression

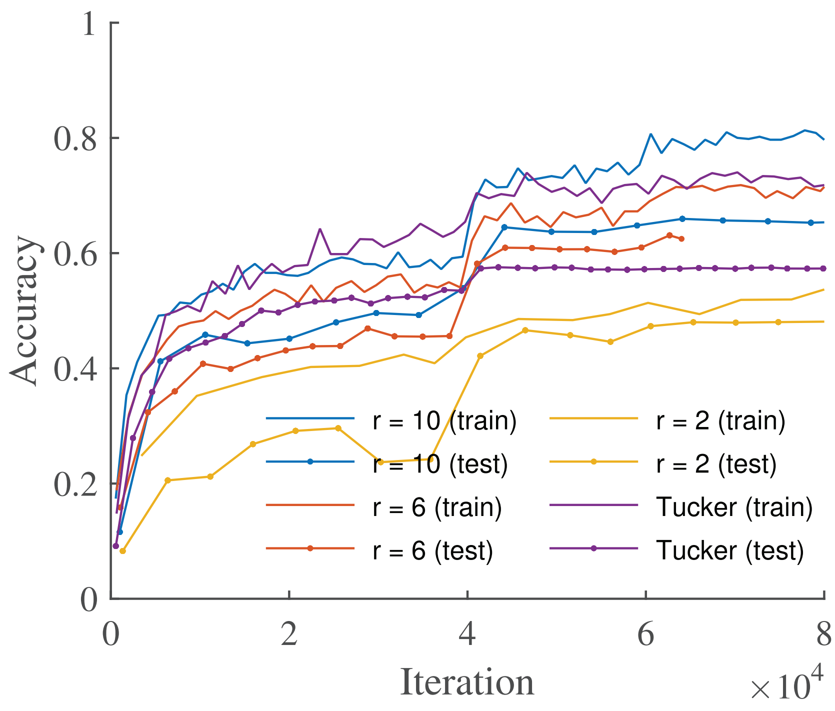


Fig. 4.5.: **Evolution.** Evolution of training compressed 32 layer ResNet on Cifar100, using TRNs with different values of r and the Tucker factorization method.

gain for any $r \leq 22$; for WideResNet this bound is closer to $r \leq 150$, suggesting high compression potential.

The results are given in Table 4.6 demonstrates that TRNs are able to significantly compress both ResNet and WideResNet for both tasks. Picking $r = 10$ for TRN on ResNet gives the same compression ratio as the Tucker compression method [84], but with almost 3% performance lift on Cifar10 and almost 10% lift on Cifar 100. Compared to the uncompressed model, we see only a 2% performance degradation on both datasets.

The compression of WideResNet is even more successful, suggesting that TRNs are well-suited for these extremely overparametrized models. At a 243$\times$ compression TRNs give a better performance on Cifar10 than uncompressed ResNet (but with fewer parameters) and only a 2% decay from the uncompressed WideResNet. For

Table 4.5.: **Large convolution compression.** Dimensions of 32 layer ResNes on Cifar10 dataset. Each ResBlock(p,I,O) includes a sequence: input $\rightarrow$ Batch Normalization $\rightarrow$ ReLU $\rightarrow$ $p \times p \times I \times O$ convolution layer $\rightarrow$ Batch Normalization $\rightarrow$ ReLU $\rightarrow$ $p \times p \times O \times O$ convolution layer. The input of length I is inserted once at the beginning and again at the end of each unit. See [8] for more details.

	Uncompressed dims.		TRN dimensions	
layer	shape	# params	shape of composite tensor	# params
conv1	$3 \times 3 \times 3 \times 16$	432	$9 \times 3 \times (4 \times 2 \times 2)$	$20r^2$
unit1	ResBlock(3, 16, 16)	4608	$9 \times (4 \times 2 \times 2) \times (4 \times 2 \times 2)$	$50r^2$
	ResBlock(3, 16, 16) $\times$ 4	18432	$9 \times (4 \times 2 \times 2) \times (4 \times 2 \times 2)$	$200r^2$
unit2	ResBlock(3, 16, 32)	13824	$9 \times (4 \times 2 \times 2) \times (4 \times 4 \times 2)$	$56r^2$
	ResBlock(3, 32, 32) $\times$ 4	73728	$9 \times (4 \times 4 \times 2) \times (4 \times 4 \times 2)$	$232r^2$
unit3	ResBlock(3, 32, 64)	55296	$9 \times (4 \times 4 \times 2) \times (4 \times 4 \times 4)$	$64r^2$
	ResBlock(3, 64, 64) $\times$ 4	294912	$9 \times (4 \times 4 \times 4) \times (4 \times 4 \times 4)$	$264r^2$
fc1	64×10	650	$(4 \times 4 \times 4) \times 10$	$22r^2$
Total	-	0.46M	-	$908r^2$

Cifar100, this decay increases to 8%, but again TRN of WideResNet achieves lower error than uncompressed ResNet, with overall fewer parameters. Compared against the Tucker compression method [84], at 5$\times$ compression rate TRNs incur only 2-3% performance degradation on both datasets, while Tucker incurs 5% and 11% performance degradation. The compressibility is even more significant for WideResNet, where to achieve the same performance as Tucker [84] at 5$\times$ compression, TRNs can compress up to 243$\times$ on Cifar10 and 286$\times$ on Cifar100. The tradeoff is runtime; we observe the Tucker model trains at about 2 or 3 times faster than TRNs for the WideResNet compression. However, for memory-constrained devices, this tradeoff may still be desirable.

Table 4.6.: **Large convolution results.** 32-layer ResNet (first 5 rows) and 28-layer Wide-ResNet (last 4 rows) on Cifar10 dataset and Cifar100 dataset, trained to 200 epochs, using a minibatch size of 128. The model is trained using SGD with momentum 0.9 and a decaying learning rate. CR = Compression ratio.

	Cifar10			Cifar100		
Method	Params	CR	Err %	Params	CR	Err %
ResNet(RN)-32L	0.46M	1×	7.50 [105]	0.47M	1×	31.9 [105]
Tucker-RN [84]	0.09M	5×	12.3	0.094M	5×	42.2
TT-RN($r = 13$) [36, 106]	0.096M	4.8×	11.7	0.102M	4.6×	37.1
TRN-RN ($r = 2$)	0.004M	115×	22.2	0.012M	39×	51.3
TRN-RN ($r = 6$)	0.03M	15×	19.2	0.041M	12×	36.6
TRN-RN ($r = 10$)	0.09M	5×	9.4	0.097M	5×	33.3
WideResNet(WRL)-28L	36.2M	1×	5.0 [105]	36.3M	1×	21.7 [105]
Tucker-WRN [84]	6.7M	5×	7.8	6.7M	5×	30.8
TT-RN($r = 13$) [36, 106]	0.18M	201×	8.4	0.235M	154×	31.9
TRN-WRN ($r = 2$)	0.03M	1217×	16.3	0.087M	417×	43.9
TRN-WRN ($r = 6$)	0.07M	521×	9.7	0.126M	286×	30.3
TRN-WRN ($r = 10$)	0.15M	243×	**7.3**	0.21M	173×	28.3
TRN-WRN(r=15)	0.30M	122×	**7.0**	0.36M	100×	25.6

Evolution Figure 4.5 shows the train and test errors during training of compressed ResNet on the Cifar100 classification task, for various choices of r and also compared against Tucker tensor factorization. In particular, we note that the generalization gap (between train and test error) is particularly high for the Tucker tensor factorization method, while for TRNs (especially for smaller values of r) it is much smaller. For $r = 10$, both the generalization error and final train and test errors improve upon the Tucker method, suggesting that TRNs are easier to train.

4.5 Conclusion

We have introduced a tensor ring factorization approach to compress deep neural networks for resource-limited devices. This is inspired by previous work that has shown tensor rings to have high representative power in image completion tasks. Our results show significant compressibility using this technique, with little or no hit in performance on benchmark image classification tasks.

One area for future work is the reduction of computational complexity. Because of the repeated reshaping needs in both fully connected and convolutional layers, there is computational overhead, especially when r is moderately large. This tradeoff is reasonable, considering our considerable compressibility gains, and is appropriate in memory-limited applications, especially if training is offloaded to the cloud. Additionally, we believe that the actual wall-clock-time will decrease as tensor-specific hardware and low-level routines continue to develop–we observe, for example, that numpy's `dot` function is considerably more optimized than Tensorflow's `tensordot`. Overall, we believe this is a promising compression scheme and can open doors to using deep learning in a much more ubiquitous computing environment.

5. SUMMARY

Multi-dimensional data brings the challenges to the popular machine learning and deep learning algorithm from the perspective of computational complexity and storage complexity, while developing efficient tensor algorithm is able to alleviate the multi-dimensional data problem and achieves a better trade-off among computation, storage and performance. In this thesis, we demonstrate the proposed tensor train subspace enables efficient discriminative features extraction and enabled better dimensionality reduction via either TTPCA and TTNPE. As an extension of popular known tensor train representation, tensor ring representation provides improved capability in data compression and exhibits superior performance in missing data completion. The novel tensor ring representation is also investigated for its application in deep neural networks compression, which again demonstrates the state-of-the-arts capability in model compression.

REFERENCES

REFERENCES

[1] G. Adomavicius and A. Tuzhilin, “Context-aware recommender systems,” in *Recommender systems handbook.* Springer, 2015, pp. 191–226.

[2] E. Frolov and I. Oseledets, “Tensor methods and recommender systems,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 3, 2017.

[3] F. Cong, Q.-H. Lin, L.-D. Kuang, X.-F. Gong, P. Astikainen, and T. Ristaniemi, “Tensor decomposition of eeg signals: a brief review,” *Journal of neuroscience methods*, vol. 248, pp. 59–69, 2015.

[4] X. Song, L. Meng, Q. Shi, and H. Lu, “Learning tensor-based features for whole-brain fmri classification,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention.* Springer, 2015, pp. 613–620.

[5] “Giving cars the power to see, think, and learn,” https://www.nvidia.com/en-us/self-driving-cars/.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[8] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] R. Girshick, “Fast r-cnn,” *arXiv preprint arXiv:1504.08083*, 2015.

[10] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, 2015, pp. 91–99.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[12] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[13] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” *CoRR abs/1609.03499*, 2016.

[14] W. Wang, V. Aggarwal, and S. Aeron, “Principal component analysis with tensor train subspace,” *arXiv preprint arXiv:1803.05026*, 2018.

[15] ——, “Tensor train neighborhood preserving embedding,” *IEEE Transactions on Signal Processing*, vol. 66, no. 10, pp. 2724–2732, May 2018.

[16] ——, “Efficient low rank tensor ring completion,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017, pp. 5698–5706.

[17] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, “Wide compression: Tensor ring nets,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

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

[19] F. L. Hitchcock, “The expression of a tensor or a polyadic as a sum of products,” *Studies in Applied Mathematics*, vol. 6, no. 1-4, pp. 164–189, 1927.

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

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

[22] S. Holtz, T. Rohwedder, and R. Schneider, “On manifolds of tensors of fixed tt-rank,” *Numerische Mathematik*, vol. 120, no. 4, pp. 701–731, 2012.

[23] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, “Tensor ring decomposition,” *arXiv preprint arXiv:1606.05535*, 2016.

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

[25] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, D. P. Mandic *et al.*, “Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives,” *Foundations and Trends* R *in Machine Learning*, vol. 9, no. 6, pp. 431–673, 2017.

[26] I. Jolliffe, *Principal component analysis.* Wiley Online Library, 2002.

[27] C. M. Bishop, “Pattern recognition,” *Machine Learning*, vol. 128, 2006.

[28] M. Belkin and P. Niyogi, “Laplacian eigenmaps for dimensionality reduction and data representation,” *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[29] H. N. Phien, H. D. Tuan, J. A. Bengua, and M. N. Do, “Efficient tensor completion: Low-rank tensor train,” *arXiv preprint arXiv:1601.01083*, 2016.

[30] W. Wang, V. Aggarwal, and S. Aeron, “Tensor completion by alternating minimization under the tensor train (TT) model,” *arXiv preprint arXiv:1609.05587*, 2016.

[31] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[32] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "Multilinear principal component analysis of tensor objects for recognition," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 2. IEEE, 2006, pp. 776–779.

[33] M. A. O. Vasilescu and D. Terzopoulos, "Multilinear subspace analysis of image ensembles," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 2. IEEE, 2003, pp. II–93.

[34] J. Wu, S. Qiu, R. Zeng, Y. Kong, L. Senhadji, and H. Shu, "Multilinear principal component analysis network for tensor object classification," *IEEE Access*, vol. 5, pp. 3322–3331, 2017.

[35] M. Ashraphijuo, V. Aggarwal, and X. Wang, "Deterministic and probabilistic conditions for finite completability of low rank tensor," *arXiv preprint arXiv:1612.01597*, 2016.

[36] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 442–450.

[37] A. Tjandra, S. Sakti, and S. Nakamura, "Compressing recurrent neural network with tensor train," in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 4451–4458.

[38] W. Hackbusch, *Tensor spaces and numerical tensor calculus*. Springer Science & Business Media, 2012, vol. 42.

[39] X. He, D. Cai, S. Yan, and H.-J. Zhang, "Neighborhood preserving embedding," in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. 2. IEEE, 2005, pp. 1208–1213.

[40] X. He, D. Cai, and P. Niyogi, "Tensor subspace analysis," in *Advances in neural information processing systems*, 2005, pp. 499–506.

[41] G. Dai and D.-Y. Yeung, "Tensor embedding methods," in *AAAI*, vol. 6, 2006, pp. 330–335.

[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[43] "Weizmann facebase," ftp://ftp.idc.ac.il/pub/users/cs/yael/Facebase/.

[44] R. Vidal, Y. Ma, and S. Sastry, "Generalized principal component analysis (gpca)," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 12, pp. 1945–1959, 2005.

[45] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[46] D. L. Donoho, “De-noising by soft-thresholding,” *IEEE transactions on information theory*, vol. 41, no. 3, pp. 613–627, 1995.

[47] B. Jiang, S. Ma, and S. Zhang, “Tensor principal component analysis via convex optimization,” *Mathematical Programming*, vol. 150, no. 2, pp. 423–457, 2015.

[48] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, “A survey of multilinear subspace learning for tensor data,” *Pattern Recognition*, vol. 44, no. 7, pp. 1540–1551, 2011.

[49] S. Yan, D. Xu, Q. Yang, L. Zhang, X. Tang, and H.-J. Zhang, “Multilinear discriminant analysis for face recognition,” *IEEE Transactions on Image Processing*, vol. 16, no. 1, pp. 212–220, 2007.

[50] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, “Uncorrelated multilinear principal component analysis for unsupervised multilinear subspace learning,” *IEEE Transactions on Neural Networks*, vol. 20, no. 11, pp. 1820–1836, 2009.

[51] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman, “From few to many: Illumination cone models for face recognition under variable lighting and pose,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 23, no. 6, pp. 643–660, 2001.

[52] K.-C. Lee, J. Ho, and D. J. Kriegman, “Acquiring linear subspaces for face recognition under variable lighting,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 5, pp. 684–698, 2005.

[53] R. Basri and D. W. Jacobs, “Lambertian reflectance and linear subspaces,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 2, pp. 218–233, 2003.

[54] A. Beck, “On the convergence of alternating minimization for convex programming with applications to iteratively reweighted least squares and decomposition schemes,” *SIAM Journal on Optimization*, vol. 25, no. 1, pp. 185–209, 2015.

[55] Z. Wen and W. Yin, “A feasible method for optimization with orthogonality constraints,” *Mathematical Programming*, vol. 142, no. 1-2, pp. 397–434, 2013.

[56] M. S. Moslehian, “Ky fan inequalities,” *Linear and Multilinear Algebra*, vol. 60, no. 11-12, pp. 1313–1325, 2012.

[57] N. S. Altman, “An introduction to kernel and nearest-neighbor nonparametric regression,” *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[58] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, “Tensor ring decomposition,” *arXiv preprint arXiv:1606.05535*, 2016.

[59] https://www.mathworks.com/matlabcentral/fileexchange/43627-download-daily-data-from-google-and-yahoo--finance.

[60] S. Yadav, R. Sinha, and P. Bora, “An efficient svd shrinkage for rank estimation,” *IEEE Signal Processing Letters*, vol. 22, no. 12, pp. 2406–2410, 2015.

[61] S. Ubaru and Y. Saad, "Fast methods for estimating the numerical rank of large matrices," in *International Conference on Machine Learning*, 2016, pp. 468–477.

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

[63] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[64] M. Vasilescu and D. Terzopoulos, "Multilinear image analysis for face recognition," *Proceedings of the International Conference on Pattern Recognition ICPR 2002*, vol. 2, pp. 511–514, 2002, quebec City, Canada.

[65] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," *Annals of Physics*, vol. 349, pp. 117–158, 2014.

[66] P. Jain, P. Netrapalli, and S. Sanghavi, "Low-rank matrix completion using alternating minimization," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing.* ACM, 2013, pp. 665–674.

[67] M. Hardt, "On the provable convergence of alternating minimization for matrix completion," *CoRR*, vol. abs/1312.0925, 2013. [Online]. Available: http://arxiv.org/abs/1312.0925

[68] L. Grasedyck, M. Kluge, and S. Kramer, "Variants of alternating least squares tensor completion in the tensor train format," *SIAM Journal on Scientific Computing*, vol. 37, no. 5, pp. A2424–A2450, 2015.

[69] A. Cichocki, "Tensor networks for big data analytics and large-scale optimization problems," *arXiv preprint arXiv:1407.3124*, 2014.

[70] "Albert Einstein image," http://orig03.deviantart.net/7d28/f/2012/361/1/6/albert_einstein_by_zuzahin-d5pcbug.jpg.

[71] A. S. Georghiades and P. N. Belhumeur, "Illumination cone models for faces recognition under variable lighting," in *Proceedings of CVPR*, 1998.

[72] "Pistol shot recorded at 73,000 frames per second," https://youtu.be/7y9apnbI6GA, published by Discovery on 2015-08-15.

[73] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in neural information processing systems*, 2013, pp. 2643–2651.

[74] S. Zhang, L. Yao, and A. Sun, "Deep learning based recommender system: A survey and new perspectives," *arXiv preprint arXiv:1707.07435*, 2017.

[75] W. Wang, C. Chen, W. Wang, P. Rai, and L. Carin, "Earliness-aware deep convolutional networks for early time series classification," *arXiv preprint arXiv:1611.04578*, 2016.

[76] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*. ACM, 2008, pp. 160–167.

[77] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[78] W. Wang, Z. Gan, W. Wang, D. Shen, J. Huang, W. Ping, S. Satheesh, and L. Carin, "Topic compositional neural language model," *arXiv preprint arXiv:1712.09783*, 2017.

[79] Y. Yang, D. Krompass, and V. Tresp, "Tensor-train recurrent neural networks for video classification," *arXiv preprint arXiv:1707.01786*, 2017.

[80] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[81] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.

[82] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[83] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[84] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[85] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and Internet-of-things devices," in *Proceedings of the 2015 International Workshop on Internet of Things towards Applications*. ACM, 2015, pp. 7–12.

[86] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in neural information processing systems*, 2014, pp. 2654–2662.

[87] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[88] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[89] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[90] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[91] M. Ashraphijuo, X. Wang, and V. Aggarwal, "An approximation of the cp-rank of a partially sampled tensor," in *55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2017.

[92] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[93] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2857–2865.

[94] J. Sokolić, R. Giryes, G. Sapiro, and M. R. Rodrigues, "Generalization error of deep neural networks: Role of classification margin and data structure," in *Sampling Theory and Applications (SampTA), 2017 International Conference on.* IEEE, 2017, pp. 147–151.

[95] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," in *International Conference on Learning Representations*, 2015.

[96] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: A tensor analysis," in *Conference on Learning Theory*, 2016, pp. 698–728.

[97] N. Cohen and A. Shashua, "Convolutional rectifier networks as generalized tensor decompositions," in *International Conference on Machine Learning*, 2016, pp. 955–963.

[98] N. Cohen, O. Sharir, and A. Shashua, "Deep SimNets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4782–4791.

[99] J. Kossaifi, Z. C. Lipton, A. Khanna, T. Furlanello, and A. Anandkumar, "Tensor regression networks," *arXiv preprint arXiv:1707.08308*, 2017.

[100] M. Ashraphijuo, V. Aggarwal, and X. Wang, "A characterization of sampling patterns for low-tucker-rank tensor completion problem," in *Information Theory (ISIT), 2017 IEEE International Symposium on.* IEEE, 2017, pp. 531–535.

[101] M. Ashraphijuo, X. Wang, and V. Aggarwal, "Rank determination for low-rank data completion," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 3422–3450, 2017.

[102] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[103] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[104] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[105] "Tensorflow Resnet," https://github.com/tensorflow/models/tree/master/research/resnet.

[106] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, "Ultimate tensorization: compressing convolutional and fc layers alike," *arXiv preprint arXiv:1611.03214*, 2016.

[107] https://socratic.org/questions/if-x-and-y-are-independent-random-variables-what-is-var-xy.

APPENDICES

A. TENSOR NETWROK MERGING FOR COMPUTING $\mathcal{A}$

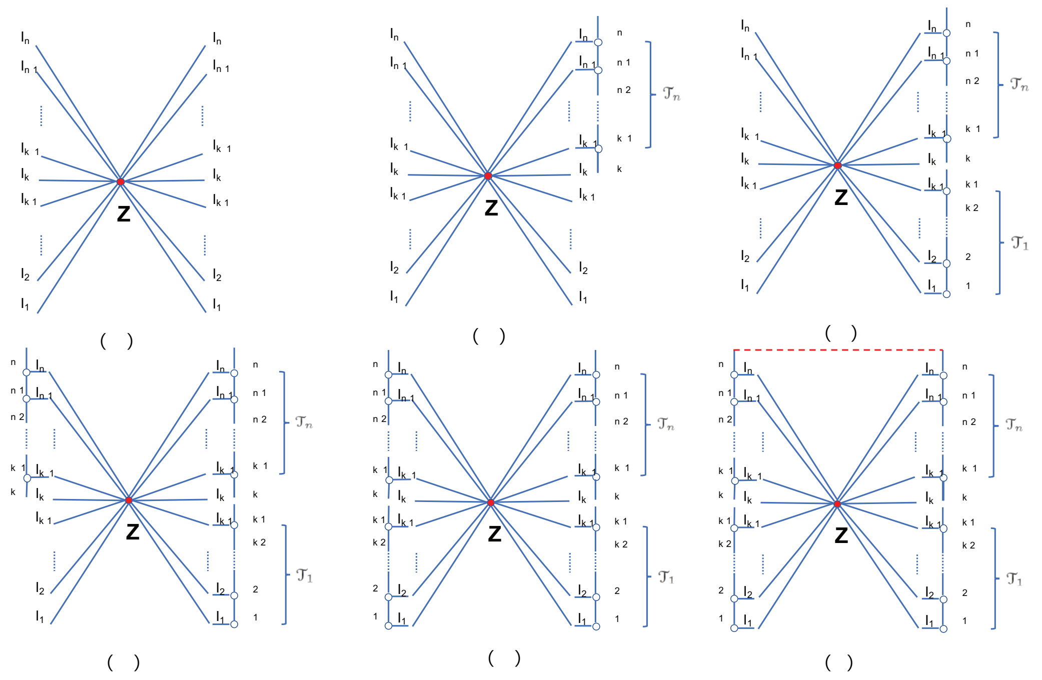


Fig. A.1.: Tensor network merging operation to compute $\mathcal{A}$. (a) Tensor $\mathcal{Z}$ (b) Tensor $\mathcal{A}_b$ (c) Tensor $\mathcal{A}_c$ (d) Tensor $\mathcal{A}_d$ (e) Tensor $\mathcal{A}_e$ (f) Tensor $\mathcal{A}$

Explanation of Tensor Network Merging Operation to compute $\mathcal{A}$ using (2.22). Figure A.1 shows the steps to compute $\mathcal{A}$. A tensor $\mathcal{Z} \in \mathbb{R}^{I_1\times\cdots\times I_n\times I_1\times\cdots\times I_n}$ in Fig A.1 (a) merged with tensor $\mathcal{T}_n \in \mathbb{R}^{R_k\times I_k\times\cdots\times I_n\times R_n}$ gives

$$\mathcal{Z} \times^{2,\cdots,n-k+1}_{n+k+1,\cdots,2n} \mathcal{T}_n = \mathcal{A}_b \in \mathbb{R}^{I_1\times\cdots\times I_n\times I_1\times\cdots\times I_k\times R_k\times R_n}, \tag{A.1}$$

as in Fig A.1 (b), where the merged dimensions $I_{k+1}\times\cdots\times I_n$ are replaced by the non-merged dimension $R_k \times R_n$. Following the same logic, we have

$$\mathcal{A}_b \times^{1,\cdots,k-1}_{n+1,\cdots,n_k-1} \mathcal{T}_1 = \mathcal{A}_c \in \mathbb{R}^{I_1\times\cdots\times I_n\times R_{k-1}\times I_k\times R_{k+1}\times R_n} \tag{A.2}$$

as in Fig A.1 (c)), where the merged dimension $I_1 \times \cdots \times I_{k-1}$ are replaced by the non-merged dimension R_{k-1}. We further give the results to obtain tensor $\mathcal{A}_d$ and tensor $\mathcal{A}_e$ in Fig A.1 (d) and (e) as follows

$$\mathcal{A}_c \times_{k+1\times\cdots\times n}^{2,\cdots,n-k+1} \mathcal{T}_n = \mathcal{A}_d \in \mathbb{R}^{I_1\times\cdots\times I_k\times R_k\times R_n\times R_{k-1}\times I_k\times R_{k+1}\times R_n} \tag{A.3}$$

and

$$\mathcal{A}_d \times_{1\times\cdots\times k-1}^{1\times\cdots\times k-1} \mathcal{T}_1 = \mathcal{A}_e \in \mathbb{R}^{R_{k-1}\times I_k\times R_{k+1}\times R_n\times R_{k-1}\times I_k\times R_{k+1}\times R_n} \tag{A.4}$$

The red marked trace operation in Fig. A.1(f) gets the trace along the 4^{th} and 8^{th} mode of $\mathcal{A}_e$, thus tensor $\mathcal{A}$ is obtained by $\mathcal{A} = \mathrm{tr}_4^8 \left(\mathcal{A}_e\right).$

B. TENSOR NETWROK MERGING FOR COMPUTING $\mathcal{B}$

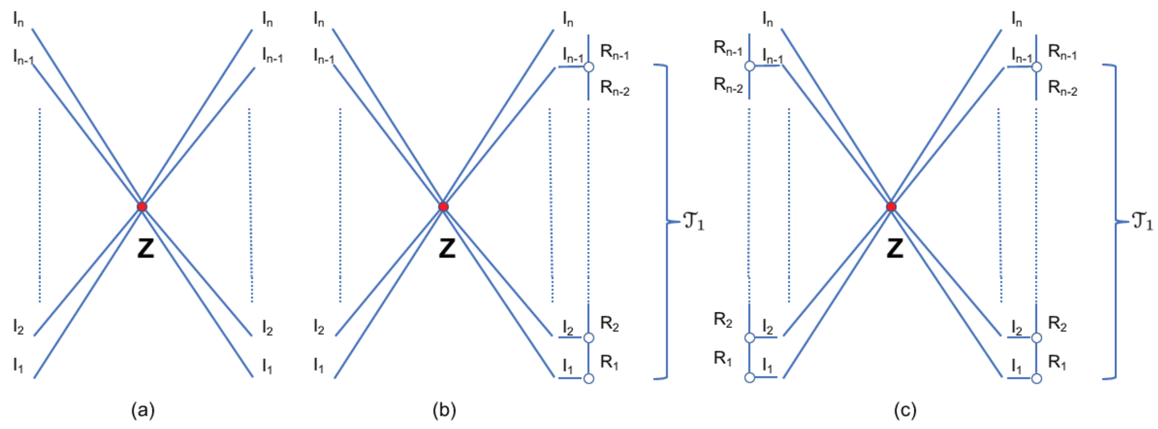


Fig. B.1.: Tensor network merging operation to compute $\mathcal{B}$. (a) Tensor $\mathcal{Z}$ (b) Tensor $\mathcal{B}_b$ (c) Tensor $\mathcal{B}$

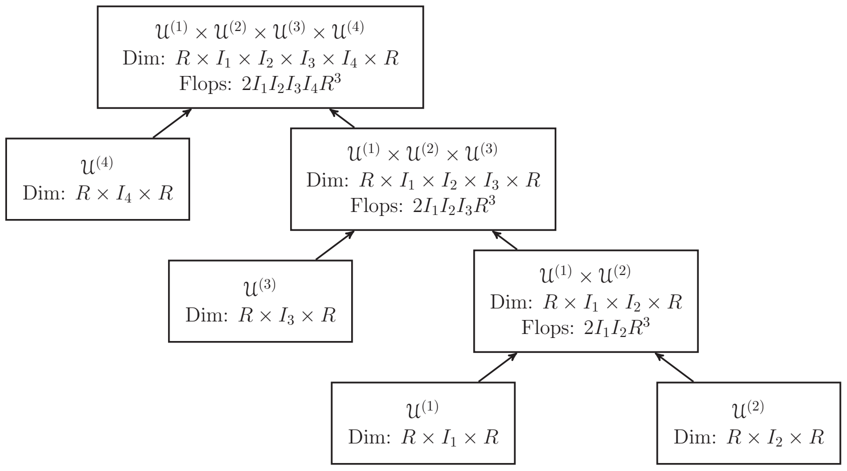
Explanation of Computing $\mathcal{B}$ using (2.25). Figure B.1 shows the steps to compute $\mathcal{B}$. Computing $\mathcal{B}$ follows the same logic as computing $\mathcal{A}$, and is simpler since $\mathcal{T}_n$ does not involve in the computation. The step-by-step computation in Fig. B.1 (b) and (c) are as follows

$$\mathcal{Z} \times_{n+1,\cdots,2n-1}^{1,\cdots,n-1} \mathcal{T}_1 = \mathcal{B}_b \in \mathbb{R}^{I_1 \times \cdots \times I_n \times R_{n-1} \times I_n}, \tag{B.1}$$

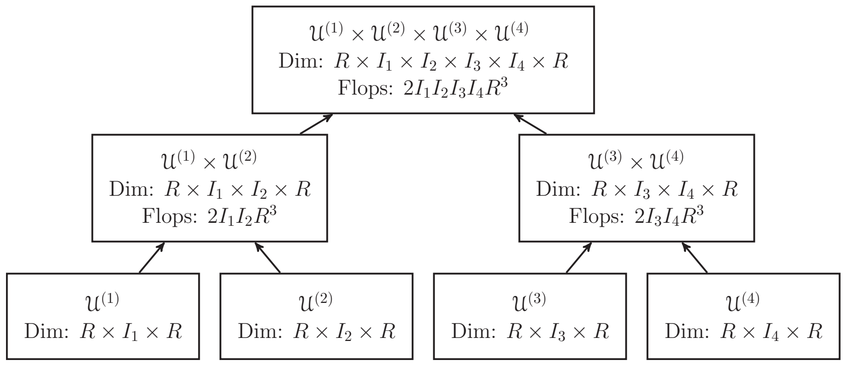
and

$$\mathcal{B}_b \times_{1,\cdots,n-1}^{1,\cdots,n-1} \mathcal{T}_1 = \mathcal{B} \in \mathbb{R}^{R_{n-1} \times I_n \times R_{n-1} \times I_n}. \tag{B.2}$$

C. TENSOR NETWORK MERGING ORDERING FOR TENSOR RING NETS


$\mathcal{U}^{(1)} \times \mathcal{U}^{(2)} \times \mathcal{U}^{(3)} \times \mathcal{U}^{(4)}$
Dim: $R \times I_1 \times I_2 \times I_3 \times I_4 \times R$
Flops: $2I_1I_2I_3I_4R^3$
$\mathcal{U}^{(4)}$
Dim: $R \times I_4 \times R$
$\mathcal{U}^{(1)} \times \mathcal{U}^{(2)} \times \mathcal{U}^{(3)}$
Dim: $R \times I_1 \times I_2 \times I_3 \times R$
Flops: $2I_1I_2I_3R^3$
$\mathcal{U}^{(3)}$
Dim: $R \times I_3 \times R$
$\mathcal{U}^{(1)} \times \mathcal{U}^{(2)}$
Dim: $R \times I_1 \times I_2 \times R$
Flops: $2I_1I_2R^3$
$\mathcal{U}^{(1)}$
Dim: $R \times I_1 \times R$
$\mathcal{U}^{(2)}$
Dim: $R \times I_2 \times R$

(a) Sequential merging


$\mathcal{U}^{(1)} \times \mathcal{U}^{(2)} \times \mathcal{U}^{(3)} \times \mathcal{U}^{(4)}$
Dim: $R \times I_1 \times I_2 \times I_3 \times I_4 \times R$
Flops: $2I_1I_2I_3I_4R^3$
$\mathcal{U}^{(1)} \times \mathcal{U}^{(2)}$
Dim: $R \times I_1 \times I_2 \times R$
Flops: $2I_1I_2R^3$
$\mathcal{U}^{(3)} \times \mathcal{U}^{(4)}$
Dim: $R \times I_3 \times I_4 \times R$
Flops: $2I_3I_4R^3$
$\mathcal{U}^{(1)}$
Dim: $R \times I_1 \times R$
$\mathcal{U}^{(2)}$
Dim: $R \times I_2 \times R$
$\mathcal{U}^{(3)}$
Dim: $R \times I_3 \times R$
$\mathcal{U}^{(4)}$
Dim: $R \times I_4 \times R$

(b) Hierarchical merging

Fig. C.1.: Merge ordering for a 4th order tensor ring segment of shape $R \times I_1 \times I_2 \times I_4 \times I_4 \times R$, with tensor ring rank R. In each node from top to bottom are tensor notation, tensor shape, and flops to obtain the tensor.

For a **merge** operation, the order that each $\mathcal{U}^{(i)}$ is merged determines the total flop count and memory needs. When d is small, a *sequential merging* is commonly applied. However, when d is large, we propose a *hieratical merging* approach instead. For instance, Figures C.1a and C.1b show the two merge orderings when $d = 4$, arriving at a total of $2I_1I_2R^3 + 2I_1I_2I_3R^3 + 2I_1I_2I_3I_4R^2$ flops to construct $\mathcal{U}^{(1,2,3,4)}$ using a sequential ordering, and $2I_1I_2R^3 + 2I_3I_4R^3 + 2I_1I_2I_3I_4R^2$ flops using a hierarchical ordering. To see how both methods scale with I_k and d, if and $d = 2^D$, then a sequential merging gives and Both quantities are upper bounded by $4R^3\tilde{I}^d$ which is a factor of $4R^3$ times the total degrees of freedom.

We can generalize this analysis by proving theorem 4.3.1.

Proof 1. Define $\tilde{I} = I_1 = \cdots = I_d$. Any merging order can be represented by a binary tree. Figures C.1a and C.1b show the binary trees for sequential and hierarchical merging; note that they do not have to be balanced, but every non-leaf node has exactly 2 children. Each $U^{(i)}$ corresponds to a leaf of the tree.

To keep the analysis consistent, we can say that the computational cost of every leaf is 0 (since nothing is actually done unless tensors are merged).

At each parent node, we note that the computational cost of merging the two child nodes is at least $2\times$ that required in the sum of both child nodes. This is trivially true if both children of a node are leaf nodes. For all other cases, define D the number of leaf node descendents of a parent node. Then the computational cost at the parent is $2R^3 \cdot \tilde{I}^D$. If only one of the two child nodes is a leaf node, then we have a recursion

$$2R^3 \cdot \tilde{I}^D = 2R^3\tilde{I} \cdot \tilde{I}^{D-1} \geq 4R^3(\tilde{I}^{D-1})$$

which is always true if $\tilde{I} \geq 2$. If both children are not leaf nodes, then define D_1, and D_2 the number of leaves descendant of two child nodes, with $D = D_1 + D_2$. Then the recursion is

$$2R^3 \cdot \tilde{I}^D = 2R^3\tilde{I}^{D_1}\tilde{I}^{D_2} \geq 4R^3(\tilde{I}^{D_1} + \tilde{I}^{D_2})$$

where the bound is always true for $\tilde{I} \geq 2$ and $D_1, D_2 \geq 2$. Note that every non-leaf node in the tree necessarily has two children, it can never be that $D_1 = 1$ or $D_2 = 1$.

The cost of merging at the root of the tree is always $2R^3\tilde{I}^d = 2R^3 I$. Since each parent costs at least $2\times$ as many flops as the child, the total flop cost must always be between $2R^3 I$ and $4R^3 I$.

2. For the storage bound, the analysis follows from the observation that the storage cost at each node is $R^2\tilde{I}^D$, where D is the number of leaf descendants. Therefore if $\tilde{I} \geq 2$, the most expensive storage step will always be at the root, with $R^2(\tilde{I}^{d_1} + \tilde{I}^{d_2} + \tilde{I}^d)$ storage cost, where $d = d_1 + d_2$ for any partition. Clearly, this value is lower bounded by $R^2\tilde{I}^d = R^2 I$. And, for any partition $d_1 + d_2 = d$, for $\tilde{I} \geq 2$, it is always $\tilde{I}^{d_1} + \tilde{I}^{d_2} \leq \tilde{I}^d$. Therefore the upper bound on storage is $2R^2\tilde{I}^d = 2R^2 I$.

3. It is sufficient to show that for any d power of 2, a sequential merging is more costly in flops than a hierarchical merging, since anything in between has either pure sequential or pure hierarchical trees as subtrees.

   Then a sequential merging gives $2R^3 \;{}_{i=2}^{d} \tilde{I}^i$ flops. If additionally $d = 2^D$ for some integer $D > 0$, then a hierarchical merging costs $2R^3 \;{}_{i=2}^{D} 2^{D-i}\tilde{I}^{2^i}$ flops. To see this, note that in a perfectly balanced binary tree of depth D, at each level i there are 2^{D-i} nodes, each of which are connected to 2^i leaves.

   We now use induction to show that whenever d is a power of 2, hierarchical merging (a fully balanced binary tree) is optimal in terms of flop count. If $d = 2$, there is no variation in merging order. Taking $d = 4$, a sequential merging costs $2R^3(\tilde{I}^3 + \tilde{I}^3 + \tilde{I}^4)$ and a hierarchical merging costs $2R^3(2\tilde{I}^2 + \tilde{I}^4)$, which is clearly cheaper. For some d a power of 2, define S the cost of sequential merging and H the cost of hierarchical merging. Define $G = 2R^3\tilde{I}^{2d}$ the cost at the root for any binary tree with $2d$ leaf nodes. (Note that the cost at the root is agnostic to the merge ordering.) Then for $\hat{d} = 2d$, a hierarchical merging costs $2H + G$ flops. The cost of a sequential merging is

$$\begin{aligned} S + 2R^3\tilde{I}^d \underset{i=1}{\overset{d}{}} \tilde{I}^i &= S + 2R^3\tilde{I}^{d-1} \underset{i=2}{\overset{d}{}} \tilde{I}^i + G \\ &= S + S\tilde{I}^{d-1} + G - 2R^3 d. \end{aligned}$$

Since $2R^3d$ is the cost at the root for d leafs, $S > 2R^3d$, and therefore the above quantity is lower bounded by $G + \tilde{I}^{d-1}S$, which for $d \geq 2$ and $\tilde{I} \geq 2$, is lower bounded by $G + 2S$. By inductive hypothesis, $S > H$, so the cost of sequential merging is always more than that of hierarchical merging, whenever d is a power of 2.

■

D. INITIALIZATION

If x and y are two independent variables, then $\mathrm{Var}[xy] = \mathrm{Var}[x]\mathrm{Var}[y]+\mathrm{Var}[x](\mathbb{E}[y])^2+\mathrm{Var}[y](\mathbb{E}[x])^2$ [107]. Thus a product of two independent symmetric distributed random variables with mean 0 and variance σ^2 itself is symmetric distributed with mean 0 and variance σ^4 (not Gaussian distribution). Further extrapolating, in a matrix or tensor product, each entry is the summation of R independent variables with the same distribution. The central limit theorem gives that the sum can be approximated by a Gaussian $\mathcal{N}(0, R\sigma^4)$ for large R. Thus if all tensor factors are drawn i.i.d. from $\mathcal{N}(0, \sigma^2)$, then after merging d factors the merged tensor elements will have mean 0 and variance $R^d\sigma^{2d}$.

VITA

VITA

Wenqi Wang received his B.S. degree in Physics from Fudan University, Shanghai, China in 2013. He is currently pursuing the Ph.D. degree from School of Industrial Engineering, Purdue University, West Lafayette, IN, USA under the supervision of Prof. Vaneet Aggarwal. During his Ph.D. study he interned at Technicolor Research Artificial Intelligence Lab, Los Altos, CA, USA. He was the recipient of Purdue University's Bilsland Dissertation Fellowship in 2017. His research interests include tensor networks and its application in machine learning and deep learning.