UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

SEARCHING OPTIMAL LOW-RANK APPROXIMATIONS USING TENSOR NETWORKS

Autor: Aran Roig

Director: Dr. Nahuel Statuto
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, May 29, 2025

Abstract

Tensor network structure search has been interesting research topic since the raise on complexity of deep learning models and quantum mechanics. This is an Undegraduate Thesis whose main goal is to give an automated search of an optimal tensor network structure for representing a given tensor with some fixed error.

For these purpose we first give an introduction to tensors, tensor networks and then we present some examples of well studied tensor networks, including Tucker decomposition, Tensor Train decomposition, Tensor Ring decomposition and Fully Connected Tensor Network decomposition.

Then with some practical experiments we demonstrate that it is possible to find more optimized structures without significant losses on performance and accuracy, and finally we will present an algorithm for finding these optimized structures.

Resum

La recerca de l'estructura òptima de xarxes de tensors ha estat un tema d'interès des de l'augment en la complexitat dels models d'aprenentatge profund i de la mecànica quàntica. Aquest és un treball de final de grau que té com a objectiu principal oferir una cerca automatitzada d'una estructura òptima de xarxa de tensors per representar un tensor donat amb un error fixat.

Per aconseguir aquest propòsit, primer oferim una introducció als tensors, a les xarxes de tensors, i tot seguit presentem alguns exemples de xarxes de tensors ben estudiades, incloent-hi la descomposició de Tucker, la descomposició en tren de tensors (Tensor Train), la descomposició en anell de tensors (Tensor Ring) i la descomposició de xarxa de tensors totalment connectada.

A continuació, amb alguns experiments pràctics, demostrem que és possible trobar estructures més optimitzades sense pèrdues significatives en el rendiment i la precissió de la representació, i finalment presentem un algoritme per trobar aquestes estructures optimitzades.

2020 Mathematics Subject Classification. 11G05, 11G10, 14G10

Agraïments

Vull agrair a ...

Contents

Chapter 1

Introduction

On the last decades, neural networks have emerged as one of the most influential topics inside the fields of artificial inteligence and machine learning. Neural networks are inspired on how the human brain works, they are made as a simplification about how our networks interact and in some way they try to emulate the way we think.

In the last years, neural networks have gained a lot of importance, positioning themselves at the center of impactful technological advances. Some models that use neural networks at its core are for example large language models (LLMs) which serve as advanced virtual agents, diffusers which generate images and other media. Other models based on neural networks have been used to make important advances in a lot of different fields such as in medicine, transportation, education, entertainment, and others.

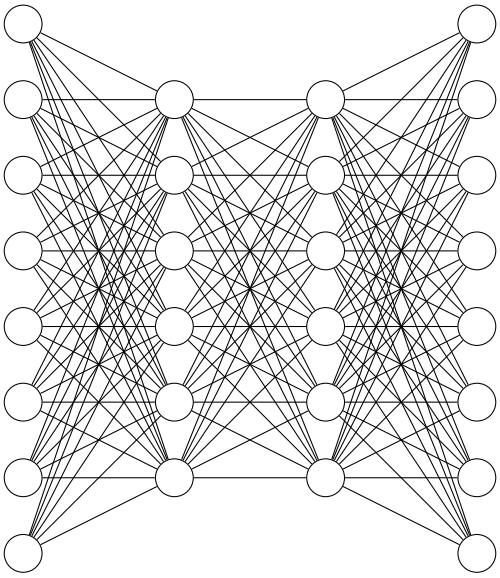


Figure 1.1: A representation of a fully connected neural network. Each dot is a neuron and each edge represents a connection

Neural networks are modeled after how real neurons work, but in a simplified manner: each neural network contains a set of neurons which are split in different layers. In a fully connected neural network, each neuron has an output that is connected to each neuron of the next layer, and all neurons of a layer have as inputs the outputs of all the neurons

of the previous layer. The first layer is called the input layer and the last layer is called the output layer.

As real neurons work, each connection will have a distinct role on when one neuron should fire or not. So, the outputs of each neuron will depend on its inputs. The output of each neuron in a neural network is modeled using usually a non-linear function $f : \mathbb{K}^N \to \mathbb{K}$, with N being the number of neurons on the previous layer.

Each neuron of the neural network also has assigned some weights w_{ji} to each input connection, with the idea that some connections will influence the firing of the network more than others. The key to making a neural network "learn" is to properly adjust these weights w_{ji} that are also called the *parameters* of the neural network.

So, more formally, the output of a neuron j in some layer will be given by the formula:

$$y_j(x) = f\left(\sum_{i=1}^{N} w_{ji}x_i + b_j\right)$$

Where x_i is the output of the neuron i of the previous layer, and b_j is some bias that is also a parameter of the neural network.

We can represent the weights of a fully connected neural network as a matrix, that we call the **weight matrix**. Since neural networks have been incresing in complexity during the recent years, these weight matrices can be very large and can contain a lot of parameters. So naturally there has been a rising interest on compressing weight matrices without sacrificing accuracy or performance.

The main goal of this thesis will be to compress the parameters of neural networks using tensor networks, a concept that originates from the study of many-body quantum systems [7] and recently has recently attracted significant interest on machine learning.

Tensor networks are a structure that is aimed to represent and efficiently manipulate large tensors by breaking them into smaller ones, called core tensors, which are connected into a specific pattern. Through the thesis we will explain how these smaller tensors are connected between them, but the general idea is that after contracting these connections, we get a representation of the original tensor.

So, what we aim to do is that given some tensor $T \in \mathbb{K}^{N_1 \times N_2 \times \cdots \times N_n}$ that will be our weight matrix reshaped onto a tensor, we will find a tensor network structure (TNS) that represents a good aproximation of T. This structure will minimize the relative error of the representation of T, the size of the TNS, since we are interested in compressing T to fewer parameters, and we will also care about the computational complexity of recovering T.

We will see that finding the best structure is an integer programming problem, that is NP-hard. Then we will deduce an algorithm that finds a locally best structure using gradient-less optimization. and finally, we will compare this structure with some other more studied cases of tensor networks, such as Tensor Train networks, Tensor Ring networks and fully connected tensor networks.

1.1 Thesis structure

First we will present some preliminaries about tensor algebra. Then, we will introduce the diagrammatic notation made by Roger Penrose in the earlies 1970. We will use it for representing tensor network contractions.

Then, we will give a formal introduction in tensor networks, based in most part from [13]. We will introduce the concept of G-ranks, and then we will introduce also well studied tensor networks such as the canonical polyadic decomposition, the tensor train decomposition, the tensor ring decomposition, and the fully connected tensor network decomposition.

We will also describe the alternating least squares algorithm applied to tensor networks for finding approximated states for any tensor network.

Then we will explain some algorithms for finding optimal tensor network structures. We will give an algorithm for finding a general tensor network structure based following [5] and we will also describe an algorithm for finding an optimal tensor network structure using gradient-less optimization algorithms.

Finally we will draw some conclusions and we will conduct some experiments of the results of combining all of the theory explained in the thesis. We will analyze the performance of the algorithms contracting random tensors, some datasets, and finally we will see the performance of the compressed tensor representation applied into fully connected neural networks.

Chapter 2

Tensors

In this chapter we will construct tensors in a formal way and we will lay down the basics of tensor algebra. We will define the notion of rank of a tensor and then, we will describe some basic tensor reshaping operations. All of this will be crucial for then presenting tensor contractions and tensor networks on the following chapter.

2.1 The tensor product space

We will denote $\mathbb{V}_1, \ldots, \mathbb{V}_n$ as finite vector spaces over a field $\mathbb{K}$ ($\mathbb{R}$ if unspecified) of dimension $\dim \mathbb{V}_i = N_i \; \forall i = 1, \ldots, n$.

We will present the notion of a tensor. From a mathematical standpoint, it can be defined as a multilinear map:

Definition 2.1. *A **multilinear map** or a tensor is an application $T : \mathbb{V}_1 \times \cdots \times \mathbb{V}_n \to \mathbb{K}$ which satisfies:*

1. $T(v_1, \ldots, \lambda v_i, \ldots, v_n) = \lambda \cdot T(v_1, \ldots, v_i, \ldots, v_n)$
2. $T(v_1, \ldots, v_i + u, \ldots, v_n) = T(v_1, \ldots, v_i, \ldots, v_n) + T(v_1, \ldots, u, \ldots, v_n)$
   $\forall i = 1, \ldots, n, \; u \in \mathbb{V}_i, \lambda \in \mathbb{K}$

Linear maps and bilinear maps are specific cases of multilinear maps with $n = 1$ and $n = 2$ respectively.

All tensors will be multilinear maps as tensors. $T : \mathbb{V}_1 \times \cdots \times \mathbb{V}_p \times \mathbb{W}_1^* \times \cdots \times \mathbb{W}_q^*$ with $n = p + q$. We will write $T : \mathbb{V}_1 \times \cdots \times \mathbb{V}_p \times \mathbb{W}_1^* \times \cdots \times \mathbb{W}_q^*$ when we need to explicitly distinguish what vector spaces are duals from another vector space.

Definition 2.2. *Let $T : \mathbb{V}_1 \times \cdots \times \mathbb{V}_p \times \mathbb{W}_1^* \times \cdots \times \mathbb{W}_q^* \to \mathbb{K}$ a multilinear map. We say that the tensor T is p-times covariant and q-times contravariant.*

Now we will present formally the tensor space and its elements by defining a relation between all the vectors in the free vector space over the cartesian product $\mathbb{V}_1 \times \cdots \times \mathbb{V}_n$. First of all, we will need to define what the free vector space of a set is:

Definition 2.3. *Let S be a set and $\mathbb{K}$ a field. We denote the free vector space over S as $\mathbb{K}[S]$, which is the vector space containing all linear combinations of elements from S*

with coefficients of $\mathbb{K}$*, i.e:*

$$\mathbb{K}[S] = \left\{ \sum_{i=1}^{n} a_i s_i \mid s_i \in S, n \in \mathbb{N}, a_i \in \mathbb{K} \right\}$$

Example 2.4. If we take $\mathbb{K} = \mathbb{R}$ and $S = (x, y)$, then the elements of $\mathbb{R}[S]$ have the form $ax + by$ with $a, b \in \mathbb{R}$. We can see that $\mathbb{R}[S]$ is the vector space $\mathbb{R}^2$ with some fixed basis (x, y).

Let $\mathbb{L} = \mathbb{K}[\mathbb{V}_1 \times \cdots \times \mathbb{V}_n]$. We define relation R as the smallest equivalence relation that satisfies:

$$(v_1, \ldots, \alpha v_i, \ldots, v_n) \sim \alpha(v_1, \ldots, v_n) \; \forall i = 1, \ldots, n, \forall \alpha \in \mathbb{K}$$

$$(v_1, \ldots, v_i + u_i, \ldots, v_n) \sim (v_1, \ldots, v_i \ldots, v_n) + (v_1, \ldots, u_i, \ldots, v_n) \; \forall i = 1, \ldots, n$$

Proposition 2.5. $[0]_R$ *is a subspace of* $\mathbb{K}[\mathbb{V}_1 \times \cdots \mathbb{V}_n]$

Proof. $0 \in [0]_R$. It is sufficient to see that for all $\lambda \in \mathbb{K}$ and $u, v \in [0]_R$ then $u + \lambda v \in [0]_R$. We can write $u + \lambda v \sim_R (u_1 + v_1, \ldots, u_i + v_i, \ldots, u_n + v_n) \sim_R 0 + (v_1, \ldots, \lambda v_i, \ldots, v_n) \sim_R 0 + \lambda 0 \sim_R 0$ and therefore $u + \lambda v \sim_R 0$ since R is an equivalence relation. □

Definition 2.6. *The* ***tensor product space*** $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ *is defined as the quotient* $\mathbb{L}/R_0$*. We denote the equivalence class of* $(v_1, \ldots, v_n)$ *as* $v_1 \otimes \cdots \otimes v_n$*.*

We will see with the universal property of the tensor product that since the elements of $\mathbb{L}/R$ satisfy the properties that define a multilineal map, each vector of $\mathbb{L}/R$ is a **tensor**.

Before, we present now an example of a tensor product space:

Example 2.7. Given $\mathbb{R}^2$ and $\mathbb{R}^3$ with its canoncial vector space structures, $\mathbb{R}^2 \otimes \mathbb{R}^3$ is defined by the equivalence classes that follow the relations

$$(\alpha u, v) \sim \alpha(u, v) \sim (u, \alpha v) \quad \forall \alpha \in \mathbb{K}, \; u \in \mathbb{R}^2, v \in \mathbb{R}^3$$

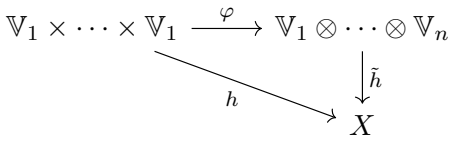
$$(u + u', v) \sim (u, v) + (u' + v) \quad \forall u, u' \in \mathbb{R}^2, v \in \mathbb{R}^3$$

$$(u, v + v') \sim (u, v) + (u + v') \quad \forall u \in \mathbb{R}^2, v, v' \in \mathbb{R}^3$$

And an element of $\mathbb{R}^2 \otimes \mathbb{R}^3$ would be the representant of $[((1, 2, 3), (0, 4))]$ in which for example other representants of the same equivalence class would be $2 \cdot ((1, 2, 3), (0, 2))$ or $((1, 0, 3), (0, 2)) + ((0, 2, 0), (0, 2))$

We will now state and prove the universal property of the tensor product:

Theorem 2.8 (Universal property of the tensor product)**.** *Let* φ *be the quotient mapping from* $\mathbb{V}_1 \times \cdots \times \mathbb{V}_n$ *to* $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$*. For every multilinear map* $h : \mathbb{V}_1 \times \cdots \times \mathbb{V}_n \to X$ *where* X *is any vector space there exists an unique linear map* $\tilde{h} : \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n \to X$ *such that the following diagram commutes:*



Proof. Let $\varphi(v_1,\dots,v_n) := [(v_1,\dots,v_n)] \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$. Let $h : \mathbb{V}_1 \times \cdots \mathbb{V}_n \to X$ be a multilinear map. We define $\tilde{H} : \mathbb{K}[\mathbb{V}_1 \times \cdots \times \mathbb{V}_n] \to X$ by:

$$\tilde{H}\left(\sum_{i=1}^{p} a_i(v_i^1,\dots,v_i^n)\right) := \sum_{i=1}^{p} a_i h(v_i^1,\dots,v_i^n)$$

Consider now the vector subspace $[0]_R$ of $\mathbb{K}[\mathbb{V}_1 \times \cdots \times \mathbb{V}_n]$. Since h is multilinear, we can see that $\tilde{H}$ sends every element of $[0]_R$ to $0 \in X$, therefore $W \subseteq \ker \tilde{H}$ and hence $\tilde{H}$ induces a well-defined linear map $\tilde{h} : \mathbb{K}[\mathbb{V}_1 \times \cdots \times \mathbb{V}_n]/R = \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n \to X$ that satisfies $\tilde{h}(v_1 \otimes \cdots \otimes v_n) = h(v_1,\dots,v_n)$

Suppose that exists another mapping $f : \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n \to X$ such that $f(v_1 \otimes \cdots \otimes v_n) = h(v_1,\dots,v_n)$, then we would have

$$f\left(\sum_{i=1}^{p} a_i v_i^1 \otimes \cdots \otimes v_i^n\right) = \sum_{i=1}^{p} a_i h(v_i^1,\dots,v_i^n) = \tilde{h}\left(\sum_{i=1}^{n} a_i v_i^1 \otimes \cdots \otimes v_i^n\right)$$

therefore, the linear mapping $\tilde{h}$ is unique □

We can now construct explicitly the corresponding vector space (and a basis) of $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$.

Proposition 2.9. *Let $\{e_1^i, e_2^i, \dots, e_{N_i}^i\}$ be basis for each $\mathbb{V}_i$ and $N_i = \dim \mathbb{V}_i$. Then the set*

$$\mathcal{B}_\otimes = \{e_{i_1}^1 \otimes \cdots \otimes e_{i_n}^n = [(e_{i_1}^1,\dots,e_{i_n}^n)]_R : 1 \leqslant i_j \leqslant N_j, 1 \leqslant j \leqslant n\}$$

is a basis of $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$.

Proof. If we have $v_1 \otimes \cdots \otimes v_n \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$, if we write $v_i = \sum_{j=1}^{N_i} \lambda_j^i e_j^i$ then, because of the multilinearity of R we get that

$$v_1 \otimes \cdots \otimes v_n = \left(\sum_{i=1}^{N_1} \lambda_1^i e_1^i\right) \otimes \cdots \otimes \left(\sum_{i=1}^{N_n} \lambda_n^i e_n^i\right) = \sum_{s_1,\dots,s_n}^{N_1,\dots,N_n} \lambda_1^{s_1} \cdots \lambda_n^{s_n} (e_1^{s_1} \otimes \cdots \otimes e_n^{s_n})$$

And since all elements of $\mathbb{V}_1 \otimes \mathbb{V}_n$ can be written in this form, $\mathcal{B}_\otimes$ spans the entire space. For proving the independance of each element of $\mathcal{B}_\otimes$, suppose that we have a linear combination such that

$$\sum_{s_1,\dots,s_n}^{N_1,\dots,N_n} \lambda_{s_1,\dots,s_n} e_{s_1}^1 \otimes \cdots \otimes e_{s_n}^n = 0 \tag{2.1.1}$$

Let $\{{}^*e_1^i, \dots, {}^*e_n^i\}$ be the dual basis for $\mathbb{V}^i$ such that ${}^*e_j^i(e_k^i) = \delta_{jk}$. We define now the multilinear map

$$\begin{aligned} f_{(k_1,\dots,k_n)} : \mathbb{V}_1 \times \cdots \times \mathbb{V}_n &\longrightarrow \mathbb{K} \\ (v_1,\dots,v_n) &\longmapsto {}^*e_{k_1}^1(v_1) \cdot {}^*e_{k_2}^2(v_2) \cdots {}^*e_{k_n}^n(v_n) \end{aligned}$$

With $1 \leqslant k_i \leqslant N_i$. The image of $f_{(k_1,\dots,k_n)}$ extracts the coefficient $\lambda_{k_1,\dots,k_n}$ of any tensor $v_1 \otimes \cdots \otimes v_n$.

Applying the universal property of the tensor product, there exists an unique linear map $\tilde{f}_{(s_1,\dots,s_n)} : \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n \to \mathbb{K}$ such that

$$\tilde{f}_{(k_1,\dots,k_n)}(e^1_{j_1} \otimes \cdots \otimes e^n_{j_n}) = f_{(k_1,\dots,k_n)}(e^1_{j_1},\dots,e^n_{j_n}) = \delta_{j_1k_1} \cdot \delta_{j_2k_2} \cdots \delta_{j_nk_n}$$

If we now apply the linerar combination to $\tilde{f}_{(s_1,\dots,s_n)}$ we get

$$\tilde{f}_{(k_1,\dots,k_n)}\left(\sum_{s_1,\dots,s_n}^{N_1,\dots,N_n} \lambda_{s_1,\dots,s_n} e^1_{s_1} \otimes \cdots \otimes e^n_{s_n}\right) = \sum_{s_1,\dots,s_n}^{N_1,\dots,N_n} \lambda_{s_1,\dots,s_n} \cdot \delta_{k_1s_1} \cdot \delta_{k_2s_2} \cdots \delta_{k_ns_n} = \lambda_{k_1,\dots,k_n}$$

But from (2.1.1) $\lambda_{k_1,\dots,k_n} = 0$ for all $1 \leqslant k_i \leqslant N_i$. Therefore, the elements of $\mathcal{B}_\otimes$ are independent and form a basis. □

So now, we can directly work with the tensor space since it is a vector space and we have a well defined basis. The dimension of $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ is $N_1 \cdot N_2 \cdots N_n$ and its elements can be expressed as

$$T = \sum_{s_1,\dots,s_n}^{N_1,\dots,N_n} T_{s_1,\dots,s_n} \cdot e^1_{s_1} \otimes \cdots \otimes e^n_{s_n} \tag{2.1.2}$$

If we wanted to store a tensor in a computer program, it would be enough to save all the $T_{s_1,\dots,s_n}$ entries.

Definition 2.10. *We will define the* ***size*** *of the tensor $T \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ as* $\text{Size}(T) = N_1N_2 \cdots N_n$. *We will also say that the* ***order*** *of T is n.*

Now, we want to be able to do the product between two tensors of different tensor product spaces since we will need this for contracting tensor networks. In other words, suppose that $T \in \mathbb{V}_1 \otimes \cdots \mathbb{V}_n$ and $U \in \mathbb{W}_1 \otimes \cdots \otimes \mathbb{W}_m$. We want that a tensor product operation results in a tensor $T \otimes U \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n \otimes \mathbb{W}_1 \otimes \cdots \otimes \mathbb{W}_m$. We will define this operation in the following way:

Definition 2.11 (Tensor product)**.** *With the above notation, let* $\dim \mathbb{V}_i = N_i$, $\dim \mathbb{W}_j = M_j$ *and some basis* $\{e^i_1,\dots,e^i_{N_i}\}$ *of each* $\mathbb{V}_i$ *and* $\{p^j_1,\dots,p^j_{M_j}\}$ *of each* $\mathbb{W}_j$, *we define the tensor product $T \otimes U$ as*

$$T \otimes U = \sum_{i_1,\dots,i_n}^{N_1,\dots,N_n} \sum_{j_1,\dots,j_m}^{M_1,\dots,M_m} T_{i_1,\dots,i_n} U_{j_1,\dots,j_m} \cdot e^1_{i_1} \otimes \cdots \otimes e^n_{i_n} \otimes p^1_{j_1} \otimes \cdots \otimes p^m_{j_m} \tag{2.1.3}$$

Which naturally is an element of $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n \otimes \mathbb{W}_1 \otimes \cdots \otimes \mathbb{W}_m$. We call the tensor product for two tensors of order 2 as the **outer product**, so we can say that the tensor product is a generalization of it.

Example 2.12. Let $\{a_1, a_2\} \subset \mathbb{V}_1$, $\{b_1, b_2\} \subset \mathbb{V}_2$ and $\{c_1, c_2\} \subset \mathbb{W}_1$, $\{d_1, d_2\} \subset \mathbb{W}_2$ be basis of their corresponding vector spaces. Let $T \in \mathbb{V}_1 \otimes \mathbb{V}_2$ and $U \in \mathbb{W}_1 \otimes \mathbb{W}_2$ defined as:

$$T = 2(a_1 \otimes b_1) + 3(a_2 \otimes b_1) \qquad U = c_1 \otimes d_1 + c_2 \otimes d_2$$

Then, the tensor product $T \otimes U$ would be:

$$T \otimes U = 2(a_1 \otimes b_1 \otimes c_1 \otimes d_1) + 2(a_1 \otimes b_1 \otimes c_2 \otimes d_2) +$$

$$3(a_2 \otimes b_1 \otimes c_1 \otimes d_1) + 3(a_2 \otimes b_1 \otimes c_2 \otimes d_2)$$

Now, we already know how to add and subtract two tensors of the same tensor product space since its sum is already defined by its vector space. We will introduce a tensor norm since in the following chapters we will want to know if a tensor is "small" or "big", and also but not less important, we want to know if two tensors are near each other to test convergance for algorithms involving tensor networks. In our case we will stick to the Frobenius norm:

Definition 2.13. *We define the frobenius norm as:*

$$\|\cdot\|_F : \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n \longrightarrow \mathbb{R}_+$$
$$\left(\sum_{s_1,\ldots,s_n}^{N_1,\ldots,N_n} T_{s_1,\ldots,s_n} \cdot e^1_{s_1} \otimes \cdots \otimes e^n_{s_n}\right) \longmapsto \sqrt{\sum_{s_1,\ldots,s_n}^{N_1,\ldots,N_n} T^2_{s_1,\ldots,s_n}}$$

2.2 Tensor ranks

In this section we will present the rank of a tensor. It will serve as the extension of the matrix rank, which is defined as the dimension of the vector space spanned by the vectors on its columns. For the matrix rank, we say that if it is spanned by a single vector, the rank is 1, so we could somewhat say that a vector has rank 1. We will do something similar, defining that a tensor t is a rank-1 tensor if it can be written as

$$t = v^1 \otimes \cdots \otimes v^n$$

with $v^i \in \mathbb{V}_i$. Therefore, the rank of a tensor will be r if it can be spanned from r rank-1 tensors:

Definition 2.14. *We say that a tensor T has rank r as* $\operatorname{rank} T = r$ *with $r \in \mathbb{N}$ if r is the minimum value such that we can write T as the following form*

$$T = \sum_{p=1}^{r} \lambda_p v^1_p \otimes \cdots \otimes v^n_p \tag{2.2.1}$$

where $v^i_1, \ldots, v^i_r \in \mathbb{V}_i, i = 1, \ldots, n$ and $\lambda_p \in \mathbb{K}$

The rank of a tensor is bounded by $\prod_{i=1}^n N_i$ since we can decompose every tensor as the sum of the elements of a basis of the tensor product space, and so the rank of the tensor does not exceed the number of elements of the basis.

Unlike matrices, determining the rank of a tensor is an NP-hard problem (See Section 8 of [3]). Finding the maximum rank, i.e determining $\max_{T \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n} \operatorname{rank} T$ still remains an unresolved problem. [1]

By manipulating the vectors $v^1_p \otimes \cdots \otimes v^n_p$ to represent each value of the tensor $T_{s_1,\ldots,s_n}$, we can find an slightly better upper bound for the tensor rank:

Proposition 2.15. *Let* $\dim \mathbb{V}_i = N_i$*, then*

$$\operatorname{rank} T \leqslant \left\lfloor \frac{\prod_{i=1}^n N_i}{\sum_{i=1}^n N_i} \right\rfloor \tag{2.2.2}$$

Proof. Let $r = \operatorname{rank} T$. We can write $T = \sum_{p=1}^{r} v_p^1 \otimes \cdots \otimes v_p^n$. Now, each term of this sum has $\sum_{i=1}^{n} N_i$ adjustable parameters, since each $v_p^{(i)}$ is a vector of $\mathbb{V}_i$ with its dimension being N_i. So, in total we will have $r \sum_{i=1}^{n} N_i$ adjustable parameters in our decomposition. Since our tensor T is completly determined by $\prod_{i=1}^{n} N_i$ parameters, we can impose $r \sum_{i=1}^{n} N_i \leqslant \prod_{i=1}^{n} N_i$ □

Decomposing a tensor T in rank-1 tensors as in eq. (2.2.1) is known as **tensor rank decomposition**. One could ask if given a tensor T and fixed r, can we construct a tensor T' of some fixed rank r' such that $\|T - T'\|_F$ is minimum. The decomposition T' is called **canonical polyadic decomposition** and it is an special case of a tensor network that we will see on the following chapter.

We will now present some essential reshaping operations that will help us manipulating tensors. Our main goal by presenting the reshaping operations is the unfolding and folding operations, that somehow "transforms" a tensor onto a matrix and the other way around respectively.

2.3 Reshaping operations

Any tensor $T \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ can be identified as an n-dimensional array. In other words, for each tensor T we can define a discrete function $\mathcal{T}$ that encodes the representation in a basis of the tensors T as:

$$\begin{aligned} \mathcal{T} : \prod_{i=1}^{n} \{1, \ldots, N_i\} &\longrightarrow \mathbb{K} \\ (i_1, \ldots, i_n) &\longmapsto T_{i_1, \ldots, i_n} \end{aligned}$$

From now on we will identify the set of all images of $\mathcal{T}$ as an element of $\mathbb{K}^{N_1 \times \cdots \times N_n}$. Therefore, a lot of times we will write a tensor $T \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ as an element of $\mathbb{K}^{N_1 \times \cdots \times N_n}$ with $\dim \mathbb{V}_i = N_i$, since this correspondance is already well defined. We will also write $T(i_1, \ldots, i_n)$ as the image of $\mathcal{T}$ of $(i_1, \ldots, i_n)$. Since now we can see a tensor as an n-dimensional array thanks to the mapping $\mathcal{T}$, we can start reshaping tensors. But before that, we will define what is a mode of a tensor:

Definition 2.16. *We define the j-th of a tensor as its j-th dimension. A tensor of order n has n different modes.*

For example, having a 3-order tensor $T \in \mathbb{R}^{N_1 \times N_2 \times N_3}$. We can write each entry of the tensor as $T(i_1, i_2, i_3)$. The first mode of T is i_1, the second one, i_2 and the third i_3. In other words, when we say the j-mode of a tensor we are referring at one argument of the discrete function $\mathcal{T}$

Now we will introduce the linearization operation which simplify the notation a lot when we define reshaping operations.

Definition 2.17 (Linearization)**.** *Fixed $N_1, \ldots, N_n \in \mathbb{N}$, given $i_1, \ldots, i_n \in \mathbb{N}$ such that $1 \leqslant i_1, \leqslant N_1, \ldots, 1 \leqslant i_n \leqslant N_n$, we define the **linearization** of the indices $i_1, \ldots, i_n$ as the mapping $\prod_{i=1}^{n} \{1, \ldots, N_i\} \to \{1, \ldots, \prod_{i=1}^{n} N_i\}$ and with its images defined as:*

$$\overline{i_1, i_2, \ldots, i_n} = \sum_{j=2}^{n} \left((i_j - 1) \prod_{k=1}^{j} N_k \right) + i_1$$

The purpose of the linearization mapping is to give a bijection within each element of the form $(i_1, \dots, i_n) \in \prod_{i=1}^{n} \{1, \dots, N_i\}$ to a positive natural number. For example, consider $n = 3$ and $N_1 = N_2 = N_3 = 3$. The encoding of the tuple $(1, 1, 1)$ corresponds to 1, the tuple $(2, 1, 1)$ to 2, the tuple $(1, 2, 3)$ to $1 + (2-1) \cdot 3 + (3-1) \cdot 3 = 10$. One should keep in mind that when defining an array in computer science, usually the first element starts at the index 0, and our linearization operation starts with the indices at 1. Throughout the thesis we will stick to array indices starting at 1 for consistency, but by doing a change of variables $i'_j = i_j - 1$ before applying the linearization and then subtracting 1 also on the image, we will get the same operation but for arrays starting at 0.

We will now define the vectorization operation, which reshapes a tensor $T \in \mathbb{K}^{N_1 \times N_2 \times \cdots \times N_n}$ to a vector of $\mathbb{K}^{N_1 \cdot N_2 \cdots N_n}$

Definition 2.18. *We define the* ***vectorization*** *of T as the first order tensor (or vector) $\mathcal{V} \in \mathbb{K}^{N_1 N_2 \cdots N_n}$ defined entrywise as*

$$\mathcal{V}(\overline{i_1 i_2 \dots i_n}) = T(i_1, i_2, \dots, i_n)$$

We will write the vectorization of T as $\operatorname{vec} T$

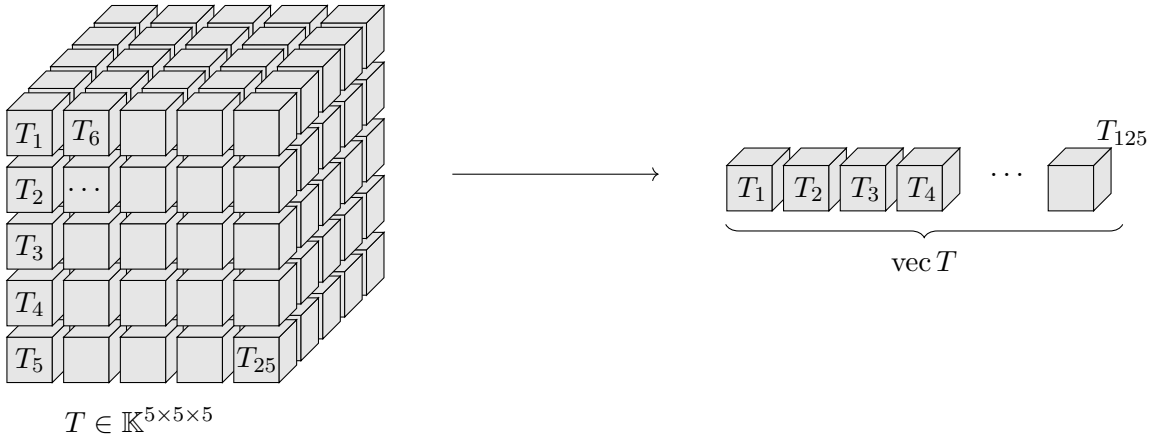


Figure 2.1: Tensor vectorization

We will also define the permutation of two modes of a tensor. That means swapping arguments of the function $\mathcal{T}$ following some permutation $\sigma \in S_n$.

Definition 2.19 (Tensor permutation)**.** *Given a permutation $\sigma \in S_n$, and a tensor $T \in \mathbb{K}^{N_1 \times N_2 \times \cdots \times N_n}$ we define $T_\sigma \in \mathbb{K}^{N_{\sigma(1)} \times N_{\sigma(2)} \times \cdots \times N_{\sigma(n)}}$ entrywise as*

$$T_\sigma(i_1, \dots, i_n) = T(i_{\sigma^{-1}(1)}, i_{\sigma^{-1}(2)}, \dots, i_{\sigma^{-1}(n)})$$

Example 2.20. Let $M \in \mathbb{K}^{N_1 \times N_2}$. The matrix transposition M^T can be written as $M_{(2,1)}$

Example 2.21. Suppose that we have $X \in \mathbb{R}^{3 \times 2 \times 2}$ defined as

$$X = \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \right]$$

(3, 1, 2) Then the tensor $X_{(3,1,2)} \in \mathbb{R}^{2\times2\times3}$ would be written entrywise as

$$X_{(3,1,2)}(i_1, i_2, i_3) = X(i_3, i_1, i_2)$$

And therefore,

$$X_{(3,1,2)} = \left[\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & -1 \end{pmatrix}\right]$$

Reshaping tensors onto matrices will also be very useful since it will let us treat high order tensors as matrices and then apply numerical algorithms there.

Definition 2.22 (Tensor unfolding)**.** *Let T be a tensor of order n with $n \geqslant 2$. Let $\sigma \in S_n$ be a permutation of $(1, 2, \ldots, n)$. We define the **unfolding** of the tensor T as the 2nd-order tensor or matrix $\mathcal{U} \in \mathbb{R}^{\prod_{i=1}^{d} N_{\sigma(i)} \times \prod_{i=d+1}^{n} N_{\sigma(i)}}$ entrywise as*

$$\mathcal{U}(\overline{i_{\sigma(1)}, \ldots, i_{\sigma(d)}}, \overline{i_{\sigma(d+1)}, \ldots, i_{\sigma(n)}}) = T(i_1, \ldots, i_n)$$

We will write $\mathcal{U} = \text{unfold}\,(\mathcal{T}, (\sigma(1), \ldots, \sigma(d)), (\sigma(d+1), \ldots, \sigma(n)))$

Example 2.23. Suppose that we have X defined as in Example 2.21. Then unfold $(X, (1, 2), (3))$ would result in a matrix $\mathbb{R}^{3\times4}$ with its elements defined as $\mathcal{U}(\overline{i_1 i_2}, \overline{i_3}) = X(i_1, i_2, i_3)$. Computing each entry gives

$$\text{unfold}\,(X, (1, 2), (3)) = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 \end{pmatrix}$$

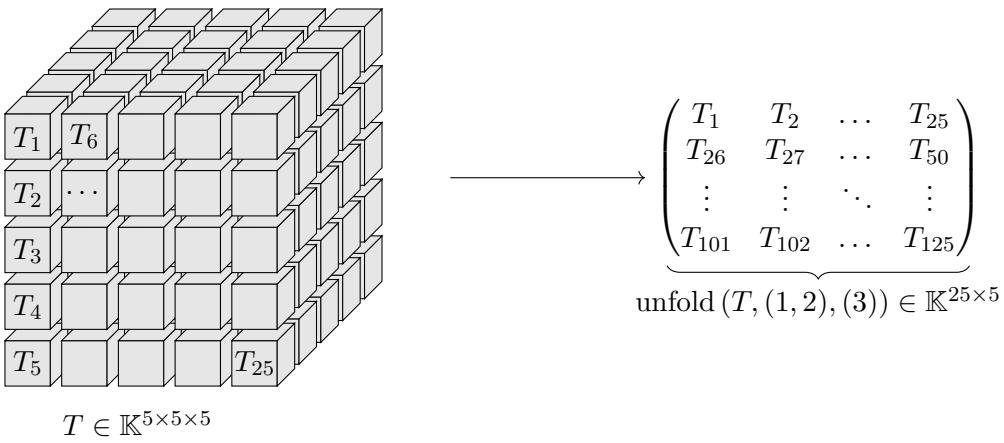


Figure 2.2: Tensor unfolding

Now we will introduce the reverse operation of the tensor unfolding: tensor folding.

Definition 2.24 (Tensor folding)**.** *Let $\sigma \in S_n$, $1 \leqslant d \leqslant n$. Let $N = \prod_{i=1}^{d}$, $\mathcal{U} = \prod i = d + 1^n$ and M be a matrix of $\mathbb{K}^{N\times M}$. We define the **folding** of M following σ as the tensor $T \in \mathbb{K}^{N_{\sigma(1)}\times\cdots\times N_{\sigma(n)}}$ defined entrywise as:*

$$T(i_1, \ldots, i_n) = \mathcal{U}(\overline{i_{\sigma^{-1}(1)}, \ldots, i_{\sigma^{-1}(d)}}, \overline{i_{\sigma^{-1}(d+1)}, \ldots, i_{\sigma^{-1}(n)}})$$

We will write $T = (\mathcal{U}, (\sigma(1), \ldots, \sigma(d)), (\sigma(d+1), \ldots, \sigma(n)))$

Chapter 3

Tensor networks

In this chapter we will start by presenting tensor networks from a mathematically formal standpoint. Then, we will present the notion of the tensor G-rank that will be useful for determining if a tensor network can represent a tensor. We will present some common tensor network structures. After that, we will discuss over the ordering in which its the most optimal to contract a tensor network and we will give an algorithm that finds an optimal order of contraction, and finally, we will present the alternating least squares algorithm applied to generic tensor networks for explicitly finding the tensor cores of a tensor network.

The concept of tensor networks originated from a physics background. Roger Penrose described how its diagrammatic language could be used in various applications of physics [9]. Later, in 1992, Steven R. White developed de Density Matrix Renormalization Group (DRMG) algorithm for quantum lattice systems. It was considered the first successfull tensor network application [12].

3.1 Tensor contraction and the Penrose Notation

In this section we will define the tensor contraction operation, which is one of the most important operations in tensor algebra and is the core concept behind tensor networks. The idea behind tensor contraction is that given two tensors from different spaces, say for example $T \in \mathbb{R}^{N_1 \times \cdots \times N_n}$ and $U \in \mathbb{R}^{M_1 \times \cdots \times M_m}$, we pick one dimension of each tensor N_i and M_j with the same size $N_i = M_j$ and then we obtain a new tensor of order $n + m - 2$ that contains all dimensions except for N_i and M_j, and each element of the resulting tensor is obtained by generalizing the matrix product opertion but along N_i and M_j dimensions.

So, the elements of the tensor contraction $T \times_j^i U$ would be, element-wise:

$$\sum_{k=1}^{N_i} T(s_1, \ldots, s_{i-1}, k, s_{i+1}, \ldots s_n) \cdot U(t_1, \ldots, t_{j-1}, k, t_{j+1}, \ldots, t_m)$$

Where $T \times_j^i U \in \mathbb{K}^{N_1 \times \cdots \times N_{i-1} \times N_{i+1} \times \cdots \times N_n \times M_1 \times \cdots \times M_{j-1} \times M_{j+1} \times \cdots \times M_m}$

Since the notation of tensor contractions is often very tedius, we will introduce the Penrose Notation for representing tensor contractions in a more compact and elegant way. The

Penrose notation dates back from at least the early 1970s and was firstly used by Robert Penrose, to which the name is owed. [9]

Given an nth-order tensor $T \in \mathbb{K}^{N_1 \times \cdots \times N_n}$ we represent it using the Penrose notation is as a circle with as many edges as the order of the tensor, as seen in fig. 3.1

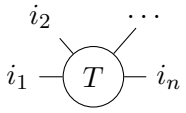


Figure 3.1: Representation of a tensor $T \in \mathbb{K}^{N_1 \times \cdots \times N_n}$ using the Penrose notation

The dimensions of the tensor are not explicitly written in the Penrose notation. Instead, only the order of the indices is preserved. If they are not explicitly set on the labels of the edges, the order of the indexes of the tensor will be determined by their orientation respect to the circle: the order starts from the left and then follows a clockwise rotation. The order in which we encounter the edges will be the order of the indexes. For example, in fig. 3.1, the order would be $i_1, i_2, \ldots, i_n$

Then, we represent a contraction of two tensors on the Penrose notation by joining to edges of different tensors. The edges that are joined will be the edges that the contraction is performed. As seen in fig. 3.2.

Now, we will give a more formal definition of the tensor contraction since there is a natural way in which tensor contraction definition appears from applying a vector space with its dual with different tensors. As before, we take two tensors T and U with $T \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ and $U \in \mathbb{W}_1 \otimes \cdots \otimes \mathbb{W}_m$. Suppose that there exists some vector space of the dimensions of U that is the dual of some space of the tensor T, in other words, suppose that exists some i and j in which $\mathbb{W}_j = \mathbb{V}_i^*$. Then, the tensor contraction is doing the tensor product of T and U and then applying $\mathbb{V}_i$ with its dual afterwards. Doing this gives the following definition:

Definition 3.1. *Let $P \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ and suppose that there exists some i and j such that $\mathbb{V}_j = \mathbb{V}_i^*$ The following map*

$$\begin{aligned} \mathcal{C}_j^i : \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n &\longrightarrow \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_{i-1} \otimes \mathbb{V}_{i+1} \otimes \cdots \otimes \mathbb{V}_{j-1} \otimes \mathbb{V}_{j+1} \otimes \cdots \otimes \mathbb{V}_n \\ v_1 \otimes \cdots \otimes v_n &\longmapsto (v_1 \otimes \cdots \otimes v_{i-1} \otimes v_{i+1} \otimes \cdots \otimes v_{j-1} \otimes v_{j+1} \otimes \cdots \otimes v_n)\, v_j(v_i) \end{aligned}$$

where $v_j \in \mathbb{V}_i^*$. We define $\mathcal{C}_j^i$ as the tensor contraction mapping of $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ over the indices i and j. We call $\mathcal{C}_i^j(T)$ the contraction of T by indices (i, j).

Now, we define the **contraction of two tensors** T and U defined as before with some space $\mathbb{W}_j$ being the dual of some $\mathbb{V}_i$ as $\mathcal{C}_j^i(T \otimes U)$. We will sometimes write $T \times_j^i U$

We will represent the contraction between two tensors as their representation in the Penrose notation with the edges that represent the indexes that are contracting by joining them, as seen in fig. 3.2.

If we fix basis for $\mathbb{V}_1, \ldots, \mathbb{V}_p, \mathbb{W}_1, \ldots, \mathbb{W}_q$ and we represent X, Y as discrete functions by its representations in those basis, we get a way for computing $\mathcal{C}_k^l(X \otimes Y)$ as:

$$\begin{aligned} &\mathcal{C}_k^l(i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_p, j_1, \ldots, j_{l-1}, j_{l+1}, \ldots, j_q) \\ &= \sum_{s=1}^{N_k} \mathcal{X}(i_1, \cdots, i_{k-1}, s, i_{k+1}, \cdots, i_p)\mathcal{Y}(j_1, \cdots, j_{l-1}, s, j_{l+1}, \cdots, j_q) \end{aligned} \tag{3.1.1}$$

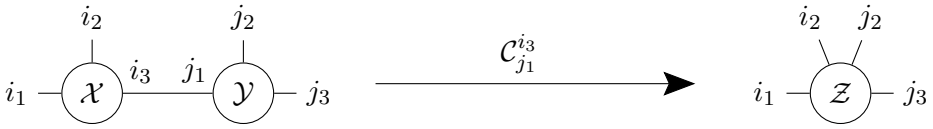
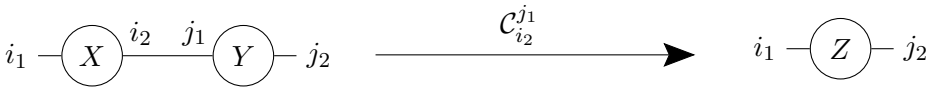


Figure 3.2: Representation in the Penrose notation of the contraction between two tensors $\mathcal{X} \in \mathbb{K}^{N_1 \times N_2 \times N_3}, \mathcal{Y} \in \mathbb{K}^{M_1 \times M_2 \times M_3}$ by their indices i_3 and j_1 with $N_1, N_2, N_3, M_1, M_2, M_3 \in \mathbb{N}$ and $N_3 = M_1$

Example 3.2. The tensor contraction for two tensors $M_1 \in \mathbb{K}^{N_1 \times N_2}$ and $M_2 \in \mathbb{K}^{N_2 \times N_3}$ over one edge of each tensor yields the matrix multiplication. Applying eq. (3.1.1) we get that $Z = C_2^2(X \otimes Y)$ is defined entry-wise as:

$$Z(i_1, j_2) = \sum_{s=1}^{N_2} M_1(i_1, s) M_2(s, j_2)$$

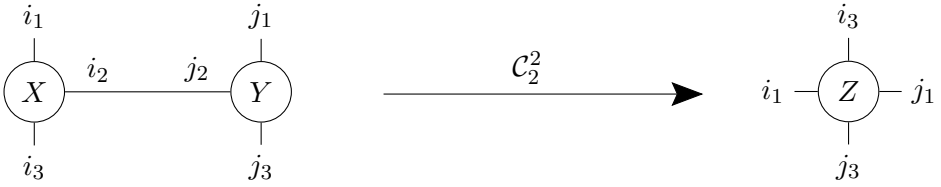
And that is identical to the conventional matrix product. Visually, we can represent it using the Penrsone notation:



Example 3.3. Suppose that we take $X \in \mathbb{R}^{3\times2\times2}$ from Example 2.23 and $Y \in \mathbb{R}^{2\times2\times3}$ defined as:

$$Y = \left[\begin{pmatrix} 1 & 0 & -1 \\ 0 & 2 & -2 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 2 & 0 & 2 \end{pmatrix} \right]$$

We want to compute the contraction of X and Y from the second index of each tensor $Z = C_2^2(X \otimes Y)$. Note that $Z \in \mathbb{R}^{3\times2\times2\times3}$. Using the Penrose Notation, the contraction would look like



And each element of the contracted tensor Z would be defined as:

$$Z(i_1, i_3, j_1, j_3) = \sum_{s=1}^{2} X(i_1, s, i_3) Y(j_1, s, j_3)$$

Therefore,

$$Z = \begin{bmatrix} \begin{pmatrix} 1 & 0 & -1 \\ 1 & 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 2 & -2 \\ 2 & 0 & 2 \end{pmatrix} \\ \begin{pmatrix} 1 & 2 & -3 \\ 3 & 1 & 2 \end{pmatrix} & \begin{pmatrix} 0 & 2 & -2 \\ 2 & 0 & 2 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 & -1 \\ 1 & 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \end{pmatrix} \end{bmatrix}$$

We can that the contraction that we defined on the Penrose notation (collapsing one edge of G) is the same as the formal definition of the contraction of two tensors over the modes that correspond to each edge.

Now we will be extending the Penrose notation for representing some special cases:

Identity tensors

Identity tensors on the Penrose notation are represented as free edges that are not connected to any tensor. We denote by I_n as the tensor identity of order n. I_2 are identity matrices.

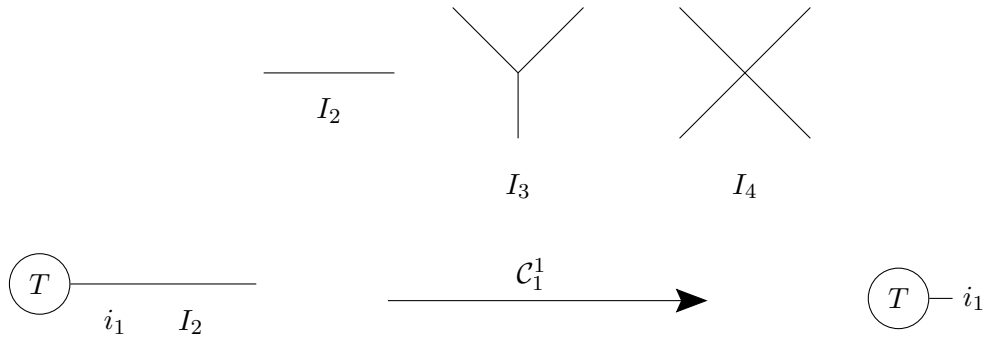


Figure 3.3: The identity matrix represented using the Penrose Notation. Contracting a mode over the identity yields the same tensor.

This makes complete sense since contracting some tensor along some mode over the identity matrix yields the same tensor. This is ilustrated in fig. 3.3

Trace

There may be the case that when contracting a series of tensors, we might end up as what we see as a loop in the Penrose Notation. Contracting over these two indexes we get the trace of the tensor $\mathcal{T}$ respect the indices i_k and i_p and we denote it as $\mathrm{Tr}_p^k(T)$ (See fig. 3.4). This operation is well defined since contracting over two modes of the same tensor is doing a tensor contraction using Definition 3.1.

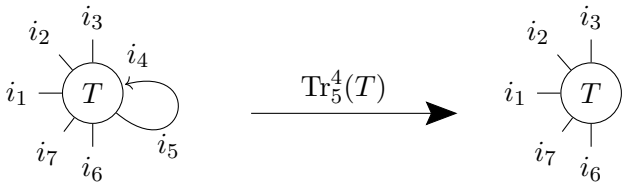


Figure 3.4: Representation of the trace of a tensor using the Penrose notation

Diagonal tensors

A diagonal tensor $\Lambda \in \mathbb{K}^{N\times\cdots\times N}$ of order n is a tensor which only has entries in his diagonals, i.e, there are $\lambda_1, \ldots, \lambda_n \in \mathbb{K}$ such that

$$\Lambda(i_1, \ldots, i_n) = \begin{cases} \lambda_j & \text{if } i_1 = \ldots i_n = j \\ 0 & \text{otherwise} \end{cases}$$

We will draw diagonal tensors as identity tensors but with an small circle on the middle.

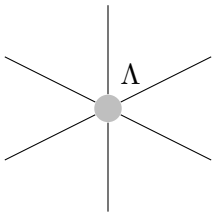


Figure 3.5: Diagonal tensor represented using the Penrose Notation

Outer product

And if two tensors are unconnected, we can represent it as one tensor but we do not perform any contraction between them since they are not connected. In this case, the outer product of the two tensors is performed.

Figure 3.6: Representation of the outer product using the Penrose notation

As we previously said, eq. (3.1.1) is equivalent to a matrix product, therefore it is possible to compute a tensor contraction by multiplying two tensors unfolded in a concrete way. the tensors $\mathcal{X}$ and $\mathcal{Y}$, we can compute $\mathcal{C}_l^k(X \otimes Y)$ as a matrix product.

Corollary 3.4. *Let* $X \in \mathbb{K}^{N_1 \times \cdots \times N_k \times \cdots \times N_n}, Y \in \mathbb{K}^{M_1 \times \cdots \times M_l \times \cdots \times M_m}$ *with* $1 \leqslant k \leqslant n$, $1 \leqslant l \leqslant m$, $N_k = M_l$. *The matrix product*

$$\mathcal{X}(\overline{i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_n}, \overline{i_k}) \cdot \mathcal{Y}(\overline{j_l}, \overline{j_1, \ldots, j_{l-1}, j_{l+1}, \ldots, j_m})$$

Results in a $\left(\prod_{i=1}^{n} N_i\right)/N_k \times \left(\prod_{i=1}^{m} M_i\right)/M_l$ *matrix, which can be reshaped back onto* $C_l^k(X \otimes Y)$.

Example 3.5. Following from Example 3.3, we will reshape the tensors X and Y for computing Z by only performing a matrix product. We need to first compute the matrices $\mathcal{X} = \text{unfold}\,(X, (1,3), (2)) \in \mathbb{R}^{6\times 2}$ and $\mathcal{Y} = \text{unfold}\,(Y, (2), (1,3)) \in \mathbb{R}^{2\times 6}$. These unfoldings result in:

$$X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad Y = \begin{pmatrix} 1 & 0 & -1 & 1 & 1 & 0 \\ 0 & 2 & -2 & 2 & 0 & 2 \end{pmatrix}$$

Therefore:

$$Z = XY = \begin{pmatrix} 1 & 0 & -1 & 1 & 1 & 0 \\ 0 & 2 & -2 & 2 & 0 & 2 \\ 1 & 2 & -3 & 3 & 1 & 2 \\ 0 & 2 & -2 & 2 & 0 & 2 \\ 1 & 0 & -1 & 1 & 1 & 0 \\ 0 & -2 & 2 & -2 & 0 & -2 \end{pmatrix}$$

And we can now reshape $Z \in \mathbb{R}^{9\times 9}$ as the tensor in $\mathbb{R}^{3\times 3\times 3\times 3}$ as

$$Z(i_1, i_3, j_1, j_3) = Z(\overline{i_1, i_3}, \overline{j_1, j_3})$$

which in fact, is identical to Z in Example 3.3

3.2 Tensor Network States

In this section we will present the formal definition of tensor network states and the definition of a tensor network. We will use directed graphs to represent tensor networks. Before that we will need to also define what are input and output edges:

Definition 3.6. *Given a directed graph $G = (V, \bar{E})$ and a vertex $i \in V$ we define*

$$\mathrm{IN}(i) = \{j \in V : (j, i) \in \bar{E}\} \qquad \mathrm{OUT}(i) = \{j \in V : (i, j) \in \bar{E}\}$$

The way tensor networks are constructed consists of picking a connected directed graph $G = (V, \bar{E})$, and for each vertex $i \in V$ we assign a vector space $\mathbb{V}_i$ and for each edge $(i, j) \in \bar{E}$ we assign a vector space $\mathbb{E}_i$ to the tail of the edge and its dual covector space $\mathbb{E}_i^*$ to the head of the edge. In other words, tensor networks can be seen as a series of contractions between a set of tensors $\mathcal{G}_1, \ldots, \mathcal{G}_n$ which once all of them are contracted results in a tensor of $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$

More formally, let $\mathbb{V}_1, \ldots, \mathbb{V}_d$ be vector spaces with $\dim \mathbb{V}_i = N_i, i = 1, \ldots, d$. Let $\mathbb{E}_1, \ldots, \mathbb{E}_c$ be finite vector spaces with $\dim \mathbb{E}_i = R_i, i = 1, \ldots, c$. For each vertex $i \in V$ we will associate the tensor product space

$$\xi_i := \left(\bigotimes\nolimits_{j\in \mathrm{IN}(i)} \mathbb{E}_j\right) \otimes \mathbb{V}_i \otimes \left(\bigotimes\nolimits_{j\in \mathrm{OUT}(i)} \mathbb{E}_j^*\right)$$

A tensor network is completly defined by the graph G and all the tensor product spaces $\xi_1, \ldots, \xi_n$. We will also associate to the tensor network a contraction mapping $\mathcal{C}_G$ defined by contracting factors in $\mathbb{E}_j$ with factors of $\mathbb{E}_j^*$:

$$\mathcal{C}_G : \bigotimes\nolimits_{i=1}^{n} \xi_i \longrightarrow \bigotimes\nolimits_{i=1}^{d} \mathbb{V}_i$$

Note that we have given this shapes to the tensors that we fix onto each vertex $i \in V$ because when we contract the whole graph, we will get a tensor of $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$. Since every directed edge (i, j) must point out of a vertex i and point into a vetex j, each copy of $\mathbb{E}_j$ is paired with one copy of $\mathbb{E}^*$, so the contraction $\mathcal{C}_G$ is well defined and it results in a tensor of $\mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ (See fig. 3.7)

By picking some tensors $\mathcal{G}_i \in \xi_i$ and evaluating them through $\mathcal{C}_G$ we get a tensor T that we will call **tensor network state**. We will call the tensors $\mathcal{G}_i$ **core tensors**.

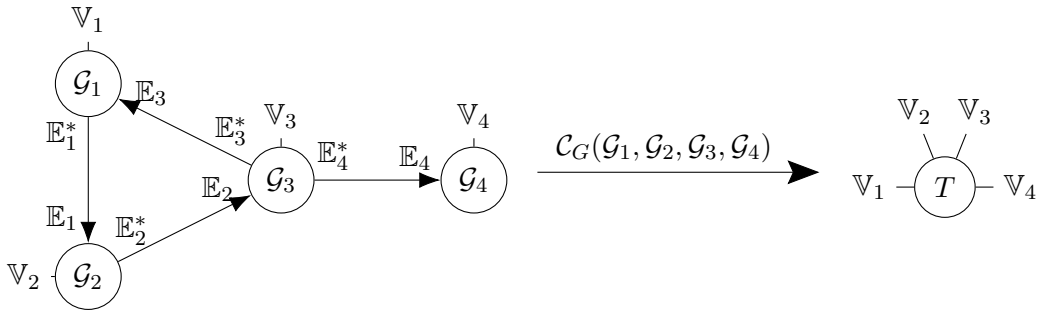


Figure 3.7: Example of a tensor network and a tensor network state evaluation using the Penrose notation.

Definition 3.7. *A tensor $T \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ is a tensor state of a tensor network if it can be written as $T = \mathcal{C}_G(\mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n)$ with $\mathcal{G}_i \in \xi_i$*

Definition 3.8. *We will define the set of all possible tensor states of a tensor network as the set* $\mathrm{TNS}(G; \mathbb{E}_1, \ldots, \mathbb{E}_c, \mathbb{V}_1, \ldots, \mathbb{V}_n)$*, i.e*

$$\mathrm{TNS}(G; \mathbb{E}_1, \ldots, \mathbb{E}_c, \mathbb{V}_1, \ldots, \mathbb{V}_n) := \left\{ \kappa_G(\mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n) \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n : \mathcal{G}_i \in \xi_i \right\}$$

The vector spaces $\mathbb{E}_1, \ldots, \mathbb{E}_c, \mathbb{V}_1, \ldots, \mathbb{V}_n$ are not really important on the Penrose representation of the tensor network, and we will usually not write them. Also, since all vector spaces are determined up to isomorphism by its dimension, we will write the tensor network as $\mathrm{TNS}(G; R_1, \ldots, R_c, N_1, \ldots, N_n)$.

And since n is equal to the number of vertices of G and c is equal to the number of edges of G, we will write $\mathrm{TNS}(G; R)$ for a more compact notation.

Definition 3.9. *Given a tensor network state* $\mathrm{TNS}(G, R)$*, from the graph given by its Penrose Notation, we will call the edges with a dangling end free edges, and the edges that connect two vertex contracted edges*

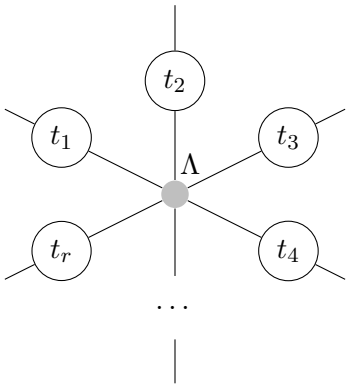
For example, in fig. 3.7, the edges labeled as $\mathbb{V}_1, \mathbb{V}_2, \mathbb{V}_3, \mathbb{V}_4$ are free edges and the rest are contracted edges.

3.3 Common Tensor network structures

Example 3.10. [Canonical polyadic decomposition] As we have seen, canonocal polyadic decomposition consists on decomposing T as a sum of r 1-rank tensors. We can write

$$\sum_{p=1}^{r} \lambda_p v_i^1 \otimes v_i^2 \otimes \cdots \otimes v_i^n$$

As the contraction of a diagonal tensor Λ of order r with its enties being λ_p and the other 1-rank tensors. If we denote $t_i = v_i^1 \otimes v_i^2 \otimes \cdots \otimes v_i^n$ so that the decomposition can be written as $\sum_{p=1}^{r} \lambda_p t_p$, the resulting tensor network has the following star shape:



Example 3.11. A tensor train decomposition or matrix product state of T are a set of 3th-order tensors $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$ with $\mathcal{G}_i \in \mathbb{K}^{R_{i-1} \times N_i \times R_i}$ and $R_0 = R_n = 1$ such that every element of T is written in the form

$$T(i_1, i_2, \ldots, i_n) = \sum_{r_0, \ldots, r_n}^{R_0, \ldots, R_n} \mathcal{G}_1(r_0, i_1, r_1)\mathcal{G}_2(r_1, i_2, r_2) \cdots \mathcal{G}_n(r_{n-1}, i_n, r_n) \tag{3.3.1}$$

We denote $R_0, R_1, \ldots, R_n$ as the ranks of the tensor train decomposition, or TT-ranks.

We can easily see that the tensor train decomposition (or TT) is obtained by our definition of a tensor network when G is a path, also the contraction of the whole network yields eq. (3.3.1)

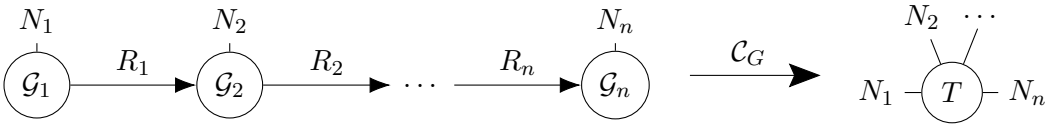


Figure 3.8: Tensor Train decomposition

Tensor train decompositions are very well studied because since they are a chain of products of matrices, there are a lot of mathematical theory that can be applied in this structure, for example, for finding the cores of the TT decomposition one could fix a core, reshape and unfold onto a matrix the objective tensor T, apply then SVD and update the cores according to the decomposition. A more detailed algorithm can be found in [10]. Since in this thesis is more general and not focused on tensor train decompositions we will skip this algorithm.

Example 3.12. Tensor ring decomposition (or TR) or also known a matrix product state with periodic boundary conditions, is obtained when G is a cycle.

Tensor Ring decomposition is considered generalization of Tensor Train decomposition, it's contraction is the same as eq. (3.3.1) but removing the condition $R_0 = R_1 = 1$. Tensor Rings have also been widely studied because of their circular invariance: shifting the cores on the tensor network results in an equivalent tensor network that once its evaluated results in the objective tensor with its dimensions fixed. Thanks to this fact, one can still apply SVD decomposition on tensor rings [14].

Example 3.13. The fully connected tensor network decomposition is obtenied when G is a complete graph.

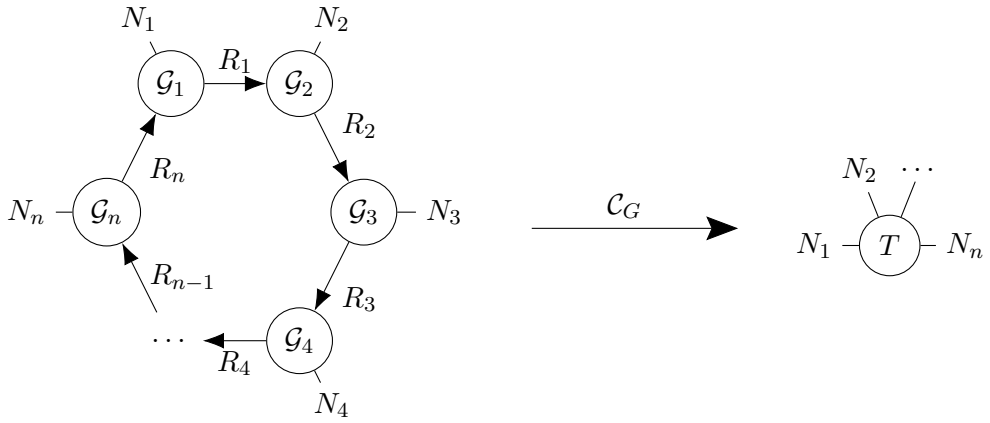


Figure 3.9: Tensor Ring (TR) decomposition

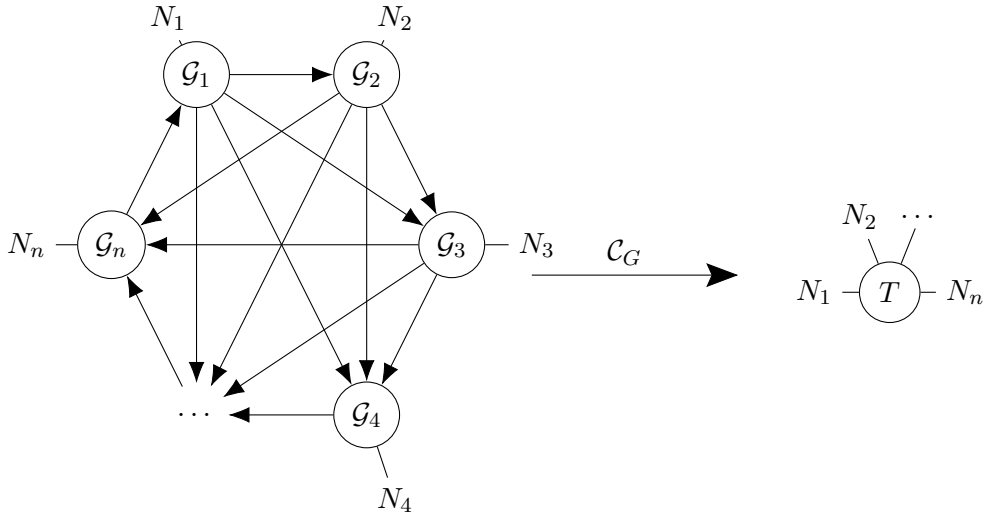


Figure 3.10: Fully connected tensor network decomposition (FCTN)

Example 3.14. A tree tensor network decomposition is obtained when G is a tree graph. We will see that tensor tree networks have the the nice property that if we can represent a tensor T in some state of the network, the G-rank of T is unique. We will see this in more detail on the next chapter.

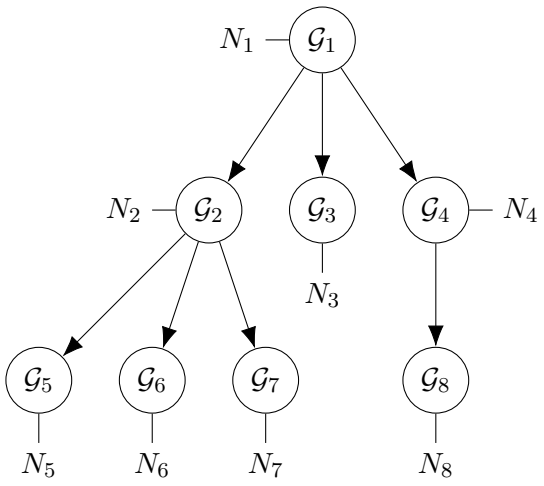


Figure 3.11: A tree tensor network

3.4 Tensor Network Ranks

In this section we will present the notion of G-rank of a tensor. The G-rank of a tensor is defined as the minimum ranks that a tensor T can be represented by a tensor state. Knowing the G-rank of a tensor will be useful because we could find cores $\mathcal{G}_1, \ldots, \mathcal{G}_n$ such that we could represent T, and surely enough, with a enough sotisficated algorithm, it could converge to these core tensors. Our main goal will be to try to estimate the G-rank of a tensor, since finding the G-rank explicitly is a very hard task. Approximating the G-rank will be enough for us because in the next chapter we will slightly vary the approximated G-rank for finding a good representation.

As we defined the rank for a tensor T as the minimum number of 1-rank tensors that compose the tensor T, we will define the G-rank as the minimum ranks that a tensor network state needs for representing T. More formally,

Definition 3.15 (Tensor G-rank)**.** *Given a graph G, we define the tensor rank respect to a G or G-rank as*

$$\operatorname{rank}_G(T) = \min\{(R_1, \ldots, R_c) \in \mathbb{N}^c : T \in \operatorname{TNS}(G; R_1, \ldots, R_c, N_1, \ldots, N_d)\}$$

Where $\min(S)$ *with* $S \subset \mathbb{N}^c$ *denotes the minimal elements of S. We treat $\mathbb{N}^c$ with its usual partial order:*

$$(a_1, \ldots, a_c) \leqslant (b_1, \ldots, b_c) \iff a_1 \leqslant b_1, a_2 \leqslant b_2, \ldots, a_c \leqslant b_c$$

So for example if $S = \{(3,4,5), (2,1,3), (1,3,2)\}$*, then* $\min(S) = \{(2,1,3), (1,3,2)\}$

Now, we will see that $\operatorname{rank}_G(T)$ is a finite set. The following theorem gives us that if we make $R_1, \ldots, R_c$ big enough, every tensor T can be a state of $\operatorname{TNS}(G; R_1, \ldots, R_c, N_1, \ldots, N_n)$. In fact, these values that guarantee that T is an state are $R_1 = \ldots R_c = \operatorname{rank} T$

Theorem 3.16. *Let $T \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ and let G be a connected graph with n vertices and c edges. There exists $R_1, \ldots, R_c \in \mathbb{N}$ such that*

$$T \in \operatorname{TNS}(G; R_1, \ldots, R_c, N_1, \ldots, N_d)$$

in fact, we can choose $R_1 = \cdots = R_c = \operatorname{rank} T$

Proof. Let $r = \operatorname{rank} T$. Then there exist $v_1^{(i)}, \ldots, v_r^{(i)} \in \mathbb{V}_i, i = 1, \ldots, n$ such that

$$T = \sum_{p=1}^{r} v_1^{(p)} \otimes \cdots \otimes v_n^{(p)}$$

We take $R_1 = \cdots = R_c = r$ we take for each $i = 1, \ldots, n$

$$\mathcal{G}_i = \sum_{p=1}^{r} \left(\bigotimes\nolimits_{j \in \mathrm{IN}(i)} e_p^{(j)} \right) \otimes v_p^{(i)} \otimes \left(\bigotimes\nolimits_{j \in \mathrm{OUT}(i)} e_p^{(j)*} \right)$$

Now observe that for each $i = 1, \ldots, n$ there exists an unique h such that whenever $j \in \mathrm{IN}(i) \cap \mathrm{OUT}(i)$, $e_p^{(j)}$ and $e_p^{(j)*}$ contract and give δ_{pq}, therefore the summand vanishes except when $p = q$. This together with the assumption that G is a connected graph implies that $\kappa_G(\mathcal{G}_1 \otimes \cdots \otimes \mathcal{G}_n)$ reduces to a sum of terms of the form $v_p^{(1)} \otimes \cdots \otimes v_p^{(d)}$ for $p = 1, \ldots, r$, which is of course T □

Now, by picking the ranks as $R_1 = \cdots = R_c = \operatorname{rank} T$ is usually not optimal, since we will end up that our tensor network will use more memory to represent T than the memory we need to write T itself.

We will proceed to show that the G-rank can be a lot more smaller than the tensor rank, and in fact, we will show some examples of tensor networks that can represent tensors in a more efficient way than the canonical polyadic decomposition that we have seen in chapter 2.

The following theorem says that there exists some tensor networks and some tensors that make this claim true:

Theorem 3.17. *For $n \geqslant 3$ there exists a connected simple graph G with n vertices and c edges such there exists a tensor $T \in \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_n$ with tensor rank $\operatorname{rank}(T) = r$ is much larger than the G-rank $\operatorname{rank}_G(T) = (r_1, \ldots, r_n)$. More specifically,*

$$r \gg r_1 + \cdots + r_n$$

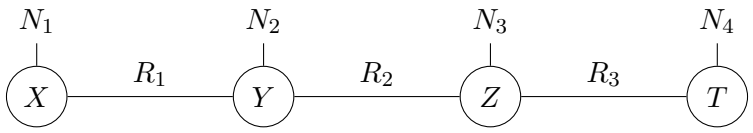
The proof of this theorem can be found in [13]. In the paper the authors find that for representing a tensor network that contracts to a tensor $T \in (\mathbb{C}^{n \times n})^* \otimes (\mathbb{C}^{n \times n})^* \otimes \mathbb{C}^{n \times n} \cong \mathbb{C}^{n \times n \times n}$ picking G as C_3 and P_3 the inequality holds for $n = 3$. Then, the proof constructs from T for $n > 3$. We won't enter in more details onto the proof of the theorem.

3.5 Contracting tensor networks

We will finish this chapter by giving an algorithm for contracting (or evaluating) a tensor network.

Evaluating a tensor network is as easy as contracting each edge of G. The order in which we contract does not affect the resulting tensor, as we can see from Equation (2.1.3). But instead the order affects the computational cost, as we can see in the following example:

Example 3.18. Consider the following tensor train network:



The contraction of the whole network is given by

$$T_{ijkl} = \sum_{p,q,r}^{R_1,R_2,R_3} X_{ip} Y_{pjq} Z_{qkr} T_{rl}$$

We could contract first R_1, then R_2 and finally R_3. That means computing:

$$A_{ijq} = \sum_{p}^{R_1} X_{ip} Y_{pjq} \qquad B_{ijkr} = \sum_{q}^{R_2} A_{ijq} Z_{qkr} \qquad T_{ijkl} = \sum_{r}^{R_3} B_{ijkr} T_{rl}$$

For this computation, we would need $R_1R_2N_1N_2 + R_2R_3N_1N_2N_3 + R_3N_1N_2N_3N_4$ products, and we would need to store first A, then B and then T

If we consider a different ordering for the contraction, for example first R_1, then R_3 and finally R_2 we get a different computation cost. We would need to compute the matrices

$$A'_{ijq} = \sum_{p}^{R_1} X_{ip} Y_{pjq} \qquad B'_{qkl} = \sum_{r}^{R_3} Z_{qkr} T_{rl} \qquad T_{ijkl} = \sum_{q}^{R_2} A'_{ijq} B'_{qkl}$$

The result of the contraction is indeed the same, but now instead the number of computations is $R_1R_2N_1N_2 + R_3N_1N_3N_4 + R_2N_1N_2N_3N_4$.

In [11], the authors discuss different algorithms for finding the optimal contraction order. First, the exhaustive search of the order is considered. It consists in searching all the possible contraction orders and finding the one that has less computational cost. The authors show that the exhaustive search algorithm scales like $\mathcal{O}(e^E)$ where E is the number of edges of the network.

In our case, we will use a greedy search algorithm which consists on contracting over one edge at a time and for each contraction step we take the one with the lowest cost. We can see k steps further and pick the one that has the less incremental cost. The computational complexity of the greedy algorithm scales lake $\mathcal{O}(E^k)$ where k is the number of next steps that we consider when picking the next contraction.

We will finish this section by providing a pseudocode of the greedy algorithm for contraction order selection:

Algorithm 1 Greedy contraction order selection

Input: A tensor $T \in \mathbb{K}^{N_1 \times \cdots \times N_n}$ and some fixed error ϵ
Output: Core tensors $\mathcal{G}_1, \ldots, \mathcal{G}_n$

Initialize tensors $\mathcal{G}_1, \ldots, \mathcal{G}_n$
while $\|T - \kappa_G(\mathcal{G}_1, \ldots, \mathcal{G}_n)\|_F > \epsilon$ **do**
    for $k = 1, \ldots, n$ **do**
        $\mathcal{G}_m \leftarrow \arg\min_{G_m} \|G^{\neq m} G_m - T^{(m)}\|_2$
return $\mathcal{G}_1, \ldots, \mathcal{G}_n$

Chapter 4

Tensor network state search

In this chapter we will study methods for finding optimal TNS such that $T \in \text{TNS}(G; R)$ or at least $T + E \in \text{TNS}(G; R)$ where E is a tensor such that $\|E\|_F \simeq 0$ that describes the absolute error of the representation of the state.

Suppose that we have already found some optimal G and R for representing T. We will start the chapter by describing the **alternated least squares** or ALS algorithm. The goal of the ALS algorithm is finding cores $\mathcal{G}_1, \ldots, \mathcal{G}_c$ of $\text{TNS}(G; R)$ given T, G and R.

4.1 The Alternating Least Squares algorithm

We will denote $T \in \mathbb{K}^{N_1 \times \cdots \times N_n}$ as our objective tensor. Let $G = (V, E)$, $c = \#E$ and $R = (r_1, \ldots, r_c)$. We would want to find some cores $\mathcal{G}_1, \ldots, \mathcal{G}_c$ of $\text{TNS}(G; R)$ such that minimize

$$\|T - \mathcal{C}_G(\mathcal{G}_1, \ldots, \mathcal{G}_c)\|_F$$

If we impose that all the tensor cores remain fixed except $\mathcal{G}_m$, our problem would become

$$\arg\min_{\mathcal{G}_m} \|T - \mathcal{C}_G(\mathcal{G}_1, \ldots, \mathcal{G}_n)\|_F$$

Now, we apply our contraction mapping $\mathcal{C}_G$ for all cores excluding $\mathcal{G}_m$ (figs. 4.1 and 4.2). We will call this contracted tensor $\mathcal{G}^{\neq m}$.

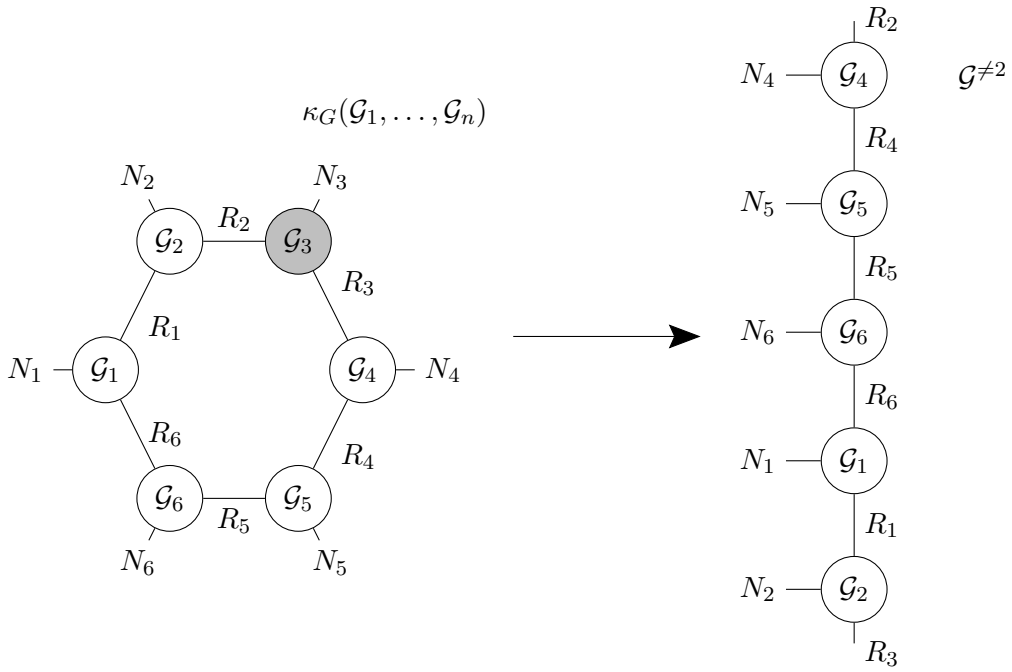


Figure 4.1: The representation of $\mathcal{G}^{\neq m}$ on the TR decomposition, with $m = 2$

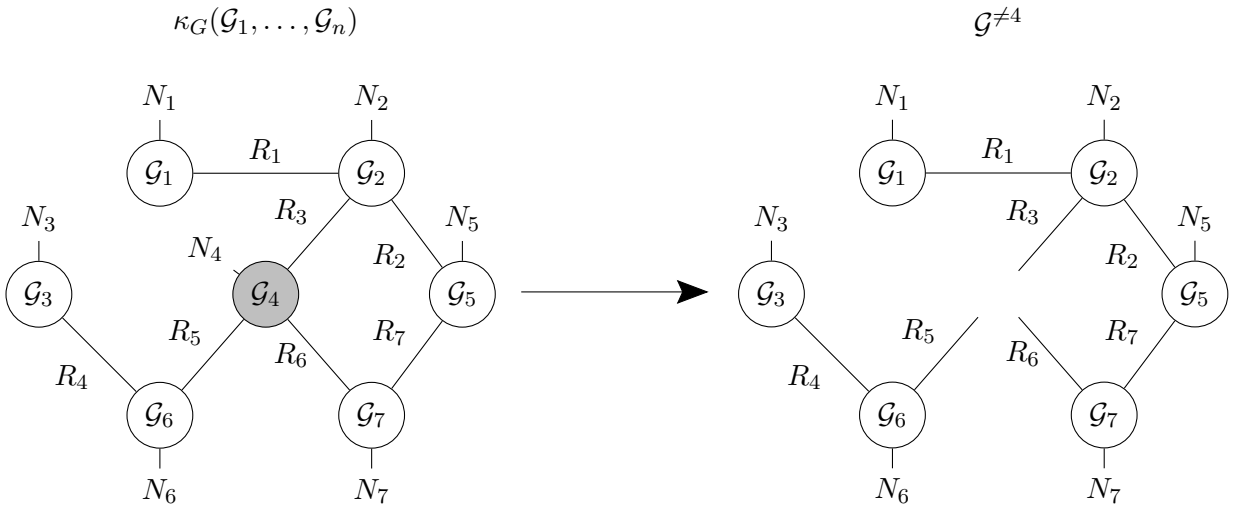


Figure 4.2: The representation of $\mathcal{G}^{\neq 4}$ over some arbitrary TN

The last contractions that remain between $\mathcal{G}^{\neq m}$ and $\mathcal{G}_m$ are equivalent to evaluating the whole network. Now, if we consider appropiate matricizations $T^{(m)}$, $G^{\neq m}$ and G_m, these contractions can be computed calculating the matrix product $G^{\neq m}G_m$, so our minimization problem is now equivalent to solving the following least squares problem:

$$\arg\min_{G_m} \|G^{\neq m}G_m - T^{(m)}\|_F \tag{4.1.1}$$

Let $x^{(i)}$ be the i-th column of G_m and $y^{(i)}$ the i-th column of $T^{(m)}$. Solving 4.1.1 means solving for each i

$$\arg\min_{x^{(i)}} \|G^{\neq m}x^{(i)} - y^{(i)}\|_2 \tag{4.1.2}$$

Since we can't assure that there exists an exact solution to $G^{\neq m}x^{(i)} = y^{(i)}$, we can use the solution to the normal equation $(G^{\neq m})^T G^{\neq m}x^{(i)} = (G^{\neq m})^T y^{(i)}$.

Once we have solved (4.1.2), we can reshape back G_m to $\mathcal{G}_m$. We can iteratively change the varying core tensor $\mathcal{G}_i$ until the contraction of the whole tensor network is T with some fixed error ϵ. The Tensor Network ALS algorithm does this entire process:

Algorithm 2 Tensor Network ALS

Input: A tensor network state $T \in \mathbb{K}^{N_1 \times \cdots \times N_n}$ and some fixed error ϵ
Output: Core tensors $\mathcal{G}_1, \ldots, \mathcal{G}_n$

Initialize tensors $\mathcal{G}_1, \ldots, \mathcal{G}_n$
while $\|T - \kappa_G(\mathcal{G}_1, \ldots, \mathcal{G}_n)\|_F > \epsilon$ **do**
    for $k = 1, \ldots, n$ **do**
        $\mathcal{G}_m \leftarrow \arg\min_{G_m} \|G^{\neq m} G_m - T^{(m)}\|_2$
return $\mathcal{G}_1, \ldots, \mathcal{G}_n$

4.2 Initial rank determination

4.3 Structure search

Tensor network structure search prioblem aims to generally find the most compressed tensor network models for computational purposes while maintaining the expressivity of the network. That means, that our tensor network is capable of giving good representations of the set of tensors that we want to represent with it.

By now, we know that for any tensor T, if we choose a graph $G = (V, E)$ we can choose $R_1, \ldots, R_c \leqslant \operatorname{rank} T \leqslant \left\lfloor \frac{\prod_{i=1}^n N_i}{\sum_{i=1}^n N_i} \right\rfloor$ such that $T \in \mathrm{TNS}(G, R)$. We can also find thanks to the ALS algorithm the cores $\mathcal{G}_1, \ldots, \mathcal{G}_n$ of $\mathrm{TNS}(G, R)$. The only thing that remains is how we pick an optimal G for compressing T with a fixed error ϵ, since we know that the optimal graph for a tensor network state of a given tensor T depends on the underlying data of T itself.

Suppose that we have defined a loss function $\pi_D : \mathbb{R}^{N_1 \times N_2 \times \cdots \times N_n} \to \mathbb{R}_+$ involving D, with D being a dataset. For example, if our objective is compressing an objective tensor $T_o \in \mathbb{R}^{N_1 \times N_2 \times \cdots \times N_n}$, we could define our loss function as the relative error of our evaluated tensor respect to the objective tensor:

$$\pi_D(T) = \frac{\|T - T_o\|_F}{\|T_o\|_F}$$

The tensor network structure search problem is defined as solving the following discrete optimization problem:

$$\min_{(G,R) \in \mathbb{G} \times \mathbb{F}_G} \left(\phi(G, R) + \lambda \cdot \min_{\mathcal{Z} \in TNS(G,R)} \pi_D(\mathcal{Z}) \right) \tag{4.3.1}$$

where $\mathbb{G}$ is the space of all graphs, $G \in \mathbb{G}$ is a graph of N vertices and K edges, $R = (r_1, \ldots, r_K) \in \mathbb{F}_G \subseteq \mathbb{Z}_+^K$ are the ranks of the tensor network, $\phi(G, R)$ represents a function that determines the complexity of the tensor network, and $\lambda > 0$ is a tuning parameter.

The intuition behind presenting this optimization problem is that the inner minimization that depends on the tuning parameter λ serves to evaluate the expressivity of the tensor network.

We call the problem 4.3.1 as Tensor Network Structure Selection (TN-SS). TN-SS is a very generalized problem. There has been a lot of research on solving it under certain conditions:

- Tensor Network Rank Selection (TN-RS), it restricts G to be a fixed graph, and its objective is to find the tensor network ranks $R \in \mathbb{F}_G$.
- Tensor Network Permutation Selection (TN-PS) fixes the ranks R and search over the set of all the simple graphs that are isomorphic to G.
- Lastly, Tensor Network Topology Selection (TN-TS) searches over the set of all simple graphs of N vertices and the tensor ranks R are fixed

In this chapter we will present the tensor network structure search algorithm using alternating local enumeration (TnALE) [4]

The purpose of the TnALE algorithm is to solve 4.3.1, but for each evaluation that it does, it needs another algorithm for evaluating the inner minimum inside the equation. It requires that we already know how to compute $\min_{\mathcal{Z} \in TNS(G,R)} \pi_D(\mathcal{Z})$

We first start by rewriting (4.3.1) with a more general form. We define $\mathbb{P}$ as the set of all functions of the form $p : \mathbb{Z}_+^K \to \mathbb{K}_+^L$

$$\min_{x \in \mathbb{Z}_+^K, p \in \mathbb{P}} f_p(x) := f \circ p(x) \tag{4.3.2}$$

Where $f_p : \mathbb{Z}_+^L \to \mathbb{R}_+$ is a generalization of the objective function, and $p : \mathbb{Z}_+^K \to \mathbb{Z}_+^L \in \mathbb{P}$ correspond to the topology-related variable (G, R).

We can show this correspondance in the following way: since G is completly defined by its adjacency matrix A which is defined as:

$$A_{ij} := \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

We can encode the ranks R on the adjacency matrix by changing its definition as the following way:

$$A_{ij} := \begin{cases} R_{(i,j)} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Where $R_{(i,j)}$ denotes the rank of the edge $(i.j)$. We will denote this matrix as A_R

Example 4.1. Let $G = P_4$ and $R = (1, 2, 3)$. We associate the edges of G with the ranks R as $R_{(1,2)} = r_1 = 1$, $R_{(2,3)} = r_2 = 2$ and $R_{(3,4)} = r_3 = 3$. We consider TNS$(G; R)$. The adjacency matrix of G and the encoded rank matrix of TNS$(G; R)$ is, respectively:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad A_R = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 2 & 0 & 3 \\ 0 & 0 & 3 & 0 \end{pmatrix}$$

Given G and R, we can encode $x \in \mathbb{R}_+^K$ as a vector with the rank of each edge of the graph and $p \in \mathbb{P}$ as a permutation matrix. By varying only p we get all the permutations of the graph G, and by x we get vary the ranks of the edges.

Example 4.2.

$$A_R^1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 2 & 0 & 3 \\ 0 & 0 & 3 & 0 \end{pmatrix} \Leftrightarrow p_1(x) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 2 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 2 \\ 0 \\ 3 \end{pmatrix}$$

$$A_R^2 = \begin{pmatrix} 0 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 0 & 2 & 0 \end{pmatrix} \Leftrightarrow p_1(x) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 2 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 3 \\ 1 \\ 0 \\ 2 \end{pmatrix}$$

So, now our problem is reduced to (4.3.2). We can now apply theory of gradient-less optimizations in the real domain. We will start by defining the finite gradient:

4.4 Gradient-less optimization

Definition 4.3. *For any function $f : \mathbb{Z}_+^L \to \mathbb{R}$ its finite gradient $\nabla f : \mathbb{Z}_+^L \to \mathbb{R}$ at a point $x \in \mathbb{L}_+^L$ is defined as*

$$\nabla f(x) = [f(x + e_1) - f(x), \ldots, f(x + e_L) - f(x)]^T$$

Chapter 5

Conclusions

TODO

Fent servir un símil geomètrico-cartogràfic, aquesta memòria constitueix un mapa a escala planetària de la demostració de la conjectura feble de Goldbach presentada per Helfgott i un mapa a escala continental de la verificació numèrica d'aquesta. Estudis posteriors i més profunds haurien de permetre elaborar mapes de menor escala.

La naturalesa dels nombres primers ens ha portat per molts racons diferents de les Matemàtiques; en no imposar-nos restriccions en la forma de pensar, hem pogut gaudir del viatge i assolir els objectius que ens vam plantejar a l'inici del projecte i anar més enllà, sobretot en el camp de la computació i la manipulació de grans volums de dades numèriques.

Una gran part dels coneixements bàsics que hem hagut de fer servir han estat treballats en les assignatures de Mètodes analítics en teoria de nombres i d'Anàlisi harmònica i teoria del senyal, que són optatives de quart curs del Grau de Matemàtiques. Altres els hem hagut d'aprendre durant el desenvolupant del projecte. S'ha realitzat una tasca de recerca bibliogràfica important, consultant recursos antics i moderns, tant en format digital com en format paper.

Bibliography

[1] Matthias Christandl, Asger Kjærulff Jensen, and Jeroen Zuiddam. "Tensor Rank Is Not Multiplicative under the Tensor Product". In: *Linear Algebra and its Applications* 543 (Apr. 2018), pp. 125–139. ISSN: 00243795. DOI: 10.1016/j.laa.2017.12.020. arXiv: 1705.09379 [math]. URL: http://arxiv.org/abs/1705.09379 (visited on 05/28/2025).

[2] Zheng Guo et al. *Tensor Network Structure Search Using Program Synthesis*. Feb. 22, 2025. DOI: 10.48550/arXiv.2502.02711. arXiv: 2502.02711 [cs]. URL: http://arxiv.org/abs/2502.02711 (visited on 05/12/2025). Pre-published.

[3] Christopher Hillar and Lek-Heng Lim. *Most Tensor Problems Are NP-hard*. July 1, 2013. DOI: 10.48550/arXiv.0911.1393. arXiv: 0911.1393 [cs]. URL: http://arxiv.org/abs/0911.1393 (visited on 03/31/2025). Pre-published.

[4] Chao Li et al. *Alternating Local Enumeration (TnALE): Solving Tensor Network Structure Search with Fewer Evaluations*. May 29, 2023. DOI: 10.48550/arXiv.2304.12875. arXiv: 2304.12875 [cs]. URL: http://arxiv.org/abs/2304.12875 (visited on 05/15/2025). Pre-published.

[5] Chao Li et al. *Permutation Search of Tensor Network Structures via Local Sampling*. June 14, 2022. DOI: 10.48550/arXiv.2206.06597. arXiv: 2206.06597 [cs]. URL: http://arxiv.org/abs/2206.06597 (visited on 03/24/2025). Pre-published.

[6] Osman Asif Malik, Vivek Bharadwaj, and Riley Murray. *Sampling-Based Decomposition Algorithms for Arbitrary Tensor Networks*. Oct. 7, 2022. DOI: 10.48550/arXiv.2210.03828. arXiv: 2210.03828 [math]. URL: http://arxiv.org/abs/2210.03828 (visited on 04/30/2025). Pre-published.

[7] Román Orús. "Tensor Networks for Complex Quantum Systems". In: *Nature Reviews Physics* 1.9 (Sept. 2019), pp. 538–550. ISSN: 2522-5820. DOI: 10.1038/s42254-019-0086-7. URL: https://www.nature.com/articles/s42254-019-0086-7 (visited on 05/14/2025).

[8] I. V. Oseledets. "Tensor-Train Decomposition". In: *SIAM Journal on Scientific Computing* 33.5 (Jan. 2011), pp. 2295–2317. ISSN: 1064-8275. DOI: 10.1137/090752286. URL: https://epubs.siam.org/doi/10.1137/090752286 (visited on 03/24/2025).

[9] Roger Penrose. "Applications of Negative Dimensional Tensors". In: *Combinatorial Mathematics and its Applications* (1971), pp. 221–244. URL: https://homepages.math.uic.edu/~kauffman/Penrose.pdf (visited on 03/04/2025).

[10] Melven Röhrig-Zöllner, Jonas Thies, and Achim Basermann. "Performance of the Low-Rank Tensor-Train SVD (TT-SVD) for Large Dense Tensors on Modern Multi-Core CPUs". In: *SIAM Journal on Scientific Computing* 44.4 (Aug. 2022), pp. C287–C309. ISSN: 1064-8275, 1095-7197. DOI: 10.1137/21M1395545. arXiv: 2102.00104 [math]. URL: http://arxiv.org/abs/2102.00104 (visited on 05/28/2025).

[11] Frank Schindler and Adam S. Jermyn. "Algorithms for Tensor Network Contraction Ordering". In: *Machine Learning: Science and Technology* 1.3 (Sept. 1, 2020), p. 035001. ISSN: 2632-2153. DOI: 10.1088/2632-2153/ab94c5. arXiv: 2001.08063 [cs]. URL: http://arxiv.org/abs/2001.08063 (visited on 04/01/2025).

[12] Steven R. White. "Density Matrix Formulation for Quantum Renormalization Groups". In: *Physical Review Letters* 69.19 (Nov. 9, 1992), pp. 2863–2866. DOI: 10.1103/PhysRevLett.69.2863. URL: https://link.aps.org/doi/10.1103/PhysRevLett.69.2863 (visited on 03/27/2025).

[13] Ke Ye and Lek-Heng Lim. *Tensor Network Ranks*. Feb. 9, 2019. DOI: 10.48550/arXiv.1801.02662. arXiv: 1801.02662 [math]. URL: http://arxiv.org/abs/1801.02662 (visited on 03/24/2025). Pre-published.

[14] Qibin Zhao et al. *Tensor Ring Decomposition*. June 17, 2016. DOI: 10.48550/arXiv.1606.05535. arXiv: 1606.05535 [cs]. URL: http://arxiv.org/abs/1606.05535 (visited on 03/04/2025). Pre-published.

Appendix A

Chapter 1