Université de Montréal

Adaptive Learning of Tensor Network Structures

par

Seyed Meraj Hashemizadehaghda

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

en Informatique

October 3, 2022

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Adaptive Learning of Tensor Network Structures

présenté par

Seyed Meraj Hashemizadehaghda

a été évalué par un jury composé des personnes suivantes :

Ioannis Mitliagkas
(président-rapporteur)

Guillaume Rabusseau
(directeur de recherche)

Pierre-Luc Bacon
(membre du jury)

Résumé

Les réseaux tensoriels offrent un cadre puissant pour représenter efficacement des objets de très haute dimension. Les réseaux tensoriels ont récemment montré leur potentiel pour les applications d'apprentissage automatique et offrent une vue unifiée des modèles de décomposition tensorielle courants tels que Tucker, *tensor train* (TT) et *tensor ring* (TR). Cependant, l'identification de la meilleure structure de réseau tensoriel à partir de données pour une tâche donnée est un défi.

Dans cette thèse, nous nous appuyons sur le formalisme des réseaux tensoriels pour développer un algorithme adaptatif générique et efficace pour apprendre conjointement la structure et les paramètres d'un réseau de tenseurs à partir de données. Notre méthode est basée sur une approche simple de type gloutonne, partant d'un tenseur de rang un et identifiant successivement les bords du réseau tensoriel les plus prometteurs pour de petits incréments de rang. Notre algorithme peut identifier de manière adaptative des structures avec un petit nombre de paramètres qui optimisent efficacement toute fonction objective différentiable. Des expériences sur des tâches de décomposition de tenseurs, de complétion de tenseurs et de compression de modèles démontrent l'efficacité de l'algorithme proposé. En particulier, notre méthode surpasse l'état de l'art basée sur des algorithmes évolutionnaires introduit dans [26] pour la décomposition tensorielle d'images (tout en étant plusieurs ordres de grandeur plus rapide) et trouve des structures efficaces pour compresser les réseaux neuronaux en surpassant les approches populaires basées sur le format TT [30].

Mots-clés: réseau de tenseur, décomposition de tenseur, apprentissage automatique

Abstract

Tensor Networks (TN) offer a powerful framework to efficiently represent very high-dimensional objects. TN have recently shown their potential for machine learning applications and offer a unifying view of common tensor decomposition models such as Tucker, tensor train (TT) and tensor ring (TR). However, identifying the best tensor network structure from data for a given task is challenging. In this thesis, we leverage the TN formalism to develop a generic and efficient adaptive algorithm to jointly learn the structure and the parameters of a TN from data. Our method is based on a simple greedy approach starting from a rank one tensor and successively identifying the most promising tensor network edges for small rank increments. Our algorithm can adaptively identify TN structures with small number of parameters that effectively optimize any differentiable objective function. Experiments on tensor decomposition, tensor completion and model compression tasks demonstrate the effectiveness of the proposed algorithm. In particular, our method outperforms the state-of-the-art evolutionary topology search introduced in [26] for tensor decomposition of images (while being orders of magnitude faster) and finds efficient structures to compress neural networks outperforming popular TT based approaches [30].

Keywords: tensor network, tensor decomposition, machine learning

Contents

List of tables

List of figures

Introduction

Matrix factorization is ubiquitous in machine learning and data science and forms the backbone of many algorithms. Tensor decomposition techniques emerged as a powerful generalization of matrix factorization to higher-order arrays. They are particularly suited to handle high-dimensional multi-modal data and have been successfully applied in neuroimaging [51], signal processing [4, 40], spatio-temporal analysis [1, 38] and computer vision [27]. Common tensor learning tasks include tensor decomposition (finding a low-rank approximation of a given tensor), tensor regression (which extends linear regression to the multi-linear setting), and tensor completion (inferring a tensor from a subset of observed entries).

Akin to matrix factorization, tensor methods rely on factorizing a high-order tensor into small factors. However, in contrast with matrices, there are many different ways of decomposing a tensor, each one giving rise to a different notion of rank, including CP, Tucker, Tensor Train (TT) and Tensor Ring (TR). For most tensor learning problems, there is no clear way of choosing which decomposition model to use, and the cost of model mis-specification can be high. It may even be the case that none of the commonly used models is suited for the task, and new decomposition models would achieve better tradeoffs between minimizing the number of parameters and minimizing a given loss function.

We propose an adaptive tensor learning algorithm which is agnostic to decomposition models. Our approach relies on the *tensor network* formalism, which has shown great success in the many-body physics community [35, 9, 8] and has recently demonstrated its potential in machine learning for compressing models [30, 46, 11, 31, 20, 48], developing new insights into the expressiveness of deep neural networks [5, 21], and designing novel approaches to

supervised [42, 12] and unsupervised [41, 14, 29] learning. Tensor networks offer a unifying view of tensor decomposition models, allowing one to reason about tensor factorization in a general manner, without focusing on a particular model.

In this work, we design a greedy algorithm to efficiently search the space of tensor network structures for common tensor problems, including decomposition, completion and model compression. We start by considering the novel tensor optimization problem of minimizing a loss over arbitrary tensor network structures under a constraint on the number of parameters. To the best of our knowledge, this is the first time that this problem is considered. The resulting problem is a bi-level optimization problem where the upper level is a discrete optimization over tensor network structures, and the lower level is a continuous optimization of a given loss function. We propose a greedy approach to optimize the upper-level problem, which is combined with continuous optimization techniques to optimize the lower-level problem. Starting from a rank one initialization, the greedy algorithm successively identifies the most promising edge of a tensor network for a rank increment, making it possible to adaptively identify from data the tensor network structure which is best suited for the task at hand.

The greedy algorithm we propose is conceptually simple, and experiments on tensor decomposition, completion and model compression tasks showcase its effectiveness. Our algorithm significantly outperforms a recent evolutionary algorithm [26] for tensor network decomposition on an image compression task by discovering structures that require less parameters while simultaneously achieving lower recovery errors. The greedy algorithm also outperforms CP, Tucker, TT and TR algorithms on an image completion task and finds more efficient TN structures to compress fully connected layers in neural networks than the TT based method introduced in [30].

Related work. Adaptive tensor learning algorithms have been previously proposed, but they only consider determining the rank(s) of a specific decomposition and are often tailored to a specific tensor learning task (e.g., decomposition or regression). In [1], a greedy algorithm is proposed to adaptively find the ranks of a Tucker decomposition for a spatio-temporal

forecasting task, and in [45] an adaptive Tucker based algorithm is proposed for background subtraction. In [49], the authors present a Bayesian approach for automatically determining the rank of a CP decomposition. In [2] an adaptive algorithm for tensor decomposition in the hierarchical Tucker format is proposed. In [13] a stable rank-adaptive alternating least square algorithm is introduced for completion in the TT format. The problem we consider is considerably more general since we do not assume a fixed tensor network structure (e.g. Tucker, TT, CP, etc.). Exploring other decomposition relying on the tensor network formalism has been sporadically explored. The work which is the most closely related to our contribution is [26] where evolutionary algorithms are used to approximate the best tensor network structure to exactly decompose a given target tensor. However, the method proposed in [26] only searches for TN structures with uniform ranks (with the rank being a hyperparameter) and is limited to the problem of tensor decomposition. In contrast, our method is the first to jointly explore the space of structures and (non-uniform) ranks to minimize an arbitrary loss function over the space of tensor parameters. Lastly, [17] proposes to explore the space of tensor network structures for compressing neural networks, a rounding algorithm for general tensor networks is proposed in [28] and the notions of rank induced by arbitrary tensor networks are studied in [47].

Summary of the contributions. We introduce a tensor learning algorithm which is agnostic to decomposition models. The greedy algorithm we propose is conceptually simple and experiments on tensor decomposition, completion and model compression tasks showcase its effectiveness. We believe this work opens the door to promising directions for developing tensor network based learning algorithms going beyond classical decomposition models commonly used by practitioners. To the best of our knowledge, this is the first time that the problem of learning the structure of tensor networks is considered in such a general framework encompassing a wide range of tensor learning problems, and our work is the first to propose a learning algorithm which is agnostic to decomposition models and can adaptively discover tensor network structures from data.

Outline of the thesis.

We begin in Chapter 1 with a brief introduction on tensors as multi-dimensional arrays and the main operations associated with them. We then introduce *tensor network diagrams* as a tool to intuitively represent tensor operations. Further on, we introduce some of the most common tensor decomposition methods. Finally, we review some tensor machine learning tasks, which we utilize in the following chapters.

In Chapter 2, we describe our main contribution—the adaptive greedy method for tensor structure learning. We start off by formally defining the *tensor network structure learning* problem as a bi-level optimization problem. We then introduce our algorithm, `Greedy-TN`, to tackle this problem; we discuss its components and analyze the computational complexity. This chapter also appears as a preprint on arXiv [15] written by the author, Michelle Liu, Jacob Miller, and the author's supervisor. The author lead the development of the method and the experiments; all co-authors collaborated in writing the paper.

In Chapter 3, we evaluate `Greedy-TN` on different machine learning tasks, specifically tensor decomposition, image compression, tensor completion, and neural network compression.

In Chapter 4, we provide a summary and also discuss potential further work.

This work was also presented at the first workshop on quantum tensor networks in machine learning at NeurIPS in 2020 [16].

Chapter 1

Preliminaries

1.1. Introduction

In this chapter, we present principal notions from tensor algebra and tensor networks, which are at the core of our work. We refer the enthusiastic reader to [23] for a more in-depth primer on tensor algebra.4

1.2. Notations

We first introduce the notations used throughout this thesis. For any integer k, $[k]$ denotes the set of integers from 1 to k, i.e., $[k] = \{1, 2, \cdots, k\}$. We use lower case bold letters for vectors (e.g. $\mathbf{v} \in \mathbb{R}^{d_1}$), upper case bold letters for matrices (e.g. $\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$) and bold calligraphic letters for higher order tensors (e.g. $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$). The ith row (resp. column) of a matrix $\mathbf{M}$ will be denoted by $\mathbf{M}_{i,:}$ (resp. $\mathbf{M}_{:,i}$). This notation is extended to slices (fibers) of a tensor in the obvious way. For example, given a third-order tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ its mode-1, 2 and 3 fibers are denoted as $\boldsymbol{\mathcal{T}}_{:,j,k}$, $\boldsymbol{\mathcal{T}}_{i,:,k}$, and $\boldsymbol{\mathcal{T}}_{i,j,:}$ respectively.

1.3. Tensors

A *tensor* $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times \cdots \times d_p}$ can simply be seen as a multidimensional array ($\boldsymbol{\mathcal{T}}_{i_1,\cdots,i_p} : i_n \in [d_n], n \in [p]$). The inner product of two tensors is defined by $\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{T}} \rangle = \sum_{i_1,\cdots,i_p} \boldsymbol{\mathcal{S}}_{i_1 \cdots i_p} \boldsymbol{\mathcal{T}}_{i_1 \cdots i_p}$ and the Frobenius norm of a tensor is defined by $\|\boldsymbol{\mathcal{T}}\|_F^2 = \langle \boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{T}} \rangle$. The *mode-n matrix*

product of a tensor $\boldsymbol{\mathcal{T}}$ and a matrix $\mathbf{X} \in \mathbb{R}^{m \times d_n}$ is a tensor denoted by $\boldsymbol{\mathcal{T}} \times_n \mathbf{X}$. It is of size $d_1 \times \cdots \times d_{n-1} \times m \times d_{n+1} \times \cdots \times d_p$ and is obtained by contracting the nth mode of $\boldsymbol{\mathcal{T}}$ with the second mode of $\mathbf{X}$, e.g. for a 3rd order tensor $\boldsymbol{\mathcal{T}}$, we have $(\boldsymbol{\mathcal{T}} \times_2 \mathbf{X})_{i_1 i_2 i_3} = \sum_j \boldsymbol{\mathcal{T}}_{i_1 j i_3} \mathbf{X}_{i_2 j}$. The nth mode matricization of $\boldsymbol{\mathcal{T}}$ is denoted by $\boldsymbol{\mathcal{T}}_{(n)} \in \mathbb{R}^{d_n \times \prod_{i \neq n} d_i}$.

1.4. Tensor network diagrams

Tensor network diagrams allow one to represent complex operations on tensors (mainly contractions) in a graphical and intuitive way. A tensor network (TN) is simply a graph where nodes represent tensors, and edges represent contractions between tensor modes, i.e. a summation over an index shared by two tensors. In a tensor network, the arity of a vertex (i.e. the number of *legs* of a node) corresponds to the order of the tensor (see Figure 1.1).

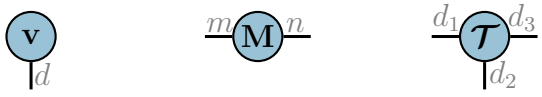
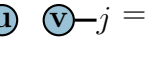
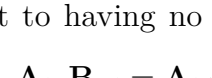
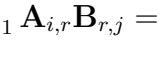


Figure 1.1. Tensor network representation of a vector $\mathbf{v} \in \mathbb{R}^d$, a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ and a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$.

We will sometimes add indices to legs of a tensor network to refer to its components or sub-tensors. For example, the tensor networks , and represent a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, the ith row of $\mathbf{A}$ and the component $\mathbf{A}_{i,j}$, respectively.

Connecting two legs in a tensor network represents a contraction over the corresponding indices. Consider the following simple tensor network with two nodes: . The first node represents a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and the second one a vector $\mathbf{x} \in \mathbb{R}^n$. Since this tensor network has one dangling leg (i.e. an edge which is not connected to any other node), it represents a first order tensor, i.e., a vector. The edge between the second leg of $\mathbf{A}$ and the leg of $\mathbf{x}$ corresponds to a contraction between the second mode of $\mathbf{A}$ and the first mode of $\mathbf{x}$. Hence, the resulting tensor network represents the classical matrix-vector product, which can be seen by calculating the ith component of this tensor network: $= \sum_j \mathbf{A}_{ij} \mathbf{x}_j = (\mathbf{A}\mathbf{x})_i$. Other examples of tensor network representations of common operations on matrices and tensors can be found in Figure 1.2.

Lastly, it is worth mentioning that disconnected tensor networks correspond to tensor products, e.g.,  $= \mathbf{u}\mathbf{v}^\top$ is the outer product of $\mathbf{u}$ and $\mathbf{v}$ with components  $= \mathbf{u}_i\mathbf{v}_j$. Consequently, an edge of dimension (or rank) 1 in a TN is equivalent to having no edge between the two nodes, e.g., if $R = 1$ we have  $= \sum_{r=1}^{R} \mathbf{A}_{i,r}\mathbf{B}_{r,j} = \mathbf{A}_{i,1}\mathbf{B}_{1,j} =$  .

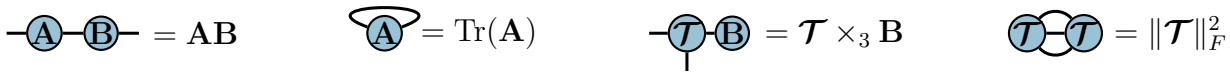


Figure 1.2. Tensor network representation of common operation on matrices and tensors.

1.5. Basic Operations on Tensors

Outer Product. The outer product of p vectors $\mathbf{v}_1 \in \mathbb{R}^{d_1}, \ldots, \mathbf{v}_p \in \mathbb{R}^{d_p}$, denoted by $\mathbf{v}_1 \circ \cdots \circ \mathbf{v}_p$, is a p-th order tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times \cdots \times d_p}$ with the elements $\boldsymbol{\mathcal{T}}_{i_1,\ldots,i_p} = (\mathbf{v}_1)_{i_1} \ldots (\mathbf{v}_p)_{i_p}$, where $(\mathbf{v}_k)_{i_k}$ is the i_k-th element of $\mathbf{v}_k$. Therefore, the outer product can be seen as one basic way to construct a tensor from vectors; tensors that can be written as an outer product of vectors are called *rank one tensors*. Figure 1.3 shows the TN diagram representation of the outer product of three vectors, $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$.

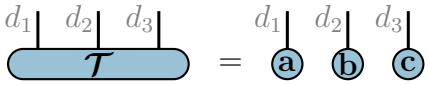


Figure 1.3. Tensor network diagram of the outer product of three vectors.

n-mode product. The n-mode (matrix) product is a generalization of the matrix multiplication. For a p-th order tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times \cdots \times d_n \times \cdots \times d_p}$ and a matrix $\mathbf{M} \in \mathbb{R}^{m \times d_n}$, the n-mode product is the contraction of the nth mode of $\boldsymbol{\mathcal{T}}$ with the second mode of the matrix $\mathbf{M}$, resulting in a p-th order tensor $\boldsymbol{\mathcal{T}} \times_n \mathbf{M} \in \mathbb{R}^{d_1 \times \cdots \times d_{n-1} \times m \times d_{n+1} \times \cdots \times d_p}$. More formally,

$$(\boldsymbol{\mathcal{T}} \times_n \mathbf{M})_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_p} = \sum_{i_n=1}^{d_n} \boldsymbol{\mathcal{T}}_{i_1,\ldots,i_n,\ldots,i_p} \mathbf{M}_{j,i_n}. \tag{1.5.1}$$

Figure 1.4 illustrates the tensor network diagram associated with the mode-3 product of a fourth order tensor with a matrix.

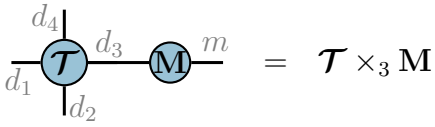


Figure 1.4. Tensor network diagram of the mode-3 product of a fourth order tensor with a matrix.

Matricization. Matricization or *unfolding* is the operation that rearranges the entries of a tensor into a matrix. There are many ways to matricize a tensor, e.g,. a $3 \times 5 \times 7$ tensor can be arranged as a 15×7 matrix or a 3×35 matrix, and so on [23]. In the general case a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_p}$ can be rearranged into a matrix where each of the p modes is either mapped to the row or the column of the flattened matrix. Formally, let I and J be a bi-partition of $[p]$ (i.e., $[p] = I \cup J$ and $I \cap J = \emptyset$) then we will denote by $\flat_{I,J}(\boldsymbol{\mathcal{T}}) \in \mathbb{R}^{\prod_{i \in I} d_i \times \prod_{j \in J} d_j}$ the matricization of $\boldsymbol{\mathcal{T}}$ obtained by using the modes in I for rows and the modes in J for columns.

One widely used matricization is the *mode-n* matricization, where the mode-n fibers are arranged as the columns of matricization. The mode-n matricization of a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times \cdots \times d_n \times \cdots \times d_p}$ is denoted by $\boldsymbol{\mathcal{T}}_{(n)}$; this is the same as $\flat_{\{n\},[p]\setminus n}(\boldsymbol{\mathcal{T}})$.

1.6. Tensor decomposition

Tensor decompositions are generalizations of matrix factorizations to their high-order extensions—tensors, and as so their usecases are two-fold: 1) compressing large tensors to reduce storage costs, and 2) discovering (low rank) latent representations in complex high dimensional data. We now briefly present the most common tensor decomposition models.

1.6.1. CP decomposition

The CP decomposition [19] of a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_p}$ consists in expressing T as a sum of rank-1 tensors:

$$\boldsymbol{\mathcal{T}} = \sum_{r=1}^{R} \mathbf{v}_1^r \circ \cdots \circ \mathbf{v}_p^r, \tag{1.6.1}$$

where the CP rank of $\boldsymbol{\mathcal{T}}$ is defined as the smallest R for which the equation holds. Unlike the matrix case where computing the rank can be done in polynomial time ($\mathcal{O}(d^3)$), computing the CP rank is an NP-hard problem [18]. However, certain bounds can be obtained, e.g., CP-rank($\boldsymbol{\mathcal{T}}$) $\leq \min_i \prod_{j \neq i} d_j$. Another distinction between the CP rank and matrix rank is that a random matrix is almost surely (with probability equal to one) full rank, but the CP rank of a random tensor can take multiple values with non-zero probability. Lastly, another exciting difference is the uniqueness of the CP decomposition under some mild assumptions [25]. The TN representation of the CP decomposition of a fourth order tensor is illustrated in Figure 1.5.

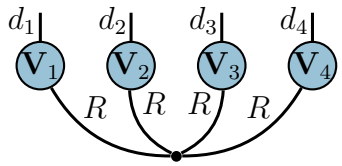


Figure 1.5. Tensor network diagram of the CP decomposition of a fourth order tensor. The factor matrices $\mathbf{V}_i \in \mathbb{R}^{d_i \times R}$ are construced by concatenating the rank one vectors as, i.e, $\mathbf{V}_i = [\mathbf{v}_i^1 | \mathbf{v}_i^2 | \cdots | \mathbf{v}_i^R]$. The black dot represents a diagonal tensor with ones along the superdiagonal.

1.6.2. Tucker decomposition

The Tucker decomposition, first introduced in [43], expresses a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_p}$ as a core tensor $\boldsymbol{\mathcal{G}}$ multiplied by a orthogonal matrix along each of its modes.

$$\boldsymbol{\mathcal{T}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \cdots \times_p \mathbf{U}_p \tag{1.6.2}$$

where $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_P}$, $\mathbf{U}_i \in \mathbb{R}^{d_i \times R_i}$ and $\mathbf{U}_i^\top \mathbf{U}_i = \mathbf{I}$ for all $i \in [p]$. The TN representation of the tucker decomposition of a fourth order tensor is illustrated in Figure 1.6.

The Tucker rank, or multilinear rank (n-rank), of a tensor $\boldsymbol{\mathcal{T}}$ is the smallest tuple $(R_1, R_2, \cdots, R_p)$ for which the tucker decomposition (Equation 1.6.2) exists. It can be shown that the multilinear rank of $\boldsymbol{\mathcal{T}}$ is given by the ranks of its matricizations, i.e., n-rank($\boldsymbol{\mathcal{T}}$) $= \Big(\text{rank}(\boldsymbol{\mathcal{T}}_{(1)}), \text{rank}(\boldsymbol{\mathcal{T}}_{(2)}), \cdots, \text{rank}(\boldsymbol{\mathcal{T}}_{(p)})\Big)$ [7].

In contrast to CP decomposition, Tucker decomposition is not unique as we can multiply the factor matrices with any unitary matrix as long as the invert is applied to the corresponding mode of the core tensor.

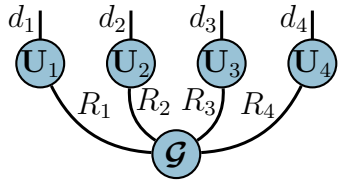


Figure 1.6. Tensor network diagram of the Tucker decomposition of a fourth order tensor.

1.6.3. Tensor train decomposition

Given a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_p}$, the tensor train (TT) decomposition [33], also known as matrix product states (MPS) [36, 32] in the physics community, factorizes $\boldsymbol{\mathcal{T}}$ into p core tensors $\boldsymbol{\mathcal{G}}^1 \in \mathbb{R}^{d_1 \times R_1}, \boldsymbol{\mathcal{G}}^2 \in \mathbb{R}^{R_1 \times d_2 \times R_2}, \ldots, \boldsymbol{\mathcal{G}}^{p-1} \in \mathbb{R}^{R_{p-2} \times d_{p-1} \times R_{p-1}}, \boldsymbol{\mathcal{G}}^p \in \mathbb{R}^{R_{p-1} \times d_p}$ in the following form

$$\boldsymbol{\mathcal{T}}_{i_1,\cdots,i_p} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_{p-1}=1}^{R_{p-1}} \boldsymbol{\mathcal{G}}^1_{i_1,r_1} \boldsymbol{\mathcal{G}}^2_{r_1,i_2,r_2} \boldsymbol{\mathcal{G}}^3_{r_2,i_3,r_3} \cdots \boldsymbol{\mathcal{G}}^{p-1}_{r_{p-1},i_{p-1},r_{p-1}} \boldsymbol{\mathcal{G}}^p_{r_{p-1},i_p}. \tag{1.6.3}$$

The TN representation of this decomposition for a fourth order tensor is illustrated in Figure 1.7.

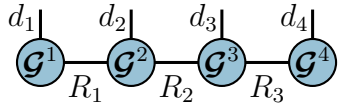


Figure 1.7. Tensor network diagram of the TT decomposition of a fourth order tensor.

Similar to Tucker, the TT decomposition naturally gives rise to an associated notion of rank: the TT rank is the smallest tuple $(R_1, R_2, \cdots, R_{p-1})$ such that a TT decomposition exists. The TT rank of a tensor can be determined in terms of the rank of its matricization (a different matricization than the typical mode-n matricization), i.e., TT-rank$(\boldsymbol{\mathcal{T}}) = \Big(\text{rank}(\flat_{\{1\},[p]\setminus\{1\}}(\boldsymbol{\mathcal{T}})), \text{rank}(\flat_{\{1,2\},[p]\setminus\{1,2\}}(\boldsymbol{\mathcal{T}})), \cdots, \text{rank}(\flat_{[p-1],\{p\}}(\boldsymbol{\mathcal{T}}))\Big)$, where $\flat_{[k],[p]\setminus[k]}(\boldsymbol{\mathcal{T}}) \in \mathbb{R}^{\prod_{i=1}^{k} d_i \times \prod_{j=k+1}^{p} d_j}$ is the matricization of $\boldsymbol{\mathcal{T}}$ with modes $\{1, 2, \cdots, k\}$ as the rows and modes $\{k+1, k+2, \cdots, p\}$ as the columns.

1.6.4. Tensor ring decomposition

The tensor ring (TR) decomposition [50] represents a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_p}$ by a sequence of third-order tensors, $\boldsymbol{\mathcal{G}}^i \in \mathbb{R}^{R_{i-1} \times d_i \times R_i}$ for all $i \in [p]$ with $R_0 = R_p$, that are multiplied circularly. The decomposition takes the following form

$$\boldsymbol{\mathcal{T}}_{i_1,\cdots,i_p} = \mathrm{Tr}(\boldsymbol{\mathcal{G}}^1_{:,i_1,:}\boldsymbol{\mathcal{G}}^2_{:,i_2,:}\cdots\boldsymbol{\mathcal{G}}^p_{:,i_p,:}), \tag{1.6.4}$$

It can be seen that the tensor train decomposition is a particular case of the tensor ring decomposition where R_0 must be equal to 1 (R_0 is thus omitted when referring to the rank of a TT decomposition). The TN representation of the TR decomposition for a fourth order tensor is shown in Figure 1.8.

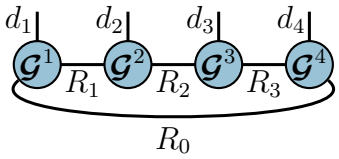


Figure 1.8. Tensor network diagram of the TR decomposition of a fourth order tensor.

As shown tensor networks offer a unifying view of tensor decomposition models. Each decomposition is naturally associated with the graph topology of the underlying TN. For example, the Tucker decomposition corresponds to star graphs, the TT decomposition corresponds to chain graphs, and the TR decomposition model corresponds to cyclic graphs. The relation between the rank of a decomposition and its number of parameters is different for each model. Letting p be the order of the tensor, d its largest dimension and R the rank of the decomposition (assuming uniform ranks), the number of parameters is in $\mathcal{O}\left(R^p + pdR\right)$ for Tucker, and $\mathcal{O}\left(pdR^2\right)$ for TT and TR. One can see that the Tucker decomposition is not well suited for tensors of very high order since the size of the core tensor grows exponentially with p.

1.7. Tensor learning tasks

In this section we briefly introduce some of the common machine learning tasks where tensors play a central role. These tasks include the tensor low-rank approximation problem and the tensor completion problem.

1.7.1. Low-rank tensor approximation

Approximating a given tensor by a low-rank tensor which is representable with less number of parameters is a common task in many applications such as data compression, image denoising and genomic data analysis. Here we are interested in identifying the "best" rank-r tensor to approximate a given tensor. More formally given a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_p}$ the problem can be stated as

$$\min_{\boldsymbol{\mathcal{W}}} \|\boldsymbol{\mathcal{T}} - \boldsymbol{\mathcal{W}}\|_F^2 \quad \text{s.t. } \operatorname{rank}(\boldsymbol{\mathcal{W}}) \leq R \tag{1.7.1}$$

where $\boldsymbol{\mathcal{W}}$ is the low-rank approximation of $\boldsymbol{\mathcal{T}}$ and rank could be any notion of tensor rank as defined in the previous section.

Unlike for matrices where the solution of the low-rank approximation problem is given by the Eckart–Young theorem, the problem for tensor is much more difficult as the set of low-rank tensors is not closed and the Eckart–Young theorem cannot be extended to tensors.

1.7.2. Tensor completion

Tensor completion is a generalization of the classical matrix completion problem [3]. Similarly, tensor completion applications are ubiquitous, e.g., recommendation systems [52] and image processing [44]. In these applications, the underlying tensor is only partially observed, and the goal is to estimate the missing entries. More formally given a partially observed target tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_p}$ from a set of observed entries $\{\boldsymbol{\mathcal{X}}_{i_1,\cdots,i_p}\}_{(i_1,\cdots,i_p)\in\Omega}$ where $\Omega \subset [d_1] \times \cdots \times [d_p]$ we wish to recover the missing entries of $\boldsymbol{\mathcal{X}}$. Similar to the matrix case, to avoid the problem being underdetermined, and to restrict the degrees of freedom a

low rank assumption is necessary [10]. Therefore, the problem could be presented as

$$\min_{\boldsymbol{\mathcal{W}}} \frac{1}{|\Omega|} \sum_{(i_1,\cdots,i_p)\in\Omega} (\boldsymbol{\mathcal{W}}_{i_1,\cdots,i_p} - \boldsymbol{\mathcal{X}}_{i_1,\cdots,i_p})^2 \quad \text{s.t. } \operatorname{rank}(\boldsymbol{\mathcal{W}}) \leq R, \tag{1.7.2}$$

where the rank constrain could refer to any notation of rank as defined in the previous section.

1.8. Tensor network learning

In this section, we introduce a unifying view of common tensor learning problems. Most tensor learning problems can be seen as special cases of the following optimization problem:

$$\min_{\boldsymbol{\mathcal{W}}\in\mathbb{R}^{d_1\times\cdots\times d_p}} \mathcal{L}(\boldsymbol{\mathcal{W}}) \quad \text{s.t. } \operatorname{rank}(\boldsymbol{\mathcal{W}}) \leq R \tag{1.8.1}$$

where $\mathcal{L} : \mathbb{R}^{d_1\times\cdots\times d_p} \to \mathbb{R}$ is a loss function and $\operatorname{rank}(\boldsymbol{\mathcal{W}})$ denotes *some* notion of tensor rank (e.g. CP, Tucker, TT, ...). The rank constraint R is either a single number or a tuple of integers depending on the decomposition considered and it often corresponds to an hyper-parameter of the underlying tensor learning problem controlling model capacity.

Different choices of loss functions in Problem 1.8.1 give rise to different common tensor learning problems. For tensor decomposition, the objective is to find the best low rank approximation of a given target tensor $\boldsymbol{\mathcal{X}}$ and a common choice of loss function is $\mathcal{L}(\boldsymbol{\mathcal{W}}) = \|\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{X}}\|_F^2$. One form of tensor regression consists in learning a linear function $f : \mathbb{R}^{d_1\times\cdots\times d_p} \to \mathbb{R}$ from a training set of input-output examples $\{(\boldsymbol{\mathcal{X}}^{(n)}, y^{(n)})\}_{n=1}^N \subset \mathbb{R}^{d_1\times\cdots d_p} \times \mathbb{R}$ where each $y^{(n)} \simeq f(\boldsymbol{\mathcal{X}}^{(n)})$. A common choice of loss function for tensor regression is the mean squared error: $\mathcal{L}(\boldsymbol{\mathcal{W}}) = \frac{1}{N}\sum_{n=1}^N (\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}^{(i)}\rangle - y^{(i)})^2$. The tensor completion task consists in estimating a target tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{d_1\times\cdots\times d_p}$ from a set of observed entries $\{\boldsymbol{\mathcal{X}}_{i_1,\cdots,i_p}\}_{(i_1,\cdots,i_p)\in\Omega}$ where $\Omega \subset [d_1]\times\cdots\times[d_p]$. A common loss function for tensor completion is again the squared error: $\mathcal{L}(\boldsymbol{\mathcal{W}}) = \frac{1}{|\Omega|}\sum_{(i_1,\cdots,i_p)\in\Omega} (\boldsymbol{\mathcal{W}}_{i_1,\cdots,i_p} - \boldsymbol{\mathcal{X}}_{i_1,\cdots,i_p})^2$. Lastly, learning matrix product state models for classification [42] and sequence modeling [14] also falls within this general formulation by using the cross-entropy or log likelihood as a loss function.

The rank constraint in Problem 1.8.1 often serves two purposes: it acts as a regularizer but is also a way to make the problem tractable. Indeed, in some instances of these tensor learning problems the size of the tensor parameter $\boldsymbol{\mathcal{W}}$ is so large that it cannot be stored in memory. Unfortunately, for almost all common tensor learning tasks, Problem 1.8.1 is NP-hard because of the tensor rank constraint [18]. There are two common ways of handling this constraint: either a convex relaxation is used and the resulting problem is solved using classical convex optimization toolboxes, or the objective function is minimized with respect to the factors involved in the decomposition of the tensor $\boldsymbol{\mathcal{W}}$ rather than w.r.t. $\boldsymbol{\mathcal{W}}$ itself. For the latter, an example for a tensor decomposition task with a Tucker rank constraint would be to rewrite Problem 1.8.1 in the following unconstrained form: $\min_{\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times \cdots \times R_p}, \mathbf{U}_i \in \mathbb{R}^{d_i \times R_i}, 1 \leq i \leq p} \|\boldsymbol{\mathcal{G}} \times_1 \mathbf{U}_1 \times_2 \cdots \times_p \mathbf{U}_p - \boldsymbol{\mathcal{X}}\|_F^2$, where the rank constraint has been removed but the objective function is not convex anymore. This is the approach we will take for the greedy algorithm we introduce in the following section.

This formulation encompasses classical tensor learning problems:

- Tensor decomposition: $\mathcal{L}(\boldsymbol{\mathcal{W}}) = \|\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{X}}\|_F^2$
- Tensor regression: $\mathcal{L}(\boldsymbol{\mathcal{W}}) = \frac{1}{N} \sum_{n=1}^{N} (\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}^{(i)} \rangle - y^{(i)})^2$
- Tensor completion: $\mathcal{L}(\boldsymbol{\mathcal{W}}) = \frac{1}{|\Omega|} \sum_{(i_1, \cdots, i_p) \in \Omega} (\boldsymbol{\mathcal{W}}_{i_1, \cdots, i_p} - \boldsymbol{\mathcal{X}}_{i_1, \cdots, i_p})^2$

Chapter 2

A Greedy Algorithm for Tensor Network Structure Learning

2.1. Introduction

In this chapter, we first introduce the problem of tensor network structure learning—minimizing a loss function defined over arbitrary tensor network structures under a constraint on the number of parameters—which we formalize as a bi-level optimization problem. We then propose an iterative greedy algorithm to tackle this problem.

2.2. Tensor Network Optimization

We consider the problem of minimizing a loss function $\mathcal{L} : \mathbb{R}^{d_1 \times \cdots \times d_p} \to \mathbb{R}_+$ w.r.t. a tensor $\boldsymbol{\mathcal{W}}$ efficiently parameterized as a tensor network (TN). We first introduce our notations for TN.

Without loss of generality, we consider TN having one factor per dimension of the parameter tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{d_1 \times \cdots \times d_p}$, where each of the factors has one dangling leg corresponding to one of the dimensions d_i (we will discuss how this encompasses TN structures with internal nodes such as Tucker at the end of this section). In this case, a TN structure is summarized by a collection of ranks $(R_{i,j})_{1 \leq i < j \leq p}$ where each $R_{i,j} \geq 1$ is the dimension of the edge connecting the ith and jth nodes of the TN (for convenience, we assume $R_{i,j} = R_{j,i}$ if $i > j$). If there

is no edge between nodes i and j in a TN, $R_{i,j}$ is thus equal to 1 (see Section 1.4). A TN decomposition of $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{d_1 \times \cdots \times d_p}$ is then given by a collection of core tensors $\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}$ where each $\boldsymbol{\mathcal{G}}^{(i)}$ is of size $R_{1,i} \times \cdots \times R_{i-1,i} \times d_i \times R_{i,i+1} \times \cdots \times R_{i,p}$. Each core tensor is of order p but some of its dimensions may be equal to one (representing the absence of edge between the two cores in the TN structure). We use $\text{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)})$ to denote the resulting tensor. Formally, for an order 4 tensor we have

$$\text{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(4)})_{i_1 i_2 i_3 i_4} = \sum_{j_1^2=1}^{R_{1,2}} \sum_{j_1^3=1}^{R_{1,3}} \cdots \sum_{j_3^4=1}^{R_{3,4}} \boldsymbol{\mathcal{G}}^{(1)}_{i_1, j_1^2, j_1^3, j_1^4} \boldsymbol{\mathcal{G}}^{(2)}_{j_1^2, i_2, j_2^3, j_2^4} \boldsymbol{\mathcal{G}}^{(3)}_{j_1^3, j_2^3, i_3, j_3^4} \boldsymbol{\mathcal{G}}^{(4)}_{j_1^4, j_2^4, j_3^4, i_4}.$$

This definition is straightforwardly extended to TN representing tensors of arbitrary orders.

As an illustration, for a TT decomposition the ranks of the tensor network representation would be such that $R_{i,j} \neq 1$ if and only if $j = i + 1$. The problem of finding a rank (r_1, r_2, r_3) TT decomposition of a target tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3 \times d_4}$ can thus be formalized as

$$\min_{\substack{\boldsymbol{\mathcal{G}}^{(1)} \in \mathbb{R}^{d_1 \times r_1 \times 1 \times 1}, \boldsymbol{\mathcal{G}}^{(2)} \in \mathbb{R}^{r_1 \times d_2 \times r_2 \times 1}, \\ \boldsymbol{\mathcal{G}}^{(3)} \in \mathbb{R}^{1 \times r_2 \times d_3 \times r_3}, \boldsymbol{\mathcal{G}}^{(4)} \in \mathbb{R}^{1 \times 1 \times r_3 \times d_4}}} \mathcal{L}(\text{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \boldsymbol{\mathcal{G}}^{(2)}, \boldsymbol{\mathcal{G}}^{(3)}, \boldsymbol{\mathcal{G}}^{(4)})) \tag{2.2.1}$$

where $\mathcal{L}(\boldsymbol{\mathcal{W}}) = \|\boldsymbol{\mathcal{T}} - \boldsymbol{\mathcal{W}}\|_F^2$. Other common tensor problems can be formalized in this manner. For example, the tensor train completion problem would be formalized similarly with the loss function being $\mathcal{L}(\boldsymbol{\mathcal{W}}) = \frac{1}{|\Omega|} \sum_{(i_1, \cdots, i_p) \in \Omega} (\boldsymbol{\mathcal{W}}_{i_1, \cdots, i_p} - \boldsymbol{\mathcal{T}}_{i_1, \cdots, i_p})^2$ where $\Omega \subset [d_1] \times \cdots \times [d_p]$ is the set of observed entries of $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times \cdots \times d_p}$, and learning TT models for classification [42] and sequence modeling [14] also falls within this general formulation by using the cross-entropy or log-likelihood as a loss function.

We now explain how our formalism encompasses TN structure with internal nodes, such as the Tucker format. Since a rank one edge in a TN is equivalent to having no edge, internal cores can be represented as cores whose dangling leg have dimension 1. Consider for example the Tucker decomposition $\boldsymbol{\mathcal{T}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \mathbf{U}_3 \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ of rank (r_1, r_2, r_3). The tensor $\boldsymbol{\mathcal{T}}$ can naturally be seen as a fourth order tensor $\tilde{\boldsymbol{\mathcal{T}}} \in \mathbb{R}^{1 \times d_1 \times d_2 \times d_3}$, $\boldsymbol{\mathcal{G}}$ as $\tilde{\boldsymbol{\mathcal{G}}} \in \mathbb{R}^{1 \times r_1 \times r_2 \times r_3}$, $\mathbf{U}_1$ as $\tilde{\mathbf{U}}_1 \in \mathbb{R}^{r_1 \times d_1 \times 1 \times 1}$, $\mathbf{U}_2$ as $\tilde{\mathbf{U}}_2 \in \mathbb{R}^{r_2 \times 1 \times d_2 \times 1 \times 1}$ and $\mathbf{U}_3$ as $\tilde{\mathbf{U}}_3 \in \mathbb{R}^{r_3 \times 1 \times 1 \times d_3}$. With these definitions, one can check that $\text{TN}(\tilde{\boldsymbol{\mathcal{G}}}, \tilde{\mathbf{U}}_1, \tilde{\mathbf{U}}_2, \tilde{\mathbf{U}}_3) = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \mathbf{U}_3 = \boldsymbol{\mathcal{T}}$. More complex TN structure with internal nodes such as hierarchical Tucker can be represented

using our formalism in a similar way. The assumption that each core tensor in a TN structure has one dangling leg corresponding to each of the dimensions of the tensor $\boldsymbol{\mathcal{T}}$ is thus without loss of generality, since it suffices to augment $\boldsymbol{\mathcal{T}}$ with singleton dimensions to represent TN structures with internal nodes.

2.3. Tensor Network Structure Learning

A large class of TN learning problems consist in optimizing a loss function w.r.t. the core tensors of a *fixed* TN structure; this is, for example, the case of the TT completion problem: the rank of the decomposition may be selected using, e.g., cross-validation, but the *overall structure* of the TN is fixed *a priori*. In contrast, we propose to optimize the loss function simultaneously w.r.t. the core tensors of the TN *and the TN structure itself*. This joint optimization problem can be formalized as

$$\min_{\substack{R_{i,j},\\ 1\leq i<j\leq p}} \min_{\boldsymbol{\mathcal{G}}^{(1)},\cdots,\boldsymbol{\mathcal{G}}^{(p)}} \mathcal{L}(\mathrm{TN}(\boldsymbol{\mathcal{G}}^{(1)},\cdots,\boldsymbol{\mathcal{G}}^{(p)})) \qquad \text{s.t. } \texttt{size}(\boldsymbol{\mathcal{G}}^{(1)},\cdots,\boldsymbol{\mathcal{G}}^{(p)}) \leq C \tag{2.3.1}$$

where $\mathcal{L}$ is a loss function, each core tensor $\boldsymbol{\mathcal{G}}^{(i)}$ is in $\mathbb{R}^{R_{1,i}\times\cdots\times R_{i-1,i}\times d_i\times R_{i,i+1}\times\cdots\times R_{i,p}}$, C is a bound on the number of parameters, and $\texttt{size}(\boldsymbol{\mathcal{G}}^{(1)},\cdots,\boldsymbol{\mathcal{G}}^{(p)})$ is the number of parameters of the TN, which is equal to $\sum_{i=1}^{p} d_i R_{1,i}\cdots R_{i-1,i}R_{i,i+1}\cdots R_{i,p}$. Note that if K is the maximum arity of a node in a TN, its number of parameters is in $\mathcal{O}\left(pdR^K\right)$ where $d = \max_i d_i$ and $R = \max_{i,j} R_{i,j}$.

Problem 2.3.1 is a bi-level optimization problem where the upper level is a discrete optimization over TN structures, and the lower level is a continuous optimization problem (assuming the loss function is continuous). If it is possible to solve the lower level continuous optimization, an exact solution can be found by enumerating the search space of the upper level, i.e. enumerating all TN structures satisfying the constraint on the number of parameters, and selecting the one achieving the lower value of the objective. This approach is, of course, not realistic since the search space is combinatorial in nature, and its size grows exponentially with p. Moreover, for most tensor learning problems, the lower-level continuous optimization

problem is NP-hard [18]. In the next section, we propose a general greedy approach to tackle this problem.

2.4. Greedy Algorithm

In this section we propose a greedy algorithm to solve the tensor network structure learning problem (Problem 2.3.1). The algorithm consists in first optimizing the loss function $\mathcal{L}$ starting from a rank one initialization of the tensor network, i.e. $R_{i,j}$ is set to one for all i, j and each core tensor $\boldsymbol{\mathcal{G}}^{(i)} \in \mathbb{R}^{R_{1,i} \times \cdots \times R_{i-1,i} \times d_i \times R_{i,i+1} \times \cdots \times R_{i,p}}$ is initialized randomly. At each subsequent iteration of the greedy algorithm, the most promising edge of the current TN structure is identified through some efficient heuristic, the corresponding rank is increased, and the loss function is optimized w.r.t. the core tensors of the new TN structure initialized through a weight transfer mechanism. In addition, at each iteration, the greedy algorithm identifies nodes that can be split to create internal nodes in the TN structure by analyzing the spectrum of matricizations of its core tensors.

The overall greedy algorithm, named `Greedy-TN`, is summarized in Algorithm 1. In the remaining of this section, we describe the continuous optimization, weight transfer, best edge identification and node splitting procedures. For Problem 2.3.1, a natural stopping criterion for the greedy algorithm is when the maximum number of parameters is reached, but more sophisticated stopping criteria can be used. For example, the algorithm can be stopped once a given loss threshold is reached, which leads to an approximate solution to the problem of identifying the TN structure with the least number of parameters achieving a given loss threshold. For learning tasks (e.g., TN classifiers or tensor completion), the stopping criterion can be based on validation data (e.g., using early stopping).

Continuous Optimization. Assuming that the loss function $\mathcal{L}$ is continuous and differentiable, standard gradient-based optimization algorithms can be used to solve the inner optimization problem (line 13 of Algorithm 1). For example, in our experiments on compressing neural network layers (see Section 3.5) we use Adam [22]. For particular losses, more efficient optimization methods can be used: in our experiments on tensor completion and

Algorithm 1 `Greedy-TN`: Greedy algorithm for tensor network structure learning.

Input: Loss function $\mathcal{L} : \mathbb{R}^{d_1 \times \cdots \times d_p} \to \mathbb{R}$, splitting node threshold ε.

1: *// Initialize tensor network to a random rank one tensor and optimize loss function.*
2: $R_{i,j} \leftarrow 1$ for $1 \leq i < j \leq p$
3: Initialize core tensors $\boldsymbol{\mathcal{G}}^{(i)} \in \mathbb{R}^{R_{1,i} \times \cdots \times R_{i-1,i} \times d_i \times R_{i,i+1} \times \cdots \times R_{i,p}}$ randomly
4: $(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leftarrow$ `optimize` $\mathcal{L}(\mathrm{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$ w.r.t. $\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}$
5: **repeat**
6: $\quad (i, j) \leftarrow$ `find-best-edge`$(\mathcal{L}, (\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$
7: $\quad$ *// Weight transfer*
8: $\quad \hat{\boldsymbol{\mathcal{G}}}^{(k)} \leftarrow \boldsymbol{\mathcal{G}}^{(k)}$ for $k \in [p] \setminus \{i, j\}$
9: $\quad R_{i,j} \leftarrow R_{i,j} + 1$
10: $\quad \hat{\boldsymbol{\mathcal{G}}}^{(i)} \leftarrow$ `add-slice`$(\boldsymbol{\mathcal{G}}^{(i)}, j)$ *// add new slice to the jth mode of $\boldsymbol{\mathcal{G}}^{(i)}$*
11: $\quad \hat{\boldsymbol{\mathcal{G}}}^{(j)} \leftarrow$ `add-slice`$(\boldsymbol{\mathcal{G}}^{(j)}, i)$ *// add new slice to the ith mode of $\boldsymbol{\mathcal{G}}^{(j)}$*
12: $\quad$ *// Optimize new tensor network structure*
13: $\quad (\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leftarrow$ `optimize` $\mathcal{L}(\mathrm{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$ from init. $\hat{\boldsymbol{\mathcal{G}}}^{(1)}, \cdots, \hat{\boldsymbol{\mathcal{G}}}^{(p)}$
14: $\quad$ *// Add internal nodes if possible (number of cores p may be increased after this step)*
15: $\quad (\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leftarrow$ `split-nodes`$((\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}), \varepsilon)$
16: **until** Stopping criterion

tensor decomposition, we use the Alternating Least-Squares (ALS) [23, 6] algorithm which consists in alternatively solving the minimization problem w.r.t. one of the core tensors while keeping the other ones fixed until convergence.

Weight Transfer. A key idea of our approach is to restart the continuous optimization process from the previous iteration of the greedy algorithm: we initialize the new slices of the two core tensors connected by the incremented edge to values close to 0, while keeping all the other parameters of the TN unchanged (line 8-11 of Algorithm 1). The detailed procedure of `add-slice` is described in Algorithm 2. For example, for a tensor network of order 4, increasing the rank of the edge $(1, 2)$ by 1 is done by adding a slice of size $d_1 \times R_{1,3} \times R_{1,4}$ (resp. $d_2 \times R_{2,3} \times R_{2,4}$) to the second mode of $\boldsymbol{\mathcal{G}}^{(1)}$ (resp. first mode of $\boldsymbol{\mathcal{G}}^{(2)}$). After this operation, the new shape of $\boldsymbol{\mathcal{G}}^{(1)}$ will be $d_1 \times (R_{1,2} + 1) \times R_{1,3} \times R_{1,4}$ and the one of $\boldsymbol{\mathcal{G}}^{(2)}$ will be $(R_{1,2} + 1) \times d_2 \times R_{2,3} \times R_{2,4}$. The following proposition shows that if these slices were initialized exactly to 0, the resulting TN would represent exactly the same tensor as the original one. In practice, we initialize the slices randomly with small values to break symmetries that could constrain the continuous optimization process.

Algorithm 2 `add-slice`$(\boldsymbol{\mathcal{G}}^{(i)}, j)$

Input: Core tensor to add new slice to $\boldsymbol{\mathcal{G}}^{(i)}$, mode to add new slice j.

1: **if** $j > i$ **then**

2: $\hat{\boldsymbol{\mathcal{G}}}^{(i)} \leftarrow \texttt{reshape}\Big(\begin{bmatrix}(\boldsymbol{\mathcal{G}}^{(i)})_{(j)} \\ \mathbf{-0-}\end{bmatrix}, (R_{1,i} \times \cdots \times R_{i-1,i} \times d_i \times R_{i,i+1} \times \cdots \times R_{i,j-1} \times (R_{i,j} + 1) \times R_{i,j+1} \times \cdots \times R_{i,p})\Big)$

3: **else if** $j < i$ **then**

4: $\hat{\boldsymbol{\mathcal{G}}}^{(i)} \leftarrow \texttt{reshape}\Big(\begin{bmatrix}(\boldsymbol{\mathcal{G}}^{(i)})_{(j)} \\ \mathbf{-0-}\end{bmatrix}, (R_{1,i} \times \cdots \times R_{j-1,i} \times (R_{j,i} + 1) \times R_{j+1,i} \times \cdots \times R_{i-1,i} \times d_i \times R_{i,i+1} \times \cdots \times R_{i,p})\Big)$

Output: $\hat{\boldsymbol{\mathcal{G}}}^{(i)}$

Proposition 2.4.1. *Let $\boldsymbol{\mathcal{G}}^{(k)} \in \mathbb{R}^{R_{1,k} \times \cdots \times R_{k-1,k} \times d_k \times R_{k,k+1} \times \cdots \times R_{k,p}}$ for $k \in [p]$ be the core tensors of a tensor network and let $1 \leq i < j \leq p$. Let $\tilde{R}_{i'j'} = R_{i',j'} + 1$ if $(i', j') = (i, j)$ and $R_{i',j'}$ otherwise, and define the core tensors $\tilde{\boldsymbol{\mathcal{G}}}^{(k)} \in \mathbb{R}^{\tilde{R}_{1,k} \times \cdots \times \tilde{R}_{k-1,k} \times d_k \times \tilde{R}_{k,k+1} \times \cdots \times \tilde{R}_{k,p}}$ for $k \in [p]$ by*

$$(\tilde{\boldsymbol{\mathcal{G}}}^{(i)})_{(j)} = \begin{bmatrix}(\boldsymbol{\mathcal{G}}^{(i)})_{(j)} \\ \mathbf{-0-}\end{bmatrix}, \ (\tilde{\boldsymbol{\mathcal{G}}}^{(j)})_{(i)} = \begin{bmatrix}(\boldsymbol{\mathcal{G}}^{(j)})_{(i)} \\ \mathbf{-0-}\end{bmatrix} \ \text{and} \ \ \tilde{\boldsymbol{\mathcal{G}}}^{(k)} = \boldsymbol{\mathcal{G}}^{(k)} \ \text{for} \ k \in [p] \setminus \{i, j\}$$

where $\mathbf{0}$ denotes a row vector of zeros of the appropriate size in each block matrix.

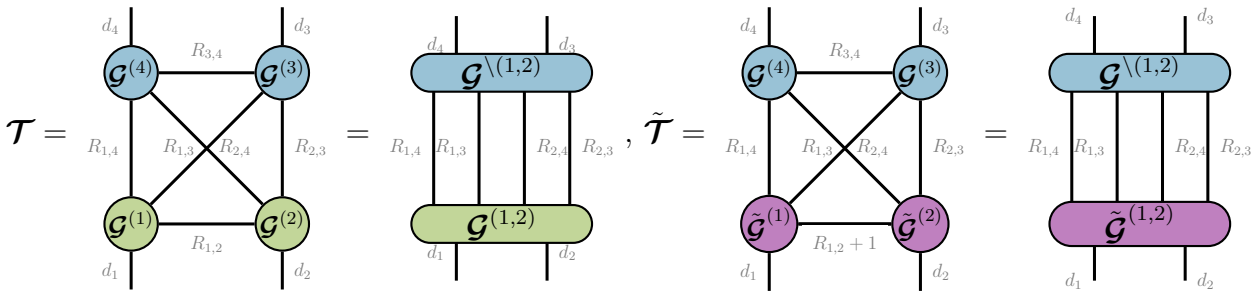
Then, the core tensors $\tilde{\boldsymbol{\mathcal{G}}}^{(k)}$ correspond to the same tensor network as the core tensors $\boldsymbol{\mathcal{G}}^{(k)}$, i.e., $TN(\tilde{\boldsymbol{\mathcal{G}}}^{(1)}, \cdots, \tilde{\boldsymbol{\mathcal{G}}}^{(p)}) = TN(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)})$.

Proof. Let $\boldsymbol{\mathcal{T}} = \mathrm{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)})$ and $\tilde{\boldsymbol{\mathcal{T}}} = \mathrm{TN}(\tilde{\boldsymbol{\mathcal{G}}}^{(1)}, \cdots, \tilde{\boldsymbol{\mathcal{G}}}^{(p)})$. We first split the TN $\boldsymbol{\mathcal{T}}$ and $\tilde{\boldsymbol{\mathcal{T}}}$ in two parts by isolating the ith and jth nodes from the other nodes of the TN:

- let $\boldsymbol{\mathcal{G}}^{\setminus(i,j)} \in \mathbb{R}^{\prod_{k \neq i,j} d_k \times \prod_{k \neq j} R_{i,k} \times \prod_{k \neq i} R_{j,k}}$ be the tensor obtained by contracting all the core tensors of $\boldsymbol{\mathcal{T}}$ except for the ith and jth cores,
- let $\boldsymbol{\mathcal{G}}^{(i,j)} \in \mathbb{R}^{d_i \times d_j \times \prod_{k \neq j} R_{i,k} \times \prod_{k \neq i} R_{j,k}}$ be the tensor obtained by contracting $\boldsymbol{\mathcal{G}}^{(i)}$ and $\boldsymbol{\mathcal{G}}^{(j)}$ along their shared index (i.e., the jth mode of the ith core is contracted with the jth mode of the ith core),
- let $\tilde{\boldsymbol{\mathcal{G}}}^{(i,j)} \in \mathbb{R}^{d_i \times d_j \times \prod_{k \neq j} R_{i,k} \times \prod_{k \neq i} R_{j,k}}$ be the tensor obtained by contracting $\tilde{\boldsymbol{\mathcal{G}}}^{(i)}$ and $\tilde{\boldsymbol{\mathcal{G}}}^{(j)}$ along their shared index.

One can check that the contraction between the last two modes of $\boldsymbol{\mathcal{G}}^{\setminus(i,j)}$ and the last two modes of $\boldsymbol{\mathcal{G}}^{(i,j)}$ is a reshaping of $\boldsymbol{\mathcal{T}}$. Similarly, since $\tilde{\boldsymbol{\mathcal{G}}}^{(k)} = \boldsymbol{\mathcal{G}}^{(k)}$ for any k distinct from i and j, the contraction over the last two modes of $\boldsymbol{\mathcal{G}}^{\setminus(i,j)}$ and $\tilde{\boldsymbol{\mathcal{G}}}^{(i,j)}$ gives rise to the same reshaping of $\tilde{\boldsymbol{\mathcal{T}}}$. Therefore to prove $\boldsymbol{\mathcal{T}} = \tilde{\boldsymbol{\mathcal{T}}}$, it suffices to show that $\boldsymbol{\mathcal{G}}^{(i,j)} = \tilde{\boldsymbol{\mathcal{G}}}^{(i,j)}$.

This argument is illustrated in the tensor network diagrams below for the particular case of $p = 4$, $i = 1$, $j = 2$.



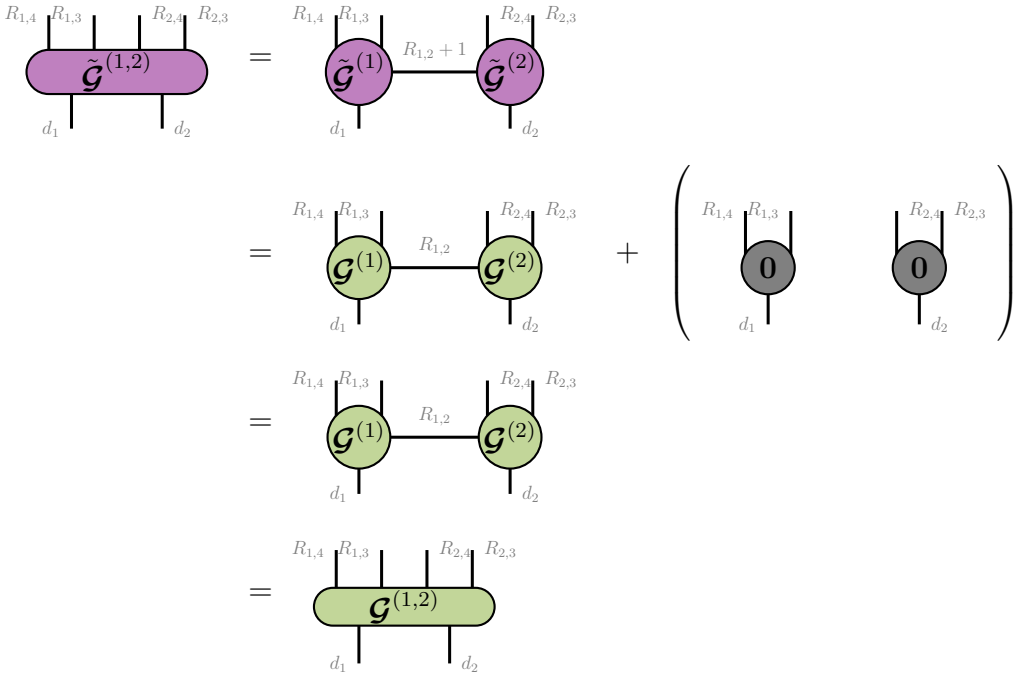
Let $(\boldsymbol{\mathcal{G}}^{(i,j)})_{[1,3]}$ (resp. $(\tilde{\boldsymbol{\mathcal{G}}}^{(i,j)})_{[1,3]}$) be the matricization of $\boldsymbol{\mathcal{G}}^{(i,j)}$ (resp. $\tilde{\boldsymbol{\mathcal{G}}}^{(i,j)}$) with modes 1 and 3 as the rows and modes 2 and 4 as the columns. We have

$$(\tilde{\boldsymbol{\mathcal{G}}}^{(i,j)})_{[1,3]} = \tilde{\boldsymbol{\mathcal{G}}}^{(i)\top}_{\langle j \rangle} \tilde{\boldsymbol{\mathcal{G}}}^{(j)}_{\langle i \rangle} = \boldsymbol{\mathcal{G}}^{(i)\top}_{\langle j \rangle} \boldsymbol{\mathcal{G}}^{(j)}_{\langle i \rangle} + \mathbf{0}\mathbf{0}^\top = (\boldsymbol{\mathcal{G}}^{(i,j)})_{[1,3]},$$

where the notation $\boldsymbol{\mathcal{A}}^{(m)}_{\langle n \rangle}$ denotes the matrix obtained by transposing the mth mode of $\boldsymbol{\mathcal{A}}^{(m)}$ to the first mode and matricizing the resulting tensor along the nth mode if $m < n$ and along the $(n+1)$th mode if $m > n$*. It then follows that $\boldsymbol{\mathcal{G}}^{(i,j)} = \tilde{\boldsymbol{\mathcal{G}}}^{(i,j)}$, hence $\boldsymbol{\mathcal{T}} = \tilde{\boldsymbol{\mathcal{T}}}$.

Continuing with the particular case of $p = 4$, $i = 1$, $j = 2$, the second part of the proof can be illustrated by the following tensor network diagrams.

*For example, if $\boldsymbol{\mathcal{A}}^{(2)} \in \mathbb{R}^{n_1 \times d \times n_3 \times n_4}$, $\boldsymbol{\mathcal{A}}^{(2)}_{\langle 3 \rangle} \in \mathbb{R}^{n_3 \times d n_1 n_4}$ is obtained by transposing $\boldsymbol{\mathcal{A}}^{(2)}$ in a tensor of size $d \times n_1 \times n_3 \times n_4$ and matricizing the resulting tensor along the 3rd mode. Similarly, $\boldsymbol{\mathcal{A}}^{(2)}_{\langle 1 \rangle} \in \mathbb{R}^{n_1 \times d n_3 n_4}$ is obtained by transposing $\boldsymbol{\mathcal{A}}^{(2)}$ in a tensor of size $d \times n_1 \times n_3 \times n_4$ and matricizing the resulting tensor along the 2nd mode. Note that $\boldsymbol{\mathcal{A}}^{(m)}_{\langle n \rangle}$ is always a column-wise permutation of the classical matricization $\boldsymbol{\mathcal{A}}^{(m)}_{(n)}$.



□

The weight transfer mechanism leads to a more efficient and robust continuous optimization by transferring the knowledge from each greedy iteration to the next and avoiding re-optimizing the loss function from a random initialization at each iteration. An ablation study showing the benefits of weight transfer is provided in section 3.2.1.

Best Edge Selection. As mentioned previously, we propose to optimize the inner minimization problem in Eq. 2.3.1 using iterative algorithms, namely gradient based algorithms or ALS depending on the loss function $\mathcal{L}$. In order to identify the most promising edge to increase the rank by 1 (line 6 of Algorithm 1), a reasonable heuristic consists in optimizing the loss for a few epochs/iterations for each possible edge and selecting the edge which led to the steepest decrease in the loss. One drawback of this approach is its computational complexity for example, when using ALS, each iteration requires solving p least-squares problem with $d_i \prod_{k \neq i} R_{i,k}$ unknowns for $i \in [p]$. We propose to reduce the complexity of the exploratory optimization in the best edge identification heuristic by only optimizing the loss function w.r.t. the new slices of the core tensors. Thus, at each iteration of the greedy algorithm, for each possible edge to increase, we transfer the weights from the previous greedy iteration,

optimize only w.r.t. the new slices for a small number of iteration, and choose the edge which led to the steepest decrease of the loss. For ALS, this reduces the complexity of each iteration to the one of solving 2 least-squares problems with $d_i \prod_{k\in[p]\setminus\{i,j\}} R_{i,k}$ and $d_j \prod_{k\in[p]\setminus\{i,j\}} R_{i,k}$ unknowns, respectively, where (i, j) is the edge being considered in the search. When using gradient-based optimization algorithms, the same approach is used where the gradient is only computed for (and back-propagated through) the new slices. The overall pseudo-code for identifying the best edge is described in Algorithm 3.

It is worth mentioning that the greedy algorithm can seamlessly incorporate structural constraints by restricting the set of edges considered when identifying the best edge for a rank increment. For example, it can be used to adaptively select the ranks of a TT or TR decomposition.

Algorithm 3 `find-best-edge`$(\mathcal{L}, (\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$

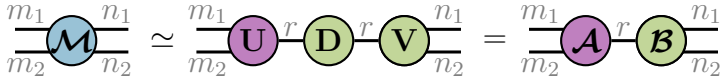
Input: Loss function $\mathcal{L}$, core tensors $\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}$.

best-loss $\leftarrow \infty$
for $i \leftarrow 1$ to p **do**
  for $j \leftarrow i+1$ to p **do**
    $\hat{\boldsymbol{\mathcal{G}}}^{(i)} \leftarrow$ `add-slice`$(\boldsymbol{\mathcal{G}}^{(i)}, j)$
    $\hat{\boldsymbol{\mathcal{G}}}^{(j)} \leftarrow$ `add-slice`$(\boldsymbol{\mathcal{G}}^{(j)}, i)$
    $(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(i-1)}, \hat{\boldsymbol{\mathcal{G}}}^{(i)}, \boldsymbol{\mathcal{G}}^{(i+1)}, \cdots, \boldsymbol{\mathcal{G}}^{(j-1)}, \hat{\boldsymbol{\mathcal{G}}}^{(j)}, \boldsymbol{\mathcal{G}}^{(j+1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leftarrow$
    `optimize` $\mathcal{L}(\mathrm{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(i-1)}, \hat{\boldsymbol{\mathcal{G}}}^{(i)}, \boldsymbol{\mathcal{G}}^{(i+1)}, \cdots, \boldsymbol{\mathcal{G}}^{(j-1)}, \hat{\boldsymbol{\mathcal{G}}}^{(j)}, \boldsymbol{\mathcal{G}}^{(j+1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$
    w.r.t. new slices in $\hat{\boldsymbol{\mathcal{G}}}^{(i)}$ and $\hat{\boldsymbol{\mathcal{G}}}^{(j)}$
    loss $= \mathcal{L}(\mathrm{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(i-1)}, \hat{\boldsymbol{\mathcal{G}}}^{(i)}, \boldsymbol{\mathcal{G}}^{(i+1)}, \cdots, \boldsymbol{\mathcal{G}}^{(j-1)}, \hat{\boldsymbol{\mathcal{G}}}^{(j)}, \boldsymbol{\mathcal{G}}^{(j+1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$
    if loss $<$ best-loss **then**
      best-edge $= (i, j)$
      best-loss $=$ loss

Output: best-edge

Internal Nodes. Lastly, we design a simple approach for the greedy algorithm to add internal nodes to the TN structure relying on a common technique used in TN methods to split a node into two new nodes using truncated SVD (see, e.g., Fig. 7.b in [42]). To illustrate this technique, let $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{m_1 \times m_2 \times n_1 \times n_2}$ be the core tensor associated with a node in a TN we want to split into two new nodes $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{m_1 \times m_2 \times r}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{n_1 \times n_2 \times r}$: the first two legs of $\boldsymbol{\mathcal{A}}$ (resp. $\boldsymbol{\mathcal{B}}$) will be connected to the core tensors that were connected to $\boldsymbol{\mathcal{M}}$ by its first two

legs (resp. last two legs), and the third leg of $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ will be connected together. This is achieved by taking the rank r truncated SVD of $(\boldsymbol{\mathcal{M}})_{[1,2]} \simeq \mathbf{U}\mathbf{D}\mathbf{V}^\top \in \mathbb{R}^{m_1 m_2 \times n_1 n_2}$ (the matricization of $\boldsymbol{\mathcal{M}}$ having modes 1 and 2 as rows and modes 3 and 4 as columns), and letting $\boldsymbol{\mathcal{A}}_{(3)} = \mathbf{U}^\top \in \mathbb{R}^{r \times m_1 m_2}$ and $\boldsymbol{\mathcal{B}}_{(3)} = \mathbf{D}\mathbf{V}^\top \in \mathbb{R}^{r \times n_1 n_2}$. If the truncated SVD is exact, the resulting TN will represent exactly the same tensor as the one before splitting the core $\boldsymbol{\mathcal{M}}$. This node splitting procedure is illustrated in the following TN diagram.



In order to allow the greedy algorithm to learn TN structures with internal nodes, at the end of each greedy iteration, we perform an SVD of each matricization of $\boldsymbol{\mathcal{G}}^{(k)}$ for $k \in [p]$ (line 15 of Algorithm 1). For each matricization, we split the corresponding node only if there are enough singular values below a given threshold ε in order for the new TN structure to have less parameters than the initial one. Algorithm 4 illustrates the pseudo-code for this procedure. While this approach may seem computationally heavy, the cost of these SVDs is negligible w.r.t. the continuous optimization step which dominates the overall complexity of the greedy algorithm.

Algorithm 4 `split-nodes`$((\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}), \varepsilon)$

Input: Core tensors $\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}$, splitting node threshold ε.

for $i \leftarrow 1$ to p **do**
  for every bi-partition (M, N) of $[p]$ **do**
    $\mathbf{U}, \mathbf{D}, \mathbf{V}^\top = \varepsilon$-`truncated-SVD`(`reshape`$(\boldsymbol{\mathcal{G}}^{(i)}, d_i \prod_{j \in M} R_{i,j} \times \prod_{j \in N} R_{i,j})$)
    $\hat{R} \leftarrow$ rank of the ε-`truncated-SVD`
    if splitting node $\boldsymbol{\mathcal{G}}^{(i)}$ reduces the number of parameters **then**
      $\forall j \in [p]$, let $\tilde{R}_{i,j} = R_{i,j}$ if $j \in M$ and 1 otherwise.
      $\boldsymbol{\mathcal{G}}^{(i)} \leftarrow$ `reshape`$(\mathbf{U}, \tilde{R}_{1,i} \times \cdots \times \tilde{R}_{i-1,i} \times d_i \times \tilde{R}_{i,i+1} \times \cdots \times \tilde{R}_{i,p} \times \hat{R})$
      $\forall j \in [p]$, let $\tilde{R}_{i,j} = R_{i,j}$ if $j \in N$ and 1 otherwise.
      $\boldsymbol{\mathcal{G}}^{(p+1)} \leftarrow$ `reshape`$(\mathbf{D}\mathbf{V}^\top, \tilde{R}_{1,i} \times \cdots \times \tilde{R}_{i-1,i} \times \hat{R} \times \tilde{R}_{i,i+1} \times \cdots \times \tilde{R}_{i,p} \times 1))$
      for $j \in [p] \setminus \{i\}$ **do**
        $\boldsymbol{\mathcal{G}}^{(j)} \leftarrow$ `reshape`$(\boldsymbol{\mathcal{G}}^{(j)}, R_{1,j} \times \cdots \times R_{j-1,j} \times d_j \times R_{j,j+1} \times \cdots \times R_{j,p} \times 1))$
      $p \leftarrow p + 1$

Output: $(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)})$

2.4.1. Computational Complexity

The overall time complexity of `Greedy-TN` is dominated by the whose complexity is in $\mathcal{O}(p^2T + pd^2R^{2p})$ where T is the time complexity of optimizing the loss function w.r.t. one of the core tensors. The first term corresponds to the `find-best-edge` subroutine and the second one corresponds to the `split-nodes` sub-routine. For example, when optimizing a squared error loss with SGD, T is in $\mathcal{O}(R^{p-1}d^p)$ where $R = \max_{i,j} R_{i,j}$ is the maximum rank in the tensor network and $d = \max_i d_i$ is the maximum dangling dimension. Thus, in this case, when $R \leq d$ the overall complexity is dominated by the `find-best-edge` subroutine.

Chapter 3

Results and discussion

3.1. Introduction

In this chapter we evaluate the performance of `Greedy-TN` on a number of machine learning tasks, namely tensor decomposition, tensor completion, image compression, and neural network compression. We compare our method to classical tensor decomposition methods as well as the recently proposed genetic algorithm method for tensor network decomposition [26].

3.2. Tensor decomposition

We first consider a tensor decomposition task, where we generate normally distributed random target tensors of size $7 \times 7 \times 7 \times 7 \times 7$ with the four TN structures shown in Figure 3.1.

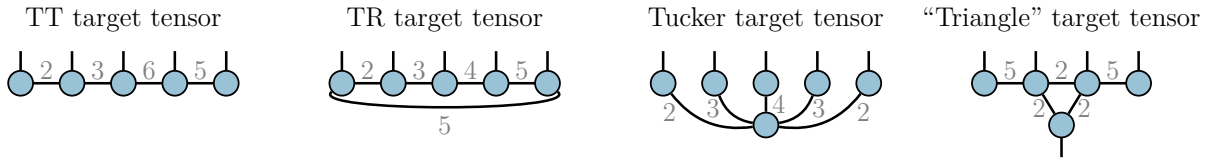


Figure 3.1. Tensor network structures for tensor decomposition.

For each of the four TN structures in Figure 3.1 we generate 100 random tensor networks. We then run `Greedy-TN` with the loss function $\mathcal{L}$ set to the squared error loss, until it recovers an almost exact decomposition (stopping when the relative error falls below 10^{-6}). We compare `Greedy-TN` with CP, Tucker and TT decomposition (using the implementations

from the TensorLy python package [24]) of increasing rank as baselines (we use uniform ranks for Tucker and TT). We also include a simple random walk baseline based on `Greedy-TN`, where the edge for the rank increment is chosen at random at each iteration.

Reconstruction errors averaged over the 100 runs are reported in Figure 3.2, where we see that the greedy algorithm outperforms all baselines for the the four target tensors. Notably, `Greedy-TN` outperforms TT/Tucker even on the TT/Tucker targets. This is because the rank of the TT and Tucker targets are not uniform and `Greedy-TN` is able to adaptively set different ranks to achieve the best compression ratio. Furthermore, `Greedy-TN` is able to recover the exact TN structure of the triangle target tensor on almost every run. Lastly, we observe that the internal node search of `Greedy-TN` is only beneficial on the Tucker target tensor, which is expected due to the absence of internal nodes in the other target TN structures.

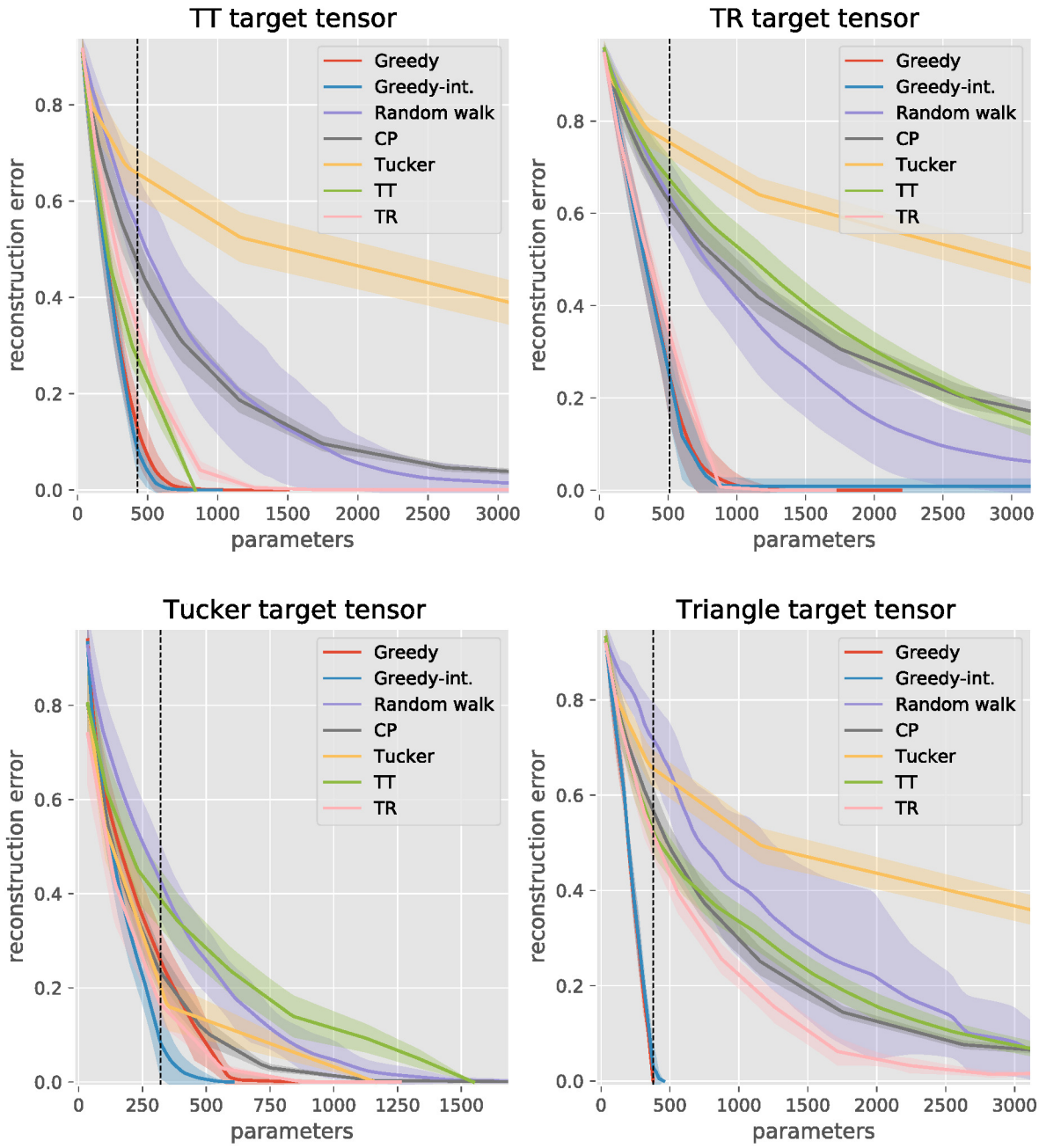


Figure 3.2. Evaluation of `Greedy-TN` on tensor decomposition. Curves represent the reconstruction error averaged over 100 runs, shaded areas correspond to standard deviations and the vertical line represents the number of parameters of the target TN. `Greedy` corresponds to `Greedy-TN` without the search for internal nodes (`split-nodes` subroutine, line 15 of Algorithm 1) while `Greedy-int.` includes this search.

In Figure 3.3, we show the most frequent tensor network structure recovered by the greedy algorithm for each of the four targets used in the experiment (see Figure 3.1). We see that

`Greedy-TN` and `Greedy-int` always recover the same structure except for the Tucker target, where `Greedy-TN` finds the best TN structure without internal nodes to approximate the target. We also observe that the greedy algorithm recovers the correct TN structure for all targets most of the time, except for the TR target.

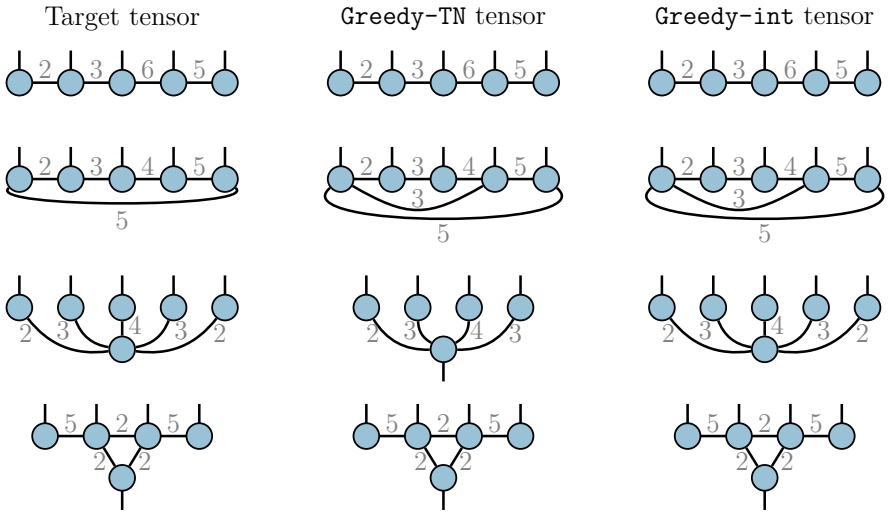


Figure 3.3. Most common tensor network structure returned by `Greedy-TN` and `Greedy-int` over the 100 runs of the tensor decomposition experiment.

As an illustration of the running time, for the TR target, one iteration of `Greedy-TN` takes approximately 0.91 second on average without the internal node search and 1.18 seconds with the search.

This experiment showcases the potential cost of decomposition model mis-specification: both CP and Tucker struggle to efficiently approximate most target tensors. Interestingly, even the random walk outperforms CP and Tucker on the TR target tensor.

3.2.1. Weight transfer benefits

Here, we study if transferring the weights at each step leads to better results. We randomly generate 50 target tensors of size $7 \times 7 \times 7 \times 7 \times 7$ with a TT structure of rank $6, 3, 6, 5$. We run `Greedy-TN` with and without weight transfer until convergence.

The results are shown in Figure 3.4, where we see that using the weight transfer mechanism results in a lower loss with the same number of parameters, compared to using a random

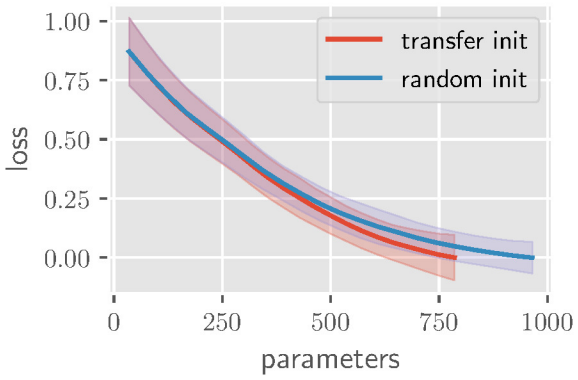


Figure 3.4. Comparison of `Greedy-TN` with and without weight transfer on a TT structure decomposition task. Curves represent the reconstruction error averaged over 50 runs, and shaded areas correspond to standard deviations.

initialization at each greedy step. This shows that transferring the knowledge from the previous greedy iterations leads to a better initialization for the continuous optimization.

3.3. Tensor completion

We compare `Greedy-TN` with the TT and TR alternating least square algorithms proposed in [44] and the CP and Tucker decomposition algorithms from Tensorly [24] on an image completion task.

We consider an experiment presented in [44]: the completion of an RGB image of Albert Einstein reshaped into a $6 \times 10 \times 10 \times 6 \times 10 \times 10 \times 3$ tensor (see [44] for details) where 10% of entries are randomly observed. The ranks of methods other than `Greedy-TN` are successively increased by one until the number of parameters gets larger than 25,000 (we use uniform ranks for TT, TR and Tucker*).

The relative errors as a function of number of parameters are reported in Figure 3.5 (left) where we see that `Greedy-TN` outperforms all methods. The best recovered images for all methods are shown in Figure 3.5 (right) along with the original image and observed pixels. The best recovery error (9.45%) is achieved by `Greedy-TN` at iteration 42 with 21,375 parameters. The second best recovery error (10.83%) is obtained by `TR-ALS` at rank 18 with

*For Tucker, the completion is performed on the original image rather than the tensor reshaping since the number of parameters of Tucker grows exponentially, leading to very poor results on the tensorized image.

17,820 parameters. At iteration 31, `Greedy-TN` already recovers an image with an error of 10.60% with 10,096 parameters, which is better than the best result of `TR-ALS` both in terms of parameters and relative error.

The images recovered at each iteration of `Greedy-TN` along with the relative test error and number of parameters for each step, are shown in Figures 3.6 and 3.7.

In this experiment, the total running time of `Greedy-TN` is comparable to the one of `TR-ALS` (on the order of hours), which is larger than the one of the other three methods.

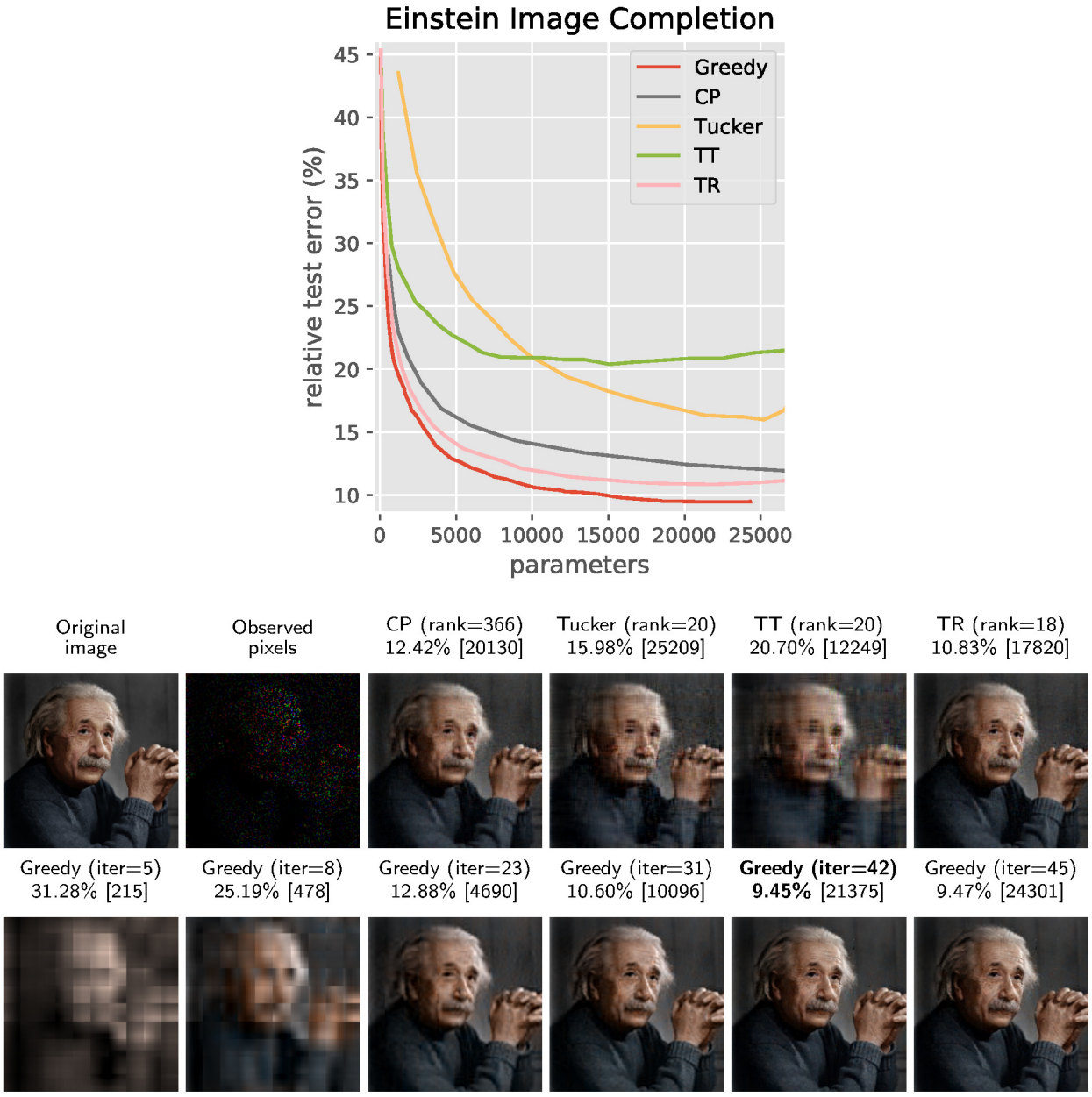


Figure 3.5. Image completion with 10% of the entries randomly observed. (top) Relative reconstruction error. (bottom) Best recovered images for CP, Tucker, TT and TR, and 6 recovered images at different iteration of greedy (image title: RSE% [number of parameters]).

Figure 3.6. Solutions found by `Greedy-TN` for the Einstein image completion experiments, labeled by number of parameters and relative test error w.r.t. the full image. [**continued on next page**]

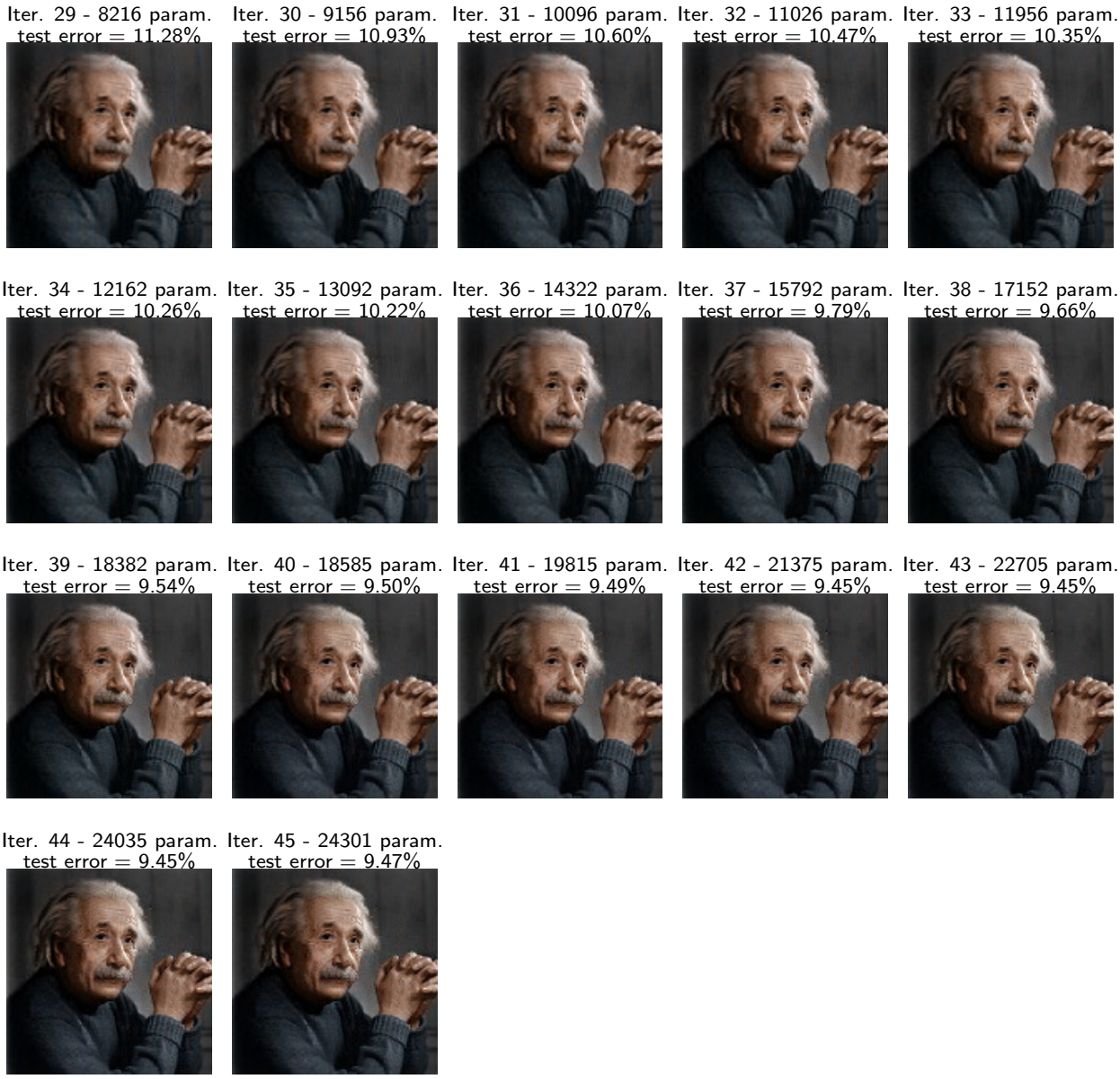


Figure 3.7. Solutions found by `Greedy-TN` for the Einstein image completion experiments, labeled by number of parameters and relative test error w.r.t. the full image. [**continued from previous page**]

3.4. Image compression

In this experiment, we compare `Greedy-TN` with the genetic algorithm (GA) for TN decomposition recently introduced in [26], denoted by GA(rank=6) and GA(rank=7) where the rank is a hyper-parameter controlling the trade-off between accuracy and compression ratio (the results of TT and TR, which are worst than GA, are available in Table 3 in [26]).

Following [26], we select 10 images of size 256×256 from the LIVE dataset [39], tensorize each image to an order-8 tensor of size 4^8 and run `Greedy-TN` to decompose each tensor using a squared error loss.

`Greedy-TN` is stopped when the lowest RSE reported in [26] is reached. In Table 1, we report the log compression ratio and root square error averaged over 50 random seeds. For all images, our method results in a higher compression ratio compared to GA(rank=7). Moreover, for images 1 to 9 our method even outperforms GA(rank=6) by achieving both higher compression ratios and significantly lower RSE. For image 0, setting the greedy stopping criterion to the RSE of GA(rank=6), `Greedy-TN` also achieves a higher compression ratio than GA(rank=6): 1.085(0.128). Our method is also orders of magnitude faster—few minutes compared to several hours for GA.

Table 3.1. Log compression ratio and RSE for 10 different images selected from the LIVE dataset.

Image	**Log compression ratio CR↑ and (RSE↓)** ±std						
	`Greedy-TN`	GA(rank=6)	GA(rank=7)	Tensor Train		Tensor Ring	
0	**0.715**(**0.105**)±0.152(0.005)	**0.901**(**0.137**)	0.660(0.115)	0.582(0.142)	0.325(0.115)	0.469(0.141)	0.457(0.127)
1	**2.313**(**0.150**)±0.189(0.005)	1.352(0.158)	1.159(0.155)	1.210(0.170)	1.137(0.166)	1.216(0.187)	0.824(0.155)
2	**2.139**(**0.167**)±0.127(0.004)	1.452(0.176)	1.268(0.171)	1.148(0.187)	0.898(0.179)	1.231(0.206)	1.022(0.182)
3	**3.009**(**0.185**)±0.088(0.002)	1.649(0.193)	1.476(0.189)	1.140(0.191)	1.265(0.206)	1.416(0.211)	1.074(0.191)
4	**0.874**(**0.111**)±0.129(0.005)	0.859(0.152)	0.621(0.121)	0.527(0.156)	0.408(0.143)	0.403(0.153)	0.372(0.141)
5	**3.668**(**0.080**)±0.103(0.001)	1.726(0.087)	1.548(0.083)	1.471(0.087)	1.531(0.083)	1.471(0.088)	1.388(0.085)
6	**2.205**(**0.097**)±0.171(0.004)	1.332(0.110)	1.141(0.104)	1.471(0.113)	1.088(0.101)	1.212(0.124)	1.052(0.102)
7	**2.132**(**0.115**)±0.202(0.002)	1.573(0.126)	1.406(0.120)	1.030(0.139)	1.179(0.142)	1.112(0.145)	0.970(0.125)
8	**3.634**(**0.080**)±0.142(0.001)	1.679(0.085)	1.505(0.081)	1.493(0.082)	1.493(0.082)	1.387(0.085)	1.357(0.084)
9	**1.669**(**0.174**)±0.202(0.002)	1.164(0.194)	0.966(0.185)	0.994(0.227)	0.774(0.190)	0.836(0.200)	0.916(0.226)

3.5. Compressing neural networks

In this section we apply our algorithm to compress a neural network with one hidden layer on the MNIST dataset. Following [30] the hidden layer weight is of size 1024×1024 which we represent as a fifth-order tensor of size $16 \times 16 \times 16 \times 16 \times 16$. We use `Greedy-TN` with loss function $\mathcal{L}$ set as cross-entropy to train the tensor network representing the hidden layer weight matrix alongside the output layer weights end-to-end. We select the best edge

for the rank increment using the validation performance on a separate random split of the train dataset with $5,000$ images.

In Figure 3.8, we report the train and test accuracies of the TT based method introduced in [30] as well as a TR tensorized model for uniform ranks 1 to 8 and `Greedy-TN` (it is worth noting that we use our own implementation of the TT method with dropout and achieve higher accuracies than the ones reported in [30]). For every model size, our method reaches higher accuracy. The best test accuracy of `Greedy-TN` is 98.74% with 15,050 parameters, while TT reaches its best accuracy of 98.46% with 14,602 parameters, and TR achieves its best accuracy of 98.42% with 14,154 parameters. At iteration 10, `Greedy-TN` already achieves an accuracy of 98.46% with only 12,266 parameters.

The running time of each iteration of `Greedy-TN` is comparable with training one tensorized neural network with TT or TR.

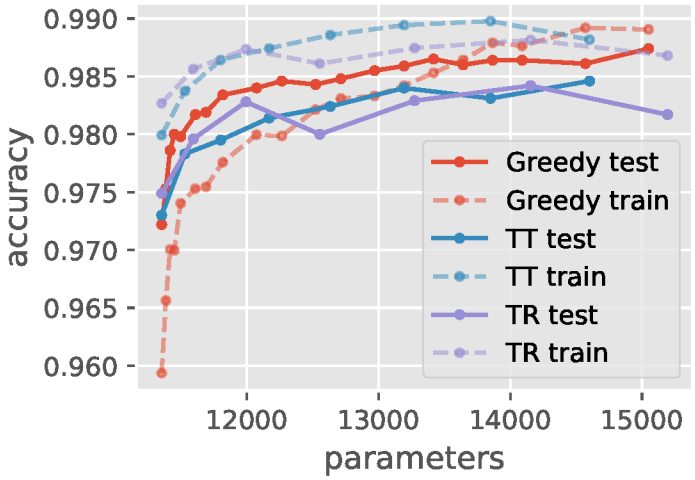


Figure 3.8. Train and test accuracies on the MNIST dataset for different model sizes.

Implementation details. We use PyTorch [34] and the NCON function [37] to implement `Greedy-TN`. For the continuous optimization step, we use the Adam [22] optimizer with a learning rate of 10^{-3} and a batch size of 256 for 50 epochs for compressing neural network, and we use ALS for the other experiments (ALS is stopped when convergence is reached). The number of iterations/epochs for the best edge identification is set to 2 for tensor decomposition, 5 for image compression and 10 for image completion and compressing neural networks. The

singular values threshold for the internal node search is set to $\varepsilon = 10^{-5}$. In all experiments except the tensor decomposition on the Tucker target, the internal node search did not lead to any improvement of the results. All experiments were performed on a single 32GB V100 GPU.

Chapter 4

Conclusion and future work

In this thesis we introduced a greedy algorithm to jointly optimize an arbitrary loss function and efficiently search the space of TN structures and ranks to adaptively find parameter efficient TN structures from data. Our experimental results show that `Greedy-TN` outperforms common methods tailored for specific decomposition models on tensor completion, image compression, and neural network compression tasks.

Even though `Greedy-TN` is orders of magnitude faster than the genetic algorithm introduced in [26], its computational complexity can still be limiting in some scenarios such as compressing neural networks with evergrowing number of parameters. Therefore, scaling up the method to discover TN structures suited for efficient compression of larger neural network models is a future direction we wish to discover.

In addition, the greedy algorithm may converge to locally optimal TN structures. And so, future work includes exploring more efficient discrete optimization techniques to solve the upper-level discrete optimization problem, `Greedy-TN` is not optimal as it does not backtrack, an interesting direction to explore is different discrete optimization methods such as A* with a carefully designed heuristic to reach better solutions.

References

[1] Mohammad Taha Bahadori, Qi Rose Yu, and Yan Liu. Fast multivariate spatio-temporal analysis via low rank tensor learning. In *Advances in Neural Information Processing Systems*, pages 3491–3499, 2014.

[2] Jonas Ballani and Lars Grasedyck. Tree adaptive approximation in the hierarchical tensor format. *SIAM journal on scientific computing*, 36(4):A1415–A1431, 2014.

[3] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.

[4] A. Cichocki, R. Zdunek, A.H. Phan, and S.I. Amari. *Nonnegative Matrix and Tensor Factorizations*. Applications to Exploratory Multi-way Data Analysis and Blind Source Separation. Wiley, 2009.

[5] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.

[6] Pierre Comon, Xavier Luciani, and André LF De Almeida. Tensor decompositions, alternating least squares and other tales. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 23(7-8):393–405, 2009.

[7] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[8] David Elieser Deutsch. Quantum computational networks. *Proc. R. Soc. Lond. A*, 425(1868):73–90, 1989.

[9] Richard P Feynman. Quantum mechanical computers. *Foundations of physics*, 16(6):507–531, 1986.

[10] Silvia Gandy, Benjamin Recht, and Isao Yamada. Tensor completion and low-n-rank tensor recovery via convex optimization. *Inverse problems*, 27(2):025010, 2011.

[11] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

[12] Ivan Glasser, Nicola Pancotti, and J Ignacio Cirac. Supervised learning with generalized tensor networks. *arXiv preprint arXiv:1806.05964*, 2018.

[13] Lars Grasedyck and Sebastian Krämer. Stable als approximation in the tt-format for rank-adaptive tensor completion. *Numerische Mathematik*, 143(4):855–904, 2019.

[14] Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang. Unsupervised generative modeling using matrix product states. *Physical Review X*, 8(3):031012, 2018.

[15] Meraj Hashemizadeh, Michelle Liu, Jacob Miller, and Guillaume Rabusseau. Adaptive learning of tensor network structures. *arXiv preprint arXiv:2008.05437*, 2020.

[16] Meraj Hashemizadeh, Michelle Liu, Jacob Miller, and Guillaume Rabusseau. Adaptive tensor learning with tensor networks. In *First Workshop on Quantum Tensor Networks in Machine Learning at NeurIPS*, 2020.

[17] Kohei Hayashi, Taiki Yamaguchi, Yohei Sugawara, and Shin-ichi Maeda. Einconv: Exploring unexplored tensor decompositions for convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2019.

[18] Christopher J Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):45, 2013.

[19] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[20] Pavel Izmailov, Alexander Novikov, and Dmitry Kropotov. Scalable gaussian processes with billions of inducing inputs via tensor train decomposition. In *International Conference on Artificial Intelligence and Statistics*, pages 726–735, 2018.

[21] Valentin Khrulkov, Alexander Novikov, and Ivan Oseledets. Expressive power of recurrent neural networks. In *International Conference on Learning Representations*, 2018.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*, 2015.

[23] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[24] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *The Journal of Machine Learning Research*, 20(1):925–930, 2019.

[25] J. B. Kruskal. *Rank, Decomposition, and Uniqueness for 3-Way and n-Way Arrays*, page 7–18. North-Holland Publishing Co., NLD, 1989.

[26] Chao Li and Sun Sun. Evolutionary topology search for tensor network decomposition. In *International Conference on Machine Learning*, 2020.

[27] H. Lu, K.N. Plataniotis, and A. Venetsanopoulos. *Multilinear Subspace Learning: Dimensionality Reduction of Multidimensional Data*. CRC Press, 2013.

[28] Oscar Mickelin and Sertac Karaman. Tensor ring decomposition. *arXiv preprint arXiv:1807.02513*, 2018.

[29] Jacob Miller, Guillaume Rabusseau, and John Terilla. Tensor networks for language modeling. *arXiv preprint arXiv:2003.01039*, 2020.

[30] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

[31] Alexander Novikov, Anton Rodomanov, Anton Osokin, and Dmitry Vetrov. Putting MRFs on a tensor train. In *International Conference on Machine Learning*, pages 811–819, 2014.

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

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

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[35] Roger Penrose. Applications of negative dimensional tensors. *Combinatorial mathematics and its applications*, 1:221–244, 1971.

[36] David Perez-García, Frank Verstraete, Michael M Wolf, and J Ignacio Cirac. Matrix product state representations. *Quantum Information and Computation*, 7(5-6):401–430, 2007.

[37] Robert NC Pfeifer, Glen Evenbly, Sukhwinder Singh, and Guifre Vidal. Ncon: A tensor network contractor for matlab. *arXiv preprint arXiv:1402.0939*, 2014.

[38] Guillaume Rabusseau and Hachem Kadri. Low-rank regression with tensor responses. In *Advances in Neural Information Processing Systems*, pages 1867–1875, 2016.

[39] H.R. Sheikh, M.F. Sabir, and A.C. Bovik. A statistical evaluation of recent full reference image quality assessment algorithms. *IEEE Transactions on Image Processing*, 15(11):3440–3451, 2006.

[40] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

[41] E. Miles Stoudenmire. Learning relevant features of data with multi-scale tensor networks. *Quantum Science and Technology*, 3(3):034003, 2018.

[42] Edwin Stoudenmire and David J. Schwab. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, pages 4799–4807, 2016.

[43] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[44] Wenqi Wang, Vaneet Aggarwal, and Shuchin Aeron. Efficient low rank tensor ring completion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5697–5705, 2017.

[45] Senlin Xia, Huaijiang Sun, and Beijia Chen. A regularized tensor decomposition method with adaptive rank adjustment for compressed-sensed-domain background subtraction. *Signal Processing: Image Communication*, 62:149–163, 2018.

[46] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. *arXiv preprint arXiv:1707.01786*, 2017.

[47] Ke Ye and Lek-Heng Lim. Tensor network ranks. *arXiv preprint arXiv:1801.02662*, 2018.

[48] Rose Yu, Guangyu Li, and Yan Liu. Tensor regression meets gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 482–490, 2018.

[49] Qibin Zhao, Liqing Zhang, and Andrzej Cichocki. Bayesian cp factorization of incomplete tensors with automatic rank determination. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1751–1763, 2015.

[50] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*, 2016.

[51] H. Zhou, L. Li, and H. Zhu. Tensor regression with applications in neuroimaging data analysis. *Journal of the American Statistical Association*, 108(502):540–552, 2013.

[52] Ziwei Zhu, Xia Hu, and James Caverlee. Fairness-aware tensor-based recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1153–1162, 2018.