Adaptive Tensor Learning with Tensor Networks

Meraj Hashemizadeh[*]
Mila & DIRO
Université de Montréal

Michelle Liu[*]
Mila & DIRO
Université de Montréal

Jacob Miller[*]
Mila & DIRO
Université de Montréal

Guillaume Rabusseau[*]
CCAI chair - Mila & DIRO
Université de Montréal

Abstract

Tensor decomposition techniques have shown great successes in machine learning and data science by extending classical algorithms based on matrix factorization to multi-modal and multi-way data. However, there exist many tensor decomposition models (CP, Tucker, Tensor Train, etc.), and the rank of such a decomposition is typically a collection of integers rather than a unique number, making model and hyper-parameter selection a tedious and costly task. At the same time, tensor network methods are powerful tools developed in the physics community which have recently shown their potential for machine learning applications and offer a unifying view of the various tensor decomposition models. In this paper, we leverage the tensor network formalism to develop a generic and efficient adaptive algorithm for tensor learning. Our method is based on a simple greedy approach optimizing a differentiable loss function starting from a rank one tensor and successively identifying the most promising tensor network edges for small rank increments. Our algorithm can adaptively identify tensor network structures with small number of parameters that effectively optimize the objective function from data. The framework we introduce is very broad and encompasses many common tensor optimization problems. Experiments on tensor decomposition and tensor completion tasks with both synthetic and real-world data demonstrate the effectiveness of the proposed algorithm.

1 Introduction

Matrix factorization is ubiquitous in machine learning and data science and is at the backbone of many algorithms. Tensor decomposition techniques emerged as a powerful generalization of matrix factorization. They are particularly suited to handle high-dimensional multi-modal data and have been successfully applied in neuroimaging [44], signal processing [3, 32], spatio-temporal analysis [1, 30] and computer vision [20]. 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 exist 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 number of parameters and minimizing a given loss function.

[*]The ordering of authors is alphabetical and do not reflect the level of contribution of each author.

First Workshop on Quantum Tensor Networks in Machine Learning, 34th Conference on Neural Information Processing Systems (NeurIPS 2020).

We propose an adaptive tensor learning algorithm which is agnostic to decomposition model. Our approach relies on the *tensor network* formalism which has shown great success in the physics community [28, 7, 6] and has recently demonstrated its potential in machine learning for compressing models [23, 39, 8, 24, 15, 41], developing new insights on the expressiveness of deep neural networks [4, 16] and designing novel approaches to supervised [34, 9] and unsupervised [33, 11, 22] 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 introduce a general tensor learning framework unifying common problems including tensor decomposition, regression and completion. We show how this framework naturally generalizes beyond common decomposition models to arbitrary tensor networks. This leads us to define 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 combined with automatic differentiation and 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 the tensor network structure which is best suited for the task at hand from data.

Summary of the contributions We introduce a general framework for tensor learning along with a learning algorithm which is agnostic to decomposition models. The greedy algorithm we propose is conceptually simple and experiments on tensor decomposition and completion 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.

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 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 [38] an adaptive Tucker based algorithm is proposed for background substraction. In [42], 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 [10] a stable rank-adaptive alternating least square algorithm is introduced for completion in the TT format. 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 [19] where evolutionary algorithms are used to approximate the best tensor network structure to exactly decompose a given target tensor. In contrast with our work, [19] only considers the problem of tensor decomposition whereas we consider an arbitrary loss function over tensor parameters. Lastly, [12] propose to explore the space of tensor network structures for compressing neural networks, a rounding algorithm for general tensor networks is proposed in [21] and the notions of rank induced by arbitrary tensor networks are studied in [40].

2 Preliminaries

In this section, we present basic notions of tensor algebra and tensor networks. We start by introducing some notations. For any integer k we use $[k]$ to denote the set of integers from 1 to 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 of a tensor in the straightforward way.

Tensors and tensor networks We first recall basic definitions of tensor algebra; more details can be found in [17]. 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 $\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{T}} \rangle =$

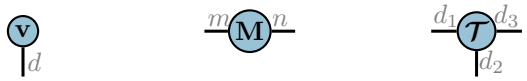
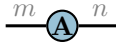


Figure 1: Tensor network representation of a vector $\mathbf{v} \in \mathbb{R}^d$, a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ and a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$.

Figure 2: Tensor network representation of common operation on matrices and tensors.

$\sum_{i_1,\cdots,i_p} \boldsymbol{\mathcal{S}}_{i_1 \cdots i_p} \boldsymbol{\mathcal{T}}_{i_1 \cdots i_p}$ 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 $(\boldsymbol{\mathcal{T}} \times_2 \mathbf{X})_{i_1 i_2 i_3} = \sum_j \boldsymbol{\mathcal{T}}_{i_1 j i_3} \mathbf{X}_{i_3 j}$. *Tensor network diagrams* allow one to represent complex operations on tensors in a graphical and intuitive way. A tensor network 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). 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-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 2.

Tensor decomposition and tensor rank We now briefly present the most common tensor decomposition models. For the sake of simplicity we consider a fourth order tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3 \times d_4}$, each decomposition can be straightforwardly extended to higher-order tensors. A CP decomposition of $\boldsymbol{\mathcal{T}}$ is an expression of $\boldsymbol{\mathcal{T}}$ as a sum of rank one tensors [14], $\boldsymbol{\mathcal{T}} = \sum_{r=1}^R \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \circ \mathbf{d}_r$, and the CP rank of $\boldsymbol{\mathcal{T}}$ is the smallest R for which such a decomposition exists. A Tucker decomposition [36] decomposes $\boldsymbol{\mathcal{T}}$ as the product of a core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times R_2 \times R_3 \times R_4}$ with four factor matrices $\mathbf{U}_i \in \mathbb{R}^{d_i \times R_i}$ for $i = 1, \cdots, 4$: $\boldsymbol{\mathcal{T}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \mathbf{U}_3 \times_4 \mathbf{U}_4$. The Tucker rank, or multilinear rank, of $\boldsymbol{\mathcal{T}}$ is the smallest tuple (R_1, R_2, R_3, R_4) for which such a decomposition exists (note that the notion of Tucker rank is well defined even though there is no total order on tuples of integers [5]). The tensor ring (TR) decomposition [43, 25, 29] (also known as matrix product states in the physics community) expresses each component of $\boldsymbol{\mathcal{T}}$ as the trace of a product of slices of four core tensors $\boldsymbol{\mathcal{G}}^{(1)} \in \mathbb{R}^{R_0 \times d_1 \times R_1}$, $\boldsymbol{\mathcal{G}}^{(2)} \in \mathbb{R}^{R_1 \times d_2 \times R_2}$, $\boldsymbol{\mathcal{G}}^{(3)} \in \mathbb{R}^{R_2 \times d_3 \times R_3}$ and $\boldsymbol{\mathcal{G}}^{(4)} \in \mathbb{R}^{R_4 \times d_4 \times R_0}$: $\boldsymbol{\mathcal{T}}_{i_1,i_2,i_3,i_4} = \mathrm{Tr}(\boldsymbol{\mathcal{G}}^{(1)}_{:,i_1,:} \boldsymbol{\mathcal{G}}^{(2)}_{:,i_2,:} \boldsymbol{\mathcal{G}}^{(3)}_{:,i_3,:} \boldsymbol{\mathcal{G}}^{(4)}_{:,i_4,:})$. The tensor train (TT) decomposition [26] 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). Similarly to CP and Tucker, the TT and TR decompositions naturally give rise to an associated notion of rank: the TR rank (resp. TT rank) is the smallest tuple (R_0, R_1, R_2, R_3) (resp. (R_1, R_2, R_3)) such that a TR (resp. TT) decomposition exists.

Tensor networks offer a unifying view of all these tensor decomposition models: Figure 3 shows the tensor network representation of each decomposition model. Each decomposition is naturally associated with the graph topology of the underlying tensor network. 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 be the rank of the decomposition (assuming uniform ranks for

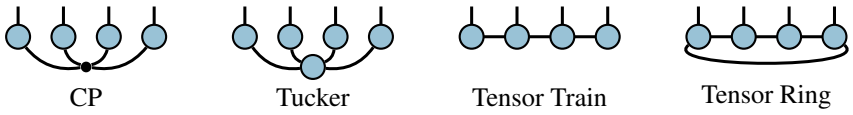


Figure 3: Tensor network representation of common decomposition models for a 4th order tensor. For CP, the black dot represents a hyperedge corresponding to a joint contraction over 4 indices.

Tucker, CP and TT), the number of parameters is in $\mathcal{O}(pdR)$ for CP, $\mathcal{O}(R^p + pdR)$ for Tucker, and $\mathcal{O}(pdR^2)$ 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 the order of the tensor.

3 Tensor Learning Framework

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:

$$\min_{\boldsymbol{\mathcal{W}} \in \mathbb{R}^{d_1 \times \cdots \times d_p}} \mathcal{L}(\boldsymbol{\mathcal{W}}) \quad \text{s.t. } \operatorname{rank}(\boldsymbol{\mathcal{W}}) \leq R \tag{1}$$

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 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 [34] and sequence modeling [11] also falls within this general formulation by using the cross-entropy or log likelihood as a loss function.

The rank constraint in Problem 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 is NP-hard because of the tensor rank constraint [13]. 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 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.

4 A Greedy Algorithm for Tensor Learning

We now show how the framework introduced in the previous section naturally generalizes beyond common decompositions to arbitrary tensor networks. This leads us to define the novel tensor optimization problem of minimizing a loss over arbitrary tensor network structures under a constraint on the number of parameters. We then present a simple greedy algorithm to tackle this problem.

For most tensor learning problems, there is no clear way of choosing which decomposition model to use. Moreover, the cost of model mis-specification can be high: e.g. for a tensor with a low

TR rank, the best low rank approximation in the TT format will almost always be worse than a TR decomposition with the same number of parameters. It may also happen that none of the commonly used models is suited for the task and using a new tensor network structure would achieve the best tradeoff between minimizing the number of parameters and minimizing the loss function. Note that achieving this tradeoff can have different implications depending on the task: for tensor decomposition this would lead to a better compression ratio, while for tensor regression this would lead to a better sample complexity of the learning algorithm. These considerations lead us to consider the problem of finding the best tensor network structure to minimize a given loss, where one wants to minimize both the number of parameters and the objective achieved by the tensor network. Of course, tensor networks with a larger number of parameters will tend to achieve lower values of the loss function. A natural way to state the problem is thus: given a bound on the maximum number of parameters, find the tensor network structure and core tensors minimizing a given loss function.

Before formalizing the tensor network structure learning problem, we introduce a few notations. For the sake of simplicity, we only consider tensor networks 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 leg corresponding to one of the dimensions (i.e. we do not consider internal core tensors such as in the Tucker decomposition, nor hyper-edges as in the CP decomposition). Note that both the TT and TR decompositions fall in this setting. A tensor network structure can then be represented as 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 tensor network; if there is no edge between nodes i and j, $R_{i,j}$ is set to one[†]. Consequently, a tensor network decomposition of $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{d_1 \times \cdots \times d_p}$ is 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. 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

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

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 $j = i + 1$.

We now formalize the tensor structure learning problem. Given a loss function $\mathcal{L}$, we want to solve

$$\min_{\substack{R_{i,j}, \\ 1 \leq i < j \leq p}} \min_{\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}} \mathcal{L}(\text{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)})) \quad \text{s.t. } \texttt{size}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leq C \tag{2}$$

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

Problem 2 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 (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 tensor network 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 (its size will grow exponentially with the order of $\boldsymbol{\mathcal{W}}$). Moreover, for most tensor learning problems the lower level continuous optimization problem is NP-hard. We now propose a simple greedy approach to optimize the upper level problem combined with automatic differentiation and continuous optimization techniques to optimize the lower level problem.

The greedy algorithm for tensor learning 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 $\boldsymbol{\mathcal{G}}^{(i)}$ is randomly initialized to a vector of dimension d_i (seen as a tensor of shape

[†]Note that in a tensor network, having an edge of dimension one is equivalent to having no edge (akin to the fact that the product of an $m \times 1$ and a $1 \times n$ matrices is equivalent to the outer product of two vectors).

Algorithm 1 `Greedy-TL`: Greedy algorithm for tensor learning.

Input: Loss function $\mathcal{L} : \mathbb{R}^{d_1 \times \cdots \times d_p} \to \mathbb{R}$, rank increment R.

// *Initialize tensor network to a random rank one tensor and optimize loss function.*
$R_{i,j} \leftarrow 1$ for $1 \leq i < j \leq p$
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
$(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leftarrow$ `optimize` $\mathcal{L}(\text{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$ w.r.t. $\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}$
repeat
  $(i, j) \leftarrow$ `find-best-edge`$(\mathcal{L}, (\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$
  $\boldsymbol{\mathcal{G}}^{(i)} \leftarrow$ `add-slice`$(\boldsymbol{\mathcal{G}}^{(i)}, j, R)$ // *add R new slices to the jth mode of $\boldsymbol{\mathcal{G}}^{(i)}$*
  $\boldsymbol{\mathcal{G}}^{(j)} \leftarrow$ `add-slice`$(\boldsymbol{\mathcal{G}}^{(j)}, i, R)$ // *add R new slices to the ith mode of $\boldsymbol{\mathcal{G}}^{(j)}$*
  $R_{i,j} \leftarrow R_{i,j} + R$
  // *Optimize tensor network with rank of edge (i, j) increased by R*
  $(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leftarrow$ `optimize` $\mathcal{L}(\text{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$ w.r.t. $\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}$
until Stopping criterion

$1 \times \cdots \times 1 \times d_i \times 1 \times \cdots \times 1$). Then, at each iteration of the greedy algorithm, the most promising edge of a tensor network is identified, the corresponding rank is increased and the loss function is optimized again w.r.t. the core tensors of the tensor network. A key idea of our approach is to restart the continuous optimization process where it left off at the previous iteration of the greedy algorithm. This is achieved by initializing 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 tensor network unchanged[‡]. As an illustration, 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 $(R_{1,2} + 1) \times d_2 \times R_{2,3} \times R_{2,4}$. Assuming that the loss function is differentiable, common automatic differentiation and gradient based optimization techniques can be used to minimize the loss function in the lower level problem. In this case, a simple heuristic to identify the most promising edge is to optimize the loss function for a few epochs for each possible edge and select the edge which led to the steepest decrease in the loss.

The overall greedy algorithm is summarized in Algorithm 1. For Problem 2, a natural stopping criterion for the greedy algorithm is when the maximum number of parameters is reached. But more evolved stopping criteria could be used. For example, in the case of tensor regression, the stopping criterion could be based on validation data (for example using an early stopping scheme). 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 could be used to adaptively select the ranks of a TT or TR decomposition. Lastly, while we propose gradient based methods for solving the lower level continuous optimization due to their broad applicability, more efficient optimization methods specifically suited for a given loss function can be used (e.g. using the alternating least square algorithm for tensor decomposition tasks).

Limitations We conclude this section by mentioning a few limitations of the proposed approach and potential avenues to address them. First, `Greedy-TL` can be time consuming since it has to test all possible edges at each iteration (quadratic in the order of the tensor). However, this step is highly parallelizable and efficient task specific heuristics can be tailored to identify the best edge to increment. Second, the greedy algorithm is one of the simplest ways of tackling the combinatorial upper level discrete optimization problem, and is prone to identifying sub-optimal solutions. More advanced discrete optimization techniques can be used, which is a promising direction for building upon `Greedy-TL`; the A^* search algorithm would be a first alternative to investigate. Lastly, the tensor network structures learned by `Greedy-TL` do not contain hyper-edges and internal nodes, but the greedy algorithm can easily be adapted to address this limitation, which is left for future work.

[‡]Indeed, one can show that if these slices were initialized exactly to 0, the resulting tensor network would represent exactly the same tensor as the original one. It is however important to initialize the slices randomly to break symmetries that could constrain the continuous optimization process.

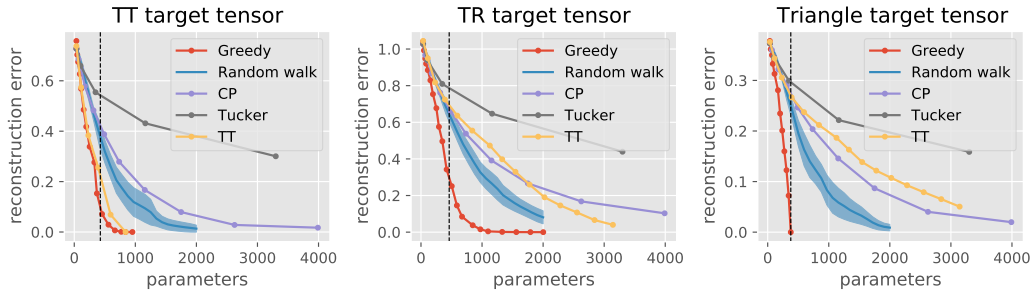


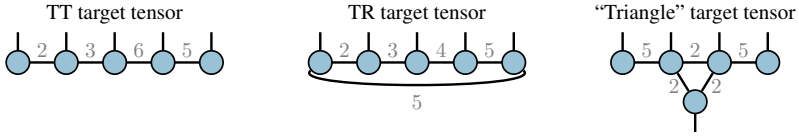
Figure 4: Reconstruction error as a function of the number of parameters for the tensor decomposition experiment on three target tensors with different low rank structure (vertical line represent the minimum number of parameters for a perfect reconstruction). The shaded area represents a 95% confidence interval for the linear interpolations of the reconstruction errors over 10 different runs of random walk.

5 Experiments

In this section we evaluate `Greedy-TL` on tensor decomposition and tensor completion tasks.

Implementation details We use PyTorch [27] and TensorNetwork [31] to implement `Greedy-TL` (Algorithm 1). We use RMSprop [35] to optimize the loss function $\mathcal{L}$ (lines 4 and 11). The learning rate is adaptively set at each step of the greedy algorithm by monitoring the loss and restarting the optimization after dividing the learning rate by 2 if the loss did not decrease during the first 10 epochs (initial learning rate is set to 0.1); the same optimization procedure is used for the search phase we describe now. To identify the best edge (line 6), we test each possible edge for a rank increment, optimize the loss for T_{search} epochs, and choose the edge leading to the largest decrease in loss; T_{search} is set to 160 for the tensor decomposition experiments and to 40 for the tensor completion experiments. In order to stabilize optimization during the search phase, the gradient updates are only applied to the new slices for the first half of the T_{search} epochs and then applied to all parameters for the remaining half. The optimization for the best rank increment (line 11) is restarted from the point it was stopped during the search phase. The components of the new slices added to the two core tensors before a rank increment (lines 7 and lines 8) are drawn independently from a normal distribution with standard deviation 10^{-6}. The rank increment parameter R of `Greedy-TL` is set to 1 for the tensor decomposition experiments and to 2 for the tensor completion ones.

Tensor decomposition We first consider a tensor decomposition task. We randomly generate three target tensors of size $7 \times 7 \times 7 \times 7 \times 7$ with the following tensor network structures:



We run the greedy algorithm until it recovers an almost exact decomposition (stopping criterion is achieved when the relative reconstruction error falls below 10^{-5}). We compare `Greedy-TL` with CP, Tucker and TT decomposition[§] of increasing rank as baselines (we use uniform ranks for Tucker and TT). We also include a simple random walk baseline based on `Greedy-TL`, where the edge for the rank increment is chosen at random at each iteration of the greedy algorithm. We run the random walk baseline for ten different random initializations. The results are shown in Figure 4 where we see that the greedy algorithm outperforms all baselines for the the three target tensors. Notably, `Greedy-TL` outperforms TT even on the TT target tensor. This is due to the fact that the rank of the TT target tensor are not uniform and `Greedy-TL` is able to adaptively set different ranks to achieve the best compression ratio. Furthermore, `Greedy-TL` is able to recover the exact tensor network structure of the triangle target tensor. For the TT and TR target tensors, `Greedy-TL` identifies the following tensor network structures at its last iteration:

[§]We use the implementations from the TensorLy python package [18] to perform these decompositions.

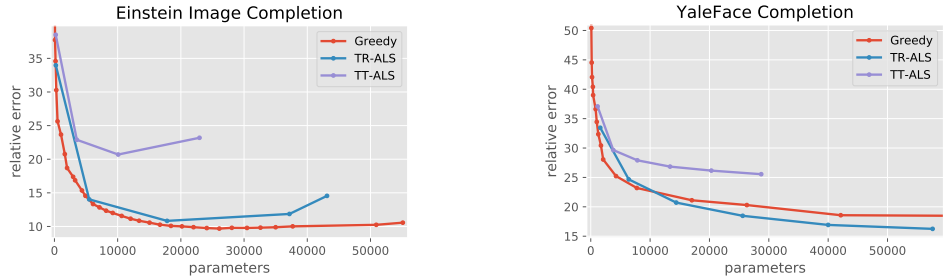


Figure 5: Relative reconstruction error of the target for the completion tasks. In both experiments, 10% of the entries are randomly observed (results for `TT-ALS` and `TR-ALS` are reported from [37]).

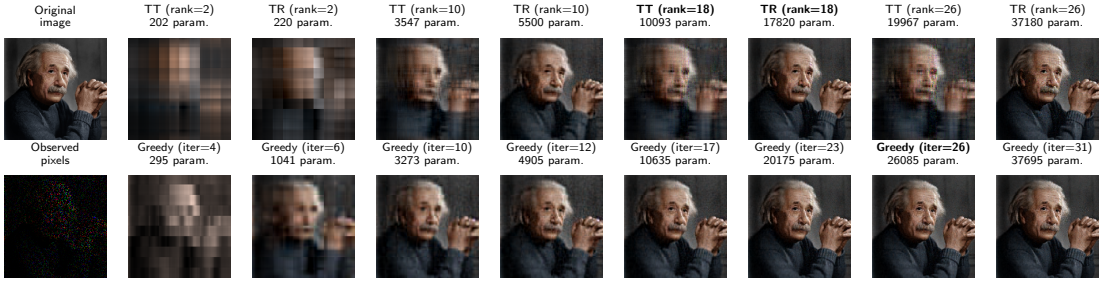


Figure 6: Recovered images from 10% of observed entries of the Einstein image for `Greedy-TL`, `TT-ALS` and `TR-ALS`. Bold titles indicate the image with the best relative error for each method.

This experiment greatly showcases the potential cost of model mis-specification: both CP and Tucker struggle to efficiently approximate the target tensors. Interestingly, even the random walk baseline outperforms CP and Tucker for all target tensors.

Tensor completion We compare `Greedy-TL` with the TT and TR alternating least square algorithms proposed in [37]. We consider two of the experiments presented in [37]: 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 and of the YaleFace database reshaped into a $6 \times 8 \times 6 \times 7 \times 8 \times 8 \times 19 \times 2$ tensor (see [37] for details). In both cases, 10% of entries are randomly observed. The relative errors as a function of number of parameters are reported in Figure 5 where we see that `Greedy-TL` outperforms both methods on the completion of the Einstein image and that all methods over-fit to the observed entries for large number of parameters (though `Greedy-TL` seems less prone to over-fitting). Recovered images for all methods are shown in Figure 6 along with the original image and observed pixels. The best recovery error (9.68%) is achieved by `Greedy-TL` at iteration 26 with 26,085 parameters. The best recovery errors for `TT-ALS` and `TR-ALS` are 20.70% and 10.83%, respectively (obtained at rank 18). At iteration 20, `Greedy-TL` already recovers an image with an error of 10.57% with 15,045 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-TL` are shown in the supplementary material. On the YaleFace data, `Greedy-TL` performs better than both methods for models with less than 10,000 parameters but falls behind `TR-ALS` for larger number of parameters. We posit that this is due to the greedy algorithm selecting sub-optimal edges, which may be explained by the unevenness of the dimensions of the YaleFace tensor: `Greedy-TL` seems to often choose to increment edges connected to the node with the largest dimension. Taking into account the increase in number of parameters when selecting the best edge is a promising direction to improve the greedy algorithm.

6 Conclusion

We introduced the novel problem of minimizing a loss function over tensor network structures and parameters. Introducing this problem opens the door to designing efficient tensor learning algorithms that are agnostic to decomposition models. We proposed a simple greedy approach to tackle this problem and we demonstrated that it can outperform common methods tailored for specific decomposition models with experiments on tensor decomposition and completion tasks. Among other promising future directions, we plan on improving the proposed approach by exploring more efficient discrete optimization techniques to solve the upper level discrete optimization problem.

Acknowledgements

This research is supported by the Canadian Institute for Advanced Research (CIFAR AI chair program) and the Natural Sciences and Engineering Research Council of Canada (Discovery program, RGPIN-2019-05949).

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] 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.

[4] 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.

[5] 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.

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

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

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

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

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

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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

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

[18] 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.

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

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

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

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

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

[24] 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.

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

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

[27] 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.

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

[29] 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.

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

[31] Chase Roberts, Ashley Milsted, Martin Ganahl, Adam Zalcman, Bruce Fontaine, Yijian Zou, Jack Hidary, Guifre Vidal, and Stefan Leichenauer. Tensornetwork: A library for physics and machine learning. *arXiv preprint arXiv:1905.01330*, 2019.

[32] 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.

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

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

[35] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

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

[37] 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.

[38] 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.

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

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

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

[42] 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.

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

[44] 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.

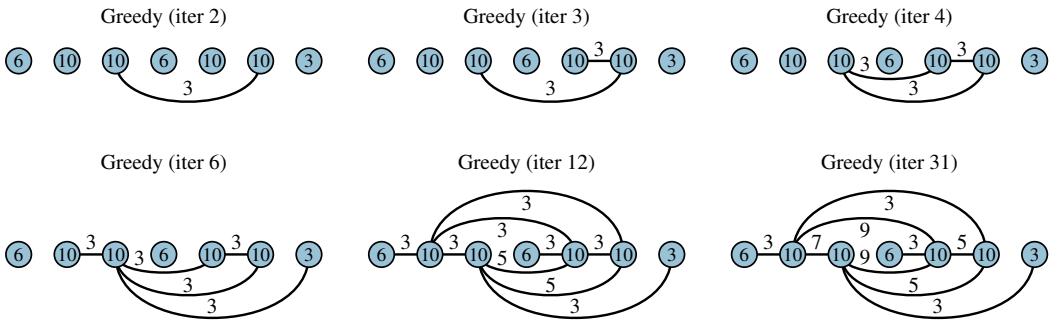


Figure 7: Learned tensor network ranks output by the Greedy-TL procedure on the Einstein image experiment. Nodes of the tensor network are labeled by the dimension of their associated mode, with the first 6 nodes describing the vertical (leftmost 3 nodes) and horizontal (next 3 nodes) position of pixels, and the last node describing the 3 color channels. The first rank increase (iteration 2) occurs between the large-scale horizontal and vertical modes of the network, with the connection of the color mode occurring at iteration 6.

A Supplementary Material

Figure 8 shows the images recovered by `Greedy-TL` at each iteration of the greedy algorithm. From these images, one can see that the tensor network node corresponding to the color channel of the image (last mode of the tensor) is connected to another node for the first time at iteration 6 (distinct colors appear in the image). Simply by looking at the images, one can also guess that the first rank increase is between two of the 10-dimensional cores of the tensor network, which can be seen by the rank one structure of each block of the 10×10 grid appearing in images recovered in the first iterations of the algorithm. These observations are verified by Figure 7, which shows the learned ranks at various iterations of the greedy search.

Figure 8: Solutions found by `Greedy-ALS` for the Einstein image completion experiments, labeled by number of parameters and relative test error w.r.t. the full image.