Università di Pisa

Doctoral Thesis

A Tensor Framework for Learning in Structured Domains

Author:
Daniele Castellana

Supervisors:
Prof. Davide Bacciu

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Department of Computer Science

April 30, 2021

UNIVERSITÀ DI PISA

Abstract

Department of Computer Science

Doctor of Philosophy

A Tensor Framework for Learning in Structured Domains

by Daniele Castellana

Tensors have been recently emerging as a popular tool in the machine learning community. This interest is firstly motivated by the natural representation of multi-modal data as tensors. In this context, tensors are considered a generalisation of arrays to the multi-dimensional case. Indeed, tensors are more than mere containers: they are powerful mathematical objects which are strictly related to multi-linear algebra. A more comprehensive application of tensors and their associated multi-linear algebra led to their use in representing and compressing parameters of machine learning models.

Despite such interest, little attention has been paid on leveraging tensor methods to model high-order interactions among information flowing in a learning model. On the other hand, learning machines for structured data (e.g., trees) are intrinsically based on their capacity to learn representations by aggregating information from the multi-way relationships captured in the structure topology. While complex aggregation functions are desirable in this context to increase expressiveness of the learned representations, the modelling of high-order interactions among structure constituents is unfeasible in practice due to the exponential number of parameters required.

The aim of this thesis is to build a bridge between tensors and adaptive structured data processing, providing a general framework for learning in structured domains which has tensor theory at its backbone. To this end, we show that tensors arise naturally in model parameters from the formulation of learning problems in structured domains. We propose to approximate such parametrisations leveraging tensor decompositions whose hyper-parameters regulate the trade-off between expressiveness and compression ability. Moreover, we show that each decomposition introduces a specific inductive bias to the model. Another contribution of the thesis is the application of these new approximations to unbounded structures, where tensor decompositions needs combining with weight sharing constraints to control model complexity. The last contribution of our work is the development of two Bayesian non-parametric models for structures which learn to adapt their complexity directly from data.

Contents

List of Figures

List of Tables

List of Symbols

Tensors

$\mathbb{R}$	The set of real numbers.
$\mathbb{R}_{\geq 0}$	The set of positive real numbers.
I, i	Real scalars.
$\boldsymbol{v}$	A real vector.
$\boldsymbol{M}$	A real matrix.
$\underline{\boldsymbol{T}}$	A real tensor.
$\bar{\boldsymbol{v}}$	A real vector in homogeneous coordinate.
$\underline{\boldsymbol{T}}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_D)$	The application of the multi-linear function associated to a tensor.
$\underline{\boldsymbol{T}}[i_1, \ldots, i_D]$	Indexing of a tensor.
$\underline{\boldsymbol{T}}[:, i_2, :, i_4]$	A sub-tensor of a tensor.
$\mathrm{vec}(\underline{\boldsymbol{T}})$, $\underline{\boldsymbol{T}}_{(n)}$	Vectorisation and n-mode matricisation of a tensor.
$\boldsymbol{a} \odot \boldsymbol{b}$	Element-wise multiplication.

Structures

$\boldsymbol{\mathcal{D}}$	A structure.
v	A node.
(u, v, l)	An edge between node u and node v in position l.
$\mathrm{vert}(\boldsymbol{\mathcal{D}})$	The set of all the nodes in $\boldsymbol{\mathcal{D}}$.
$\mathrm{edg}(\boldsymbol{\mathcal{D}})$	The set of all the edges in $\boldsymbol{\mathcal{D}}$.
$\mathrm{ch}(v)$	The set of the child nodes of v.
$\mathrm{pa}(v)$	The set of the parent nodes of v.
$\mathrm{de}(v)$	The set of all the descendant nodes of v.
$\boldsymbol{\mathcal{D}}_v$	The sub-structure in $\boldsymbol{\mathcal{D}}$ which contains only the node v and $\mathrm{de}(v)$.
$\#_L^P$	The set of all structures with maximum out-degree of L and a maximum in-degree of P.
$\mathrm{skel}(\boldsymbol{\mathcal{X}})$	The skeleton of a labelled structure.
$\mathcal{X}$	A domain for the labels attached to a structure.
$\mathcal{X}^{\#_L^P}$	The set of labelled structures whose labels are in $\mathcal{X}$ and whose skeletons are in $\#_L^P$.

Random Variables

H	A random variable.

h	A realisation of the random variable H.

Constants

N	The number of the training examples.
C	The size of the hidden encoding space.
R	The rank of the tensor decompositions.
L	The maximum out-degree of a structure.
M	The number of the possible input categorical labels.
K	The number of the possible output categorical labels.

Chapter 1

Introduction

1.1 Motivations

In many real-world domains (e.g. chemistry, natural language processing, document processing, code analysis), information is naturally represented as structured data. A structured data comprises a set of atomic entities (usually referred to as vertices or nodes) and a structure. Vertices contain pieces of information of the domain of interest, while the structure depicts how these pieces are related. For example, the sentence "The sky is blue and the grass is green" is obtained by composing the two sub-phrases "The sky is blue" and "the grass is green" with the conjunction "and". The intrinsic compositionality of sentences makes them suitable for a tree representation, where the whole sentence (the root) is built in terms of sub-phrases (the internal nodes) which in turn are defined in terms of smaller constituents; the base cases are words (the leaves) since they are the atomic piece of information.

Developing Machine Learning (ML) models for such a rich data representation poses two main challenges: they should adapt to different structures, and they should process the atomic information along with the contextual information (e.g. the surrounding entities) given by the structure. If we consider sentences as structures, the model adaptivity ensures that it can process different sentences. On the other hand, the ability to process contextual information guarantees that the model distinguishes "the sky is blue and the grass is green" from "the sky is green and the grass is blue". While these two sentences comprise the same atomic constituents, their structures define different contexts. In the first sentence, the adjective "blue" is in the context of the noun "sky"; in the second sentence, the same adjective is in the context of the noun "grass".

The function which processes contextual information is usually referred to as aggregation function. The expressive power of such a function is strictly related to the model capacity to capture complex relationships among structure constituents. For example, the conjunction "and" between two sub-phrases expresses a non-contrasting relation between them. This relation can be thought of as a sort of summation of the sub-phrases semantics. Nevertheless, the conjunction "but" expresses a contrasting relation whose semantic is more articulated than a simple summation. The sentence "It is not my favourite film, but I like it" expresses a positive judgement even if the

first sub-phrase does not.

This motivates the need for expressive aggregation functions for learning in structured domains. In this thesis, we address this challenge by means of tensor theory.

Tensors are commonly considered a generalisation of arrays to the multi-dimensional case. Indeed, tensors are more than mere containers: they are powerful mathematical objects strictly related to multi-linear algebra. Nevertheless, they suffer the *curse of dimensionality*, i.e. the exponential relationship between the number of tensor entries and its order. Therefore, working with high-order tensors becomes prohibitive due to the computational and memory resources necessary to process and store such data. In this respect, tensor decompositions are fundamental to mitigate this effect since they factorise high-order tensors into a combination of simpler ones.

Recently, tensors have aroused interest in the ML community. While such interest was first restricted to multi-way data analysis [99, 1, 37], it later spread to ML models that handle tensor data [160, 137, 100]. Nevertheless, the last emerging (and the most exciting) trend in the intersection between these two fields is the use of tensor theory and multi-linear algebra as a backbone for ML. In this context, tensor decompositions with low-rank constraints have been successfully applied to reduce the number of parameters of neural [125, 23, 105, 163] and probabilistic models [126, 176] with negligible performances loss. Nevertheless, the usage of tensors as parameters can be useful to model higher-order interactions among inputs. While the possibility to model higher-order interactions with tensors has been discussed in the literature with the introduction of the High-Order Neural Network [65], less attention has been paid on the possibility to use tensor decompositions to reduce the complexity of such networks. Thus, modelling higher-order interactions is feasible only for a small number of inputs due to the exponential number of parameters required (e.g. [19, 171]).

So far, we have discussed how:

1. ML models for structures require complex aggregation functions to express complex relationships among structure constituents;

2. tensors can implement complex functions due to their intrinsic ability to model higher-order interactions; the price to pay for such expressiveness is an exponential number of tensor entries;

3. tensor decompositions provide a well-grounded mechanism to mitigate the tensors curse of dimensionality.

Thus, it seems natural to apply tensors and their decompositions to implement new aggregation functions while limiting their complexity. Nevertheless, the connection between ML models for structured data and tensor theory is still an unexplored research area.

The use of tensors to aggregate structure constituents has been proposed in seminal papers such as [116] and [58]. Nevertheless, they are used in practice only to model the interactions among a small number of constituents (e.g. in binary trees [153]) due to the exponential size of the tensor parameters.

When the number of constituents to aggregate increases, approximations are commonly used. In probabilistic models, the Switching Parent (SP) approximation [147] has been applied to reduce the complexity of tensor-based aggregation functions [12]. The effectiveness of the SP approximation relies on its simple probabilistic interpretation and the minimal effort required to derive its learning procedure [147]. In neural models, the common approach is to consider only first-order interactions among structure constituents. The use of first-order neural models is motivated by their theoretic expressive power. It can be shown that first-order neural models for structured data are universal approximators [154, 79, 78] and they can simulate computational models such as Finite State Automata [103, 104] and Turing Machines [151].

Both existing approximations rely on simple aggregation functions based on summations which are not related to tensor decompositions. Thus, given their theoretic expressive power, it is natural to ask if models that leverage complex aggregation functions are useful at all. In this respect, the authors in [116] present an experimental comparison between second-order and first-order models on the grammatical inference task: the results show that, in practice, second-order models outperform first-order architectures, arguing that the former architectures are better suited for representing finite-state grammars. Hence, even if we have guarantees that the desired solution is in the hypothesis space of first-order models, we do not have any guarantee that the learning algorithm will find such a solution. This suggests that there are still fundamental research questions concerning the practical inductive bias introduced by the existing models.

1.2 Objectives and Contributions

The main objective of this thesis is to deepen the relationship between structured data processing and tensors theory. To this end, we first aim to define a tensor framework which relates model expressiveness with a tensor parametrisation of the aggregation function. Then, we intend to use the proposed framework to define a novel class of ML models for structured data which leverage tensor decompositions to capture complex interactions while limiting the model complexity. A key point of our study is the analysis of the inductive bias introduced by these models.

A second objective is to develop models which can automatically adapt their complexity to data, avoiding the costly model selection phase to select the right hyper-parameters values. In this respect, we use a Bayesian approach.

In the following, we discuss the main contributions in detail.

A Tensor Framework for Learning in Structured Domains

We introduce a framework for learning in structured domains grounded on tensor theory. This framework is based on the observation that the existing models in the literature can be defined by imposing specific constraints on a tensor parametrisation.

From this perspective, we argue that models parametrised by a tensor have a low inductive bias since they do not add such constraints. We instantiate the proposed framework defining a probabilistic and a neural model whose aggregation functions are parametrised by a tensor, highlighting that both models introduce a low inductive bias. Nevertheless, the price to pay for such expressiveness is an exponential relation between the number of parameters and the hidden encoding space. Also, we show how a probabilistic and a neural models commonly used in the literature can be obtained by imposing a specific constraint on the tensor parameter, reducing the number of parameters required. The advantage of such a formulation is twofold. On the one hand, it paves the way to applying tensor decompositions to approximate tensor models (and thus reducing the number of parameters required). On the other hand, it allows interpreting such approximations as the introduction of a specific inductive bias due to the constraints imposed to the tensor parameter.

New Models for Structured Data by means of Tensor Decompositions

We develop nine different models for structured domains combining three different tensor decompositions with three common classes of ML models: probabilistic, neural and LSTM-based. It is worth highlighting that LSTM-based models are neural models with a specific architecture. We treat them as a separate class due to the relevance of the LSTM architecture in the context of structured data learning. In all the proposed models, we break the exponential relation between the number of parameters and the hidden encoding space thanks to a new hyper-parameter: the decomposition rank. While the size of the hidden encoding space reflects the input structures complexity (in terms of constituents co-occurrences), the value of the rank reflects the model expressiveness (in terms of the ability to capture complex interactions among constituents). Moreover, we show that each tensor decomposition introduces a specific inductive bias into the model. The experimental analysis conducted on two different tasks shows that the proposed models outperform the existing ones. In light of the results obtained, we discuss the advantages of tensor decomposition in terms of inductive bias.

Tensor-based models for unbounded structured data

We combine tensor decompositions with weight sharing constraints to develop a set of models which handle unbounded structures, i.e. structures where we cannot determine an upper bound on the context size. Also in this case, we analyse the inductive bias they introduce. Interestingly, a connection with the canonical decomposition of symmetric tensors arises, leading to the definition of permutational invariant aggregation functions. We experimentally assess the effectiveness of the proposed models on different natural language processing tasks. The results show that neural models which leverage tensor decompositions outperform existing models in semantic similarity tasks.

Unbounded models for structured data

We introduce two probabilistic unbounded models for structured data. In this case, the term unbounded refers to the model complexity that is not a priori determined by fixing some hyper-parameters. Thus, it is (theoretically) unbounded. The first model introduced is a Bayesian non-parametric mixture model to tackle unsupervised tasks on structured data. Thanks to the Bayesian framework, the number of clusters is learned from the data. The second model introduced is a Bayesian extension of a model based on a tensor decomposition. In this case, the Bayesian framework allows estimating the decomposition rank directly from the input data. The experimental results show the validity of the Bayesian approach also in the context of structured data.

1.3 Outline of the Thesis

The thesis is organised in seven chapters.

In Chapter 1, we detail the motivations underlying our work. Then, we summarise the main objectives and the major contributions of the thesis.

In Chapter 2, we introduce the background topics relevant for developing the thesis. In particular, we discuss general concepts of ML as well as a general framework for learning in structured domains. Moreover, we define the concept of tensors, summarising the most relevant operations on them. Particular emphasis is given to tensor decompositions. Finally, we provide a taxonomy of the model introduced in this thesis to facilitate the reading.

In Chapter 3, we present a tensor framework for learning with structured data. To this end, we define a probabilistic and a neural model which rely on a tensor parametrisation of the aggregation function. Moreover, we show how existing approximations can be cast in the proposed framework.

In Chapter 4, we introduce novel probabilistic and neural models for structured data leveraging tensor decompositions. In particular, we show how tensor decompositions can be used to reduce the complexity of full-tensor models by introducing a specific inductive bias. We also provide an experimental evaluation of the proposed models.

In Chapter 5, we define a new set of models that handle unbounded structures by combining tensor decompositions with weight sharing constraints. We experimentally assess the performances of the proposed models on natural language processing tasks.

In Chapter 6, we introduce two probabilistic models which adapt their complexity directly to data. The former is a Bayesian Non-Parametric mixture model for learning clusters in structured data. The latter is a Bayesian tensor model that adjusts its aggregation function complexity during the learning procedure. Both models are experimentally evaluated.

In Chapter 7, we draw our conclusion summarising the most relevant results of our work, discussing the main limiting factors as well as possible future research directions.

1.4 Origin of the Chapters

Most of the research studies described in this thesis have been published in conference proceedings, journal papers or are currently under review. In particular:

- the tensor framework for learning in structured domains has been proposed in [25], submitted for journal publication;
- the tensor based models defined in Chapter 4 have been introduced proposed in [26, 27, 29] and in the submitted journal paper [25];
- the tensor models for unbounded structures and their application on natural language processing tasks presented in Chapter 5 have been introduced proposed in [28];
- the mixture models and the Bayesian tensor model presented in Chapter 6 have been proposed in [9, 7] and [26], respectively.

Chapter 2

Background and Related Works

In this chapter, we review the main background topics used to develop our thesis. In Section 2.1, we provide a more detailed description of the chapter, motivating why we do not describe other research directions related to our work. In Section 2.2, we introduce general machine learning concepts together with the two learning paradigms used in our work. In Section 2.3, we show how these learning paradigms can be extended to process structured data recursively. Finally, in Section 2.4, we introduce tensors and the most relevant operations on them. A particular emphasis is given to tensor decompositions.

2.1 Chapter Overview

This chapter comprises three sections, each of them dedicated to introducing a specific background topic. To facilitate the reading, we briefly resume here the concepts presented in each section highlighting why they are relevant for our work.

In Section 2.2, we present a brief introduction to machine learning. The section starts introducing the basic concepts of inductive learning. In inductive learning, the goal of a learning process is to find a solution to the given task relying only on the set of observed samples. Importantly, the solution found must be effective on new task instances (i.e. instances never observed before). The learning model ability to generalise from the observed data is the core of the inductive learning and it is strictly related to their prior beliefs. If such beliefs match the assumptions of the task, the learning models will be able to generalise over new task instances more easily. This concept is a crucial point of our thesis; in Chapter 4, we show how leveraging tensor decompositions we can inject different prior beliefs which can help learning models for structured data. In the last part of the section, we introduce two learning paradigms which are extensively used in the rest of the thesis: the probabilistic and the neural one. For each of them, we show how such prior assumptions can be encoded in their definitions as well as the learning procedures we use in this work.

In Section 2.3, we introduce a well-known general framework for the adaptive processing of structured data. This framework is based on the recursive processing of the structured data and it is used as the starting point of our thesis. We mainly focus our research on a probabilistic and a neural instantiation of this framework. It is

worth highlighting that a recursive processing of structured data also occurs in other approaches such as reservoir computing [169, 110, 59, 60] and convolution models [119]. Even if they are not directly discussed in this thesis, we believe that the application of the tensor framework proposed in these contexts is an interesting future work. The same holds for the interesting research field of learning models for graphs [10]. The intrinsic presence of cycles in graphs does not allow applying the general framework mentioned above. With no doubt, the application of tensor theory also in this area is an exciting future research direction. Finally, it is worth mentioning that kernel methods [62, 77] is another popular learning paradigm for structured. Nevertheless, they are not directly based on a recursive processing of the structure and therefore are not discussed in this thesis.

In Section 2.4, we introduce tensors and their decompositions since they are the backbone of our proposal. We also discuss their relevant applications in the machine learning field found in the literature.

Finally, in Section 2.5, we introduce a model taxonomy of the models that will be introduced in the next chapters.

2.2 Introduction on Machine Learning

Machine Learning (ML) is a multidisciplinary field aiming to develop models that can perform specific tasks without being explicitly programmed to do so.[1] Their effectiveness relies on their ability to learn from experience. In this sense, we can say that ML models are indeed explicitly programmed to learn from experience. According to Mitchell, the concept of learning can be defined as [117]:

> "A computer program A is said to learn from experience E with respect to some class of tasks T and performance measure J, if its performance at tasks in T, as measured by J, improves with experience E."

This definition introduces four key concepts of ML: the *task* T, the *experience* E, the *performance* P and the *learning model* A.

Commonly, the experience is modelled as a set of *independent identically distributed* (i.i.d.) samples $\hat{D} = \{s_1, \dots, s_N\}$. Each sample s_i is generated from an unknown distribution $\hat{P}(s)$, i.e. $s_i \sim \hat{P}(s)$. The distribution $\hat{P}(s)$ is usually referred to as *true data distribution*. In this thesis, we assume that this distribution is used to generate the experience as well as the new unobserved samples.

The characteristics of the sample s depend on the specific task T. For example, in a *supervised task*, each sample s is a pair (x, y); in an *unsupervised task*, it contains only one element (x). In this brief introduction, we focus on general aspects of the learning process which hold for all tasks. Thus, such a characterisation is not necessary.

[1] The definition "without being explicitly programmed" is often attributed to Arthur Samuel, who coined the term "machine learning" in 1959 [102].

Although, in the rest of the thesis, we focus on tasks on structured domains, i.e. tasks in which each sample contains structured data.

The performance measure J evaluates the goodness of a possible task solution h on a sample s. Commonly, it is defined as an error measure (i.e. lower is better); thus, $J(s, h)$ measures the error made by the solution h on the sample s.

The learning model A comprises a set of hypothesis H, which is referred to as *hypothesis space*, and a *learning procedure*. The hypothesis space contains all the possible solutions (i.e. hypothesis) h for the task T which can be implemented by A. The learning procedure is the core of the learning model; it aims to learn from the observed data $\hat{D}$ the best solution $\hat{h} \in H$ according to J. Note that the best solution $\hat{h}$ should minimise J over all possible task instances s, i.e.

$$\hat{h} = \arg\min_{h \in H} \int J(s, h)\, d\hat{P}(s), \tag{2.1}$$

where the value $\int J(s, h)\, d\hat{P}(s)$ is the expected error made by the hypothesis h with respect to the true distribution $\hat{P}(\cdot)$ and it is therefore referred to as the *true error*.

The true error cannot be used directly by the learning procedure to search for the best hypothesis in H since the true data distribution $\hat{P}(\cdot)$ is unknown. Everything we know about the true data distribution is the set of observed samples in $\hat{D}$. Thus, the learning procedure requires an inductive principle in order to generalise from these examples. The common *inductive learning principle* states that any hypothesis found to approximate the target function well over a sufficiently large set of training examples, will also approximate the target function well over other unobserved examples [117]. This inductive principle is justified by the assumption that both observed and unobserved data are drawn from the same distribution. Hence, the goal of the learning algorithm is to find the hypothesis which minimises the error on the observed data $\hat{D}$:

$$\tilde{h} = \arg\min_{h \in H} \sum_{i=1}^{N} J(s_i, h), \tag{2.2}$$

where the value $\sum_{i=1}^{N} J(s_i, h)$ is usually denoted by *empirical error*.

These concepts have been formalised by the *statistical learning theory* [166, 165], also providing an upper bound on the discrepancy between empirical error and true error. Such a bound depends on a quantity that grows as the learning model complexity grows but reduces as the number of training examples increases. While a detailed discussion on these topics is out of our scope (we refer the reader to [166, 165]), this result justifies the need of learning models which minimise the empirical error while limiting the model complexity. In this context, the model complexity is strictly related to its expressiveness; complex models are able to implement a wide range of possible solutions. On the other hand, too simple models can fail to minimise the empirical error since they can implement only a small range of solution.

The key to reach the trade-off between the limitation of the model complexity and the minimisation of the empirical error relies on the *inductive bias* of the learning

model. According to [118], the term *inductive bias* indicates any prior assumptions that the learning model can exploit to prefer a hypothesis over another. Note that the term prior underlines that such assumptions do not depend on the observed information but are a priori specified. It is worth highlighting that the goodness of these assumptions depends on the task T we are going to solve.

Commonly, a learning model defines the hypothesis space as a parametric space $H = \{h_\theta \mid \theta \in \mathcal{W}\}$, where θ are the hypothesis free parameters and $\mathcal{W}$ is the set of all possible parameter values. The quantity h_θ represents the parametric hypothesis implemented by the learning model. Thus, the inductive bias can be specified in two ways: (1) selecting the parametric hypothesis h_θ and (2) choosing a learning procedure which prefers a specific type of hypothesis [118]. In the former one, the inductive bias is used to explicitly limit the complexity of the model. For example, in probabilistic models, we can limit the hypothesis space by imposing independence assumptions among model variables (we discuss this aspect in Section 2.2.1). Conversely, in the non-parametric models that we introduce in Chapter 6, the model inductive bias is specified by a set of prior belief which penalise (or promote) a hypothesis during the training.

2.2.1 Bayesian Networks

Bayesian Networks (BNs) are a class of graphical models which allow depicting a joint probability distribution of random variables through a directed acyclic graph. To be more precise, they depict a factorisation of such a joint distribution asserting a set of conditional independence assumptions among variables. The graph structure allows to easily identify these independence assumptions.

Although BNs model interactions among all types of variables, for our purposes, it is convenient to assume they are all *discrete random variables*. A discrete random variable X is a random variable which has a finite number of outcomes that are also referred to as *states*. In the remainder of the thesis, we use uppercase letters to denote random variables and lowercase letter to denote outcomes. We usually use the same letter to denote random variables and their realisations, i.e. x is a realisation of the random variable X; $P(X = x)$ indicates the probability that x is the outcome of the random variable X. For the sake of simplicity, we denote this value as $P(x)$ since the random variable can be deduced from the lowercase letter that indicates the outcome. Similarly, we denote by $\sum_x P(x, y) = \sum_{x=1}^{M} P(X = x, Y = y)$ the marginalisation of a discrete random variable X with M states.

Definition 2.1 (Bayesian Network [17])**.** A BN is a directed acyclic graph where vertices represent random variables and edges represent dependencies among variables. Let $\{X_1, \ldots, X_D\}$ the set of random variables in the graph, the joint probability of these variables is factorised as:

$$P(x_1, \ldots, x_D) = \prod_{i=1}^{D} P(x_i \mid x_{\mathrm{pa}(X_i)}), \tag{2.3}$$

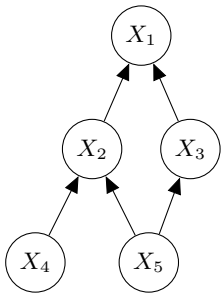


FIGURE 2.1: A Bayesian Network.

where $\text{pa}(X_i)$ represents the set of all variables which are linked to X_i in the graph.

In Figure 2.1, we show an example of BN which factorises the joint-probability of five variables $\{X_1, X_2, X_3, X_4, X_5\}$ as:

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1 \mid x_2, x_3)P(x_2 \mid x_4, x_5)P(x_3 \mid x_5)P(x_4)P(x_5). \tag{2.4}$$

The conditional distribution $P(x_1 \mid x_2, x_3)$ reflects that the variable X_1 has two incoming edges in the BN: one from X_2 and one from X_3. Similarly, the variable X_2 has two incoming edges from X_4 and one from X_5; hence, its state is determined by the conditional distribution $P(x_2 \mid x_4, x_5)$. The variable X_3 has only one incoming edge from X_5 and therefore is determined by $P(x_3 \mid x_5)$. The variables X_4 and X_5 have no incoming edges: hence, their states are determined by the distributions $P(x_4)$ and $P(x_5)$, respectively.

As we stated before, the key aspect of graphical models is their ability to depict conditional independence assumptions.

Definition 2.2 (Conditional independence [17])**.** Two sets of random variables $\mathcal{X}$ and $\mathcal{Y}$ are *conditionally independent* given another set of random variables $\mathcal{Z}$ (i.e. $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \mid \mathcal{Z}$) if it holds:

$$P(\mathcal{X}, \mathcal{Y} \mid \mathcal{Z}) = P(\mathcal{X} \mid \mathcal{Z})P(\mathcal{Y} \mid \mathcal{Z}). \tag{2.5}$$

If set $\mathcal{Z}$ is empty, the set $\mathcal{X}$ and $\mathcal{Y}$ are unconditionally independent and it is simply denoted by $\mathcal{X} \perp\!\!\!\perp \mathcal{Y}$.

For example, in the distribution in Eq. (2.4), it holds $X_4 \perp\!\!\!\perp X_5$ since:

$$\begin{aligned} P(x_4, x_5) &= \sum_{x_1}\sum_{x_2}\sum_{x_3} P(x_1, x_2, x_3, x_4, x_5) \\ &= \left(\sum_{x_1} P(x_1 \mid x_2, x_3)\right)\left(\sum_{x_2} P(x_2 \mid x_4, x_5)\right)\left(\sum_{x_3} P(x_3 \mid x_5)\right)P(x_4)P(x_5) \\ &= P(x_4)P(x_5) \end{aligned} \tag{2.6}$$

The same independence assumption can be deduced from the BN in Figure 2.1, since the variables X_4 and X_5 are *d-separated* in the BN. In general, for every disjoint

sets of variables $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ in a BN, it holds that $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} \mid \mathcal{Z}$ if $\mathcal{X}$ and $\mathcal{Y}$ are d-separated by $\mathcal{Z}$ in the BN.

Definition 2.3 (d-separation [17])**.** Let $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ be disjoint sets of vertices (i.e. variables); $\mathcal{X}$ and $\mathcal{Y}$ are d-separated by $\mathcal{Z}$ if and only if all undirected paths between some vertex in $\mathcal{X}$ and some vertex in $\mathcal{Y}$ are *blocked* by $\mathcal{Z}$. A path U is blocked by $\mathcal{Z}$ if:

- there is a collider in the path and neither the collider nor any of its descendants is in $\mathcal{Z}$;
- there is a node in the path that is not a collider and it is in $\mathcal{Z}$.

From the definition of d-separation in BNs, it is clear that colliders play a key role to determine the independence assumptions induced by a BN. A collider c in a path is a vertex such that both its neighbours in the path have directed edges to c. For example, in the BN in Figure 2.1, the node X_2 is a collider in the undirected path $X_4 - X_2 - X_5$. In fact, both neighbours of X_2 (i.e. X_4 and X_5) have edges directed to X_2.

As we have already shown mathematically in Eq. (2.6), the random variables X_4 and X_5 are independent. We can draw the same conclusion showing that these two variables are d-separated in the BN in Figure 2.1. Between X_4 and X_5 there are two undirected paths: $U_1 = X_4 - X_2 - X_5$ and $U_2 = X_4 - X_2 - X_1 - X_3 - X_5$. Nevertheless, both paths are blocked by colliders X_2 and X_1, respectively. Then, we can conclude that X_4 and X_5 are d-separated in this BN.

It is worth highlighting that while d-separations imply conditional independences, conditional independences do not imply d-separations. For example, variables X_4 and X_5 are not d-separated given X_2 in the BN in Figure 2.1. However, we can still define a distribution that is consistent with such a BN but also ensure $X_4 \perp\!\!\!\perp X_5 \mid X_2$. It is enough to define $P(x_2 \mid x_4, x_5) = \frac{P(x_2|x_4)P(x_2|x_5)}{P(x_2)}$, hence:

$$\begin{aligned} P(x_4, x_5 \mid x_2) &= \frac{P(x_2 \mid x_4, x_5)P(x_4)P(x_5)}{P(x_2)} \\ &= \frac{P(x_2 \mid x_4)P(x_2 \mid x_5)P(x_4)P(x_5)}{P(x_2)P(x_2)} = P(x_4 \mid x_2)P(x_5 \mid x_2), \end{aligned} \tag{2.7}$$

where the last equality holds applying the Bayes' theorem two times.

Unfortunately, we cannot create a BN which represent this specific conditional independence assumption. In this sense, BNs have a limited ability to graphically express conditional independence statements [17]. Nevertheless, it does not exist a graphical model which allows to express all possible conditional independence statements [17].

The parameters of a BN are the parameters of each conditional distribution $P(x_i \mid x_{\text{pa}(X_i)})$; since we assume that all variables are discrete, each conditional distribution is a categorical distribution. The number of parameters for each conditional distribution depends on how many variables it is defined. For example, assuming that X_i has

M states, the distribution $P(x_i)$ defines M probability values (one for each state of X_i) and therefore it is parametrised by a vector $\theta \in \mathbb{R}^M$. Clearly, θ can contain only positive values (i.e. $\theta[i] > 0$) and the sum on all possible outcomes of the variable must be equal to one (i.e. $\sum_{i=1}^{M} \theta[i] = 1$). These are usually referred to as *probability constraints*. For the sake of simplicity, we denote as $\theta \in \mathbb{R}_{\geq 0}^{M}$ a vector whose entries are non-negative. The sum-to-one constraint can always be satisfied normalising the positive vectors properly.

A conditional distribution with two variables $P(x_i \mid x_j)$ define a distribution over X_i for each state of X_j: hence, assuming that both variables have M states, it requires $M \times M$ parameters. We write $P(x_i \mid x_j, \theta)$ to indicate that $\theta \in \mathbb{R}_{\geq 0}^{M \times M}$ parametrises the conditional distribution. In particular, we assume that $P(X_i = i \mid X_j = j)$ is equal to $\theta[j, i]$. It is easy to show that if the probability distribution is defined over D variables with M states, it requires M^D parameters.

From this point of view, BNs allow reducing the number of parameters required to model joint probability distributions. For example, the joint distribution in the left-hand side of Eq. (2.4) requires M^5 parameters (assuming that all variables have M states). However, thanks to the independence assumptions introduced by the BN, we are able to model that joint distribution using $M^3 + M^3 + M^2 + M + M = O(M^3)$ parameters, i.e. the parameters of the distributions in the right-hand side of Eq. (2.4).

Hypothesis space. The hypothesis space defined by a BN is the set of probability distribution which are consistent with the set of independence assumptions it introduces. Hence, it is convenient to represent the hypothesis space in function of the parameters: $H = \left\{ \prod_{i=1}^{D} P(x_i \mid x_{\mathrm{pa}(X_i)}, \theta_i) \mid \forall i \in [1, D]. \, \theta_i \in \mathbb{R}_{\geq 0} \right\}$, where, for the sake of simplicity, we ignore the size of each parameter θ_i.

Learning. Thanks to the definition of the hypothesis space as a parametric set of distribution, the learning procedure aims to find the best parameters which minimise the error measure. In probabilistic models, the common error measure used is the negative-log likelihood $J(s, \theta) = -\log P(s \mid \theta)$. This allows rewriting the learning problem defined in Eq. (2.2) as:

$$\theta_{ML} = \arg\min_{\theta \in \mathbb{P}} \sum_{i=1}^{N} J(s_i, \theta) = \arg\min_{\theta \in \mathbb{P}} -\sum_{i=1}^{N} \log P(s_i \mid \theta) = \arg\max_{\theta \in \mathbb{P}} \prod_{i=1}^{N} P(s_i \mid \theta), \tag{2.8}$$

where the last equality holds for the monotonicity of the logarithm function. Hence, the best parameters θ_{ML} are the one which maximise the likelihood of the observed data.

Learning the maximum likelihood parameters of a BN is straightforward when all variables are *visible* (i.e. if each sample s contains the realisations of all the random variables defined by the BN). Nevertheless, if there are *hidden* variables (i.e. variables which realisations are not observed in s), the maximisation of the likelihood of the observed data becomes more complex. Due to the marginalisation of the hidden

variables, the visible variables can couple together. Thus, we lose the factorisation introduced by the BN [17].

In this thesis, we focus on one of the most used algorithms to learn maximum likelihood parameters of a BN in the presence of hidden variables: the Expectation Maximisation (EM) [45].

The basic idea of the EM is to maximise a lower-bound of the data log-likelihood rather than the data likelihood itself. Let assume that there are two disjoint set of variables in the BNs, i.e. $\{\mathcal{H}_i, \mathcal{V}_i\}$, and that sample s contains outcomes of only variables in $\mathcal{V}$ (i.e. $s = \{v_1, \dots v_{|\mathcal{V}|}\}$), the lower-bound $\tilde{L}$ maximised by the EM is defined as:

$$\tilde{L}(Q, \theta) = -\sum_{i=1}^{N} \mathbb{E}[\log Q(\mathcal{H}_i \mid \mathcal{V}_i)]_{Q(\mathcal{H}_i|\mathcal{V}_i)} + \sum_{i=1}^{N} \mathbb{E}[\log P(\mathcal{H}_i, \mathcal{V}_i \mid \theta)]_{Q(\mathcal{H}_i|\mathcal{V}_i)}, \tag{2.9}$$

where Q and θ are the parameter of the lower-bound. The first term of the bound is usually called the *entropy* term, while the second term is the *energy* term. The maximisation of $\tilde{L}$ is achieved iteratively, starting from an initial parametrisation θ^0. The t-th iteration comprises the following two steps:

E-step: for a fixed parametrisation θ^{t-1}, compute $Q^t = \arg\max_Q \tilde{L}(Q, \theta^{t-1})$. The best Q^t corresponds to the posterior $P(\mathcal{H}_i \mid \mathcal{V}_i, \theta^{t-1})$ of each sample s_i and it can be usually obtained applying an inference procedure which depends on the BN;

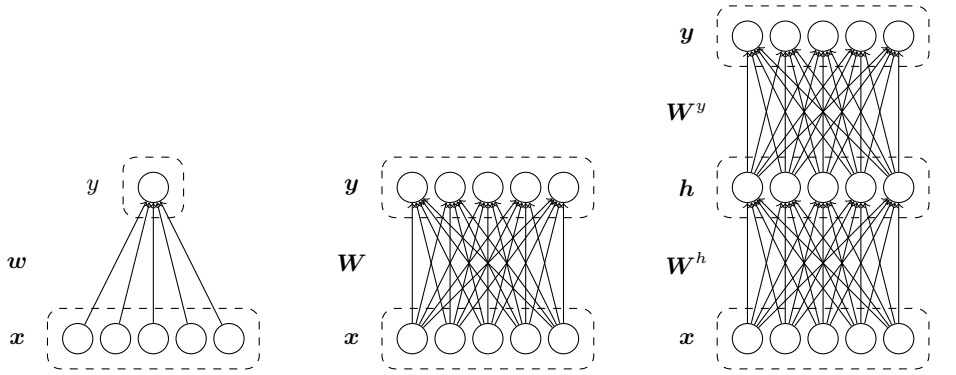
M-step: for a fixed distribution Q^t, compute $\theta^t = \arg\max_\theta \tilde{L}(Q^t, \theta)$. Since the only term in $\tilde{L}$ which depends on θ is the energy term, the best θ^t are the ones which maximise the *expected complete log-likelihood* $\sum_{i=1}^{N} \mathbb{E}[\log P(\mathcal{H}_i, \mathcal{V}_i \mid \theta)]_{Q(\mathcal{H}_i|\mathcal{V}_i)}$. In the case of BNs, this optimisation can be solved analytically.

Even if we have a guarantee that EM improves the data likelihood at each iteration, we do not have any guarantee that it will reach the global maximum [17]. The algorithm usually terminates when the likelihood of the data becomes constant (i.e. the improvement is less than a threshold) or a maximum number of iterations is reached.

2.2.2 Feed-Forward Neural Networks

Feed-forward Neural Networks (NNs) are powerful learning models in which the information is propagated from the input towards the output without feedback connections.

The computation unit of feed-forward NNs is the *neuron*. Let $\boldsymbol{x} \in \mathbb{R}^M$ be an input vector, the neuron output is a scalar which is computed as $y = \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b)$. The vector $\boldsymbol{w} \in \mathbb{R}^M$ and the scalar b are the neuron parameters, while the function $\sigma(\cdot)$ is referred to as the *activation function* of the neuron. Typical examples of activation functions are the *linear activation function* $\sigma(x) = x$ and the *sigmoid activation*



(A) A neuron. (B) A neuron layer. (C) A Multi-layer perceptron.

FIGURE 2.2: Examples of feed-forward neural networks. Biases are omitted.

function $\sigma(x) = 1/(a+e^{-x})$. We refer to [68] for others activation functions commonly used in NNs.

Neurons are usually grouped into *layers*. Let K the number of neurons in a layer, then the layer output is a vector $\boldsymbol{y} \in \mathbb{R}^K$ obtained as:

$$\boldsymbol{y} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{2.10}$$

where $\boldsymbol{W} \in \mathbb{R}^{K \times M}$ and $\boldsymbol{b} \in \mathbb{R}^K$ are the parameters of the layer obtained stacking together the parameter $\boldsymbol{w}_i$ and the bias b_i of each neuron $i \in [1, K]$ in the layer. The activation function σ is applied element-wise. We show in Figure 2.2b an example of neural layer.

If we consider the parametric function $\psi_\theta(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ (where $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$ are the function parameters), the output of a neuron layer is obtained by composing the activation function σ with ψ_θ, i.e. $\boldsymbol{y} = \sigma(\psi_\theta(\boldsymbol{x}))$. The function $\psi_\theta : \mathbb{R}^M \to \mathbb{R}^K$ defines an *affine transformation* of the input vector $\boldsymbol{x}$.

Definition 2.4 (Affine transformation)**.** A function $\psi : \mathbb{R}^M \to \mathbb{R}^K$ is an *affine transformation* if $\exists\, \phi : \mathbb{R}^M \to \mathbb{R}^K . \forall \boldsymbol{x} \in \mathbb{R}^M \wedge \boldsymbol{x}' \in \mathbb{R}^M .\ \psi(\boldsymbol{x}) - \psi(\boldsymbol{x}') = \phi(\boldsymbol{x} - \boldsymbol{x}')$ is linear.

In fact, $\psi_\theta(\boldsymbol{x}) - \psi_\theta(\boldsymbol{x}') = \boldsymbol{W}\boldsymbol{x} + b - (\boldsymbol{W}\boldsymbol{x}' + b) = \boldsymbol{W}(\boldsymbol{x} - \boldsymbol{x}')$ which is the linear function defined by the matrix $\boldsymbol{W}$. Moreover, it can be shown that every affine map can be obtained composing a linear transformation (i.e. a matrix) and a translation (i.e. vector addition); hence, the set $\{\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \mid \boldsymbol{W} \in \mathbb{R}^{K \times M}, \boldsymbol{b} \in \mathbb{R}^K\}$ contains all the affine maps between $\mathbb{R}^M$ and $\mathbb{R}^K$.

It is common to represent an affine map through a single matrix $\boldsymbol{W}' \in \mathbb{R}^{K \times (M+1)}$ which is obtained concatenating the bias vector $\boldsymbol{b}$ to the last column of $\boldsymbol{W}$, i.e. $\boldsymbol{W}' = [\boldsymbol{W} \mid \boldsymbol{b}]$. Such a matrix is called *augmented matrix* and it requires that also input vectors are augmented appending a 1 as last entry. The vector $\bar{\boldsymbol{x}} = [\boldsymbol{x}; 1]$ represents a *homogeneous coordinate* of $\boldsymbol{x}$. Hence, the equivalence $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} = \boldsymbol{W}'\bar{\boldsymbol{x}}$ holds.

More complex NN architectures can be realised stacking layers, i.e. using the output of i-th layer as input of the $(i+1)$-th layer. This architecture is usually referred to as Multi Layer Perceptron (MLP). The simplest MLP architecture contains only two layers (see Figure 2.2c) and its output is computed as:

$$\boldsymbol{y} = \phi(\boldsymbol{W}^y \boldsymbol{h} + \boldsymbol{b}^y), \quad \boldsymbol{h} = \sigma(\boldsymbol{W}^h \boldsymbol{x} + \boldsymbol{b}^h), \tag{2.11}$$

where $\boldsymbol{h} \in \mathbb{R}^C$ is the output of the first layer and it is referred to as the *hidden representation* of the input $\boldsymbol{x}$. For this reason, the first layer is denoted by *hidden layer*. Similarly, the last layer is denoted by *output layer*.

The parameters of the hidden layer are $\theta^h = \{\boldsymbol{W}^h \in \mathbb{R}^{C \times M}, \boldsymbol{b}^h \in \mathbb{R}^C\}$ while the parameters of the output layer are $\theta^y = \{\boldsymbol{W}^y \in \mathbb{R}^{K \times C}, \boldsymbol{b}^y \in \mathbb{R}^K\}$. Using augmented matrices, we can write more compactly $\boldsymbol{y} = \sigma_y(\bar{\boldsymbol{W}}^y \sigma_h(\bar{\boldsymbol{W}}^h \bar{\boldsymbol{x}}))$. For the sake of simplicity, we omit the homogeneous coordinate representation of the output $\sigma_h(\bar{\boldsymbol{W}}^h \bar{\boldsymbol{x}})$.

Clearly, more than two layers can be stacked in the same architecture. In general, layers in feed-forward NNs can be combined in more complex ways provided that no feedback connections are used. For example, skip-connections allows to connect a neuron in the i-th layer directly with a neuron in the $i+k$-th layer, with $k > 0$ [22]. Furthermore, a network can be sparse, with not all the possible connections within a layer being present [22]. Also, it is possible to use the same parameters in different points of the networks; this technique is usually referred to as *weight sharing* [22]. For example, the Convolutional Neural Network [106] is a common feed-forward NN architecture which extensively uses weight sharing and sparse connections.

The architecture of a network depicts the sequence of operations that should be performed on the input $\boldsymbol{x}$ to obtain the network output $\boldsymbol{y}$. Commonly, this is also defined as *computational graph* [68] and it plays also a key role in the learning procedure.

Hypothesis space. The hypothesis space of NNs can be defined as the set of functions which can be computed by its computational graph. For example, in the case of a two layer MLP, it can be defined as $H = \{g(\cdot, \bar{\boldsymbol{W}}_h, \bar{\boldsymbol{W}}_y) = \sigma_y(\bar{\boldsymbol{W}}_y \sigma_h(\bar{\boldsymbol{W}}_h \cdot)) \mid \bar{\boldsymbol{W}}_h \in \mathbb{R}^{C \times (M+1)}, \bar{\boldsymbol{W}}_y \in \mathbb{R}^{K \times (C+1)}\}$, where the dot $\cdot$ is a placeholder for the input vector. In general, let $g(\cdot, \theta_i, \ldots, \theta_D)$ be a computational graph of a NN with parameters $\{\theta_i, \ldots, \theta_D\}$; its hypothesis space can be defined as $H = \{g(\cdot, \theta_i, \ldots, \theta_D) \mid \forall i \in [1, D]. \theta_i \in \mathcal{W}\}$, where $\mathcal{W}$ is a suitable parameter space. Thus, we can modify the hypothesis of NN space by modifying its computational graph.

Nevertheless, it can be shown that a MLP with a linear activation function in the output layer and a sigmoid activation function in the hidden layer is able to represent any Borel measurable function from one finite-dimensional space to another with an arbitrary precision [87, 43]. This theorem (usually referred to as *universal approximation theorem*) assumes that the width of the network (i.e. the number of neuron in the hidden layer) is not fixed. Nevertheless, it has limited impact in practice

due to two main reasons. First, the hidden representation size is fixed indeed; the theorem does not provide any insight on how large this should be to represent the desired function. Second, even if we assume that the network is large enough to contain the desired function in the hypothesis space, we have no guarantee that the learning algorithm will find such a hypothesis [68].

Learning. In our thesis, we deal with NNs in the supervised setting. Let $(\boldsymbol{x}, \boldsymbol{y}^*)$ be a supervised sample and let $\boldsymbol{y} = g(\boldsymbol{x}, \theta)$ be the output of a NN with parameters θ, the empirical error defined in Eq. (2.2) can be rewritten as:

$$J(\theta) = \sum_{i=1}^{N} J\left(\boldsymbol{y}_i^*, g(\boldsymbol{x}_i, \theta)\right). \tag{2.12}$$

Most training algorithms involve an iterative procedure to minimise the empirical error. At each step, we can distinguish between two distinct stages. In the first stage, the gradients of the cost function with respect to the model parameters are evaluated. In the second stage, such gradients are used to update the parameters.

The Back-Propagation (BP) algorithm [144] is a general algorithm that allows computing gradients efficiently when a chain of operations is performed.. In the case of NNs, it is applied to compute the gradient of $J(\theta)$ with respect to θ. The core of the BP algorithm is the chain rule of calculus. Let $\boldsymbol{y} = \phi(\boldsymbol{x})$ and$cz = \psi(\boldsymbol{y})$, the chain rule states:

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\delta \boldsymbol{y}}{\delta \boldsymbol{x}}\right)^{\mathsf{T}} \nabla_{\boldsymbol{y}} z, \tag{2.13}$$

where $J_\phi = \left(\frac{\delta \boldsymbol{y}}{\delta \boldsymbol{x}}\right)$ is the Jacobian matrix of the function ϕ. The gradient $\nabla_{\boldsymbol{y}} z$ can be further decomposed as $J_\psi^{\mathsf{T}} \frac{\delta z}{\delta z} = J_\psi^{\mathsf{T}} 1$, where $J_\psi = \left(\frac{\delta z}{\delta \boldsymbol{y}}\right)$ is the Jacobian matrix of the function ψ. Hence, we obtain that $\nabla_{\boldsymbol{x}} z = J_\phi^{\mathsf{T}} J_\psi^{\mathsf{T}}$; the desired gradient can be computed just multiplying the transpose of the Jacobian matrix of the function used to compute the output z.

The BP essentially applies the chain rule recursively over the computational graph of a generic function f. In the case of NNs, it is applied to the computational graph of $J(\theta)$.

2.3 Learning with Structured Data

In this Section, we introduce some basics for learning with structured data. In Section, 2.3.1 we introduce the general framework for adaptive processing of structured data introduced in [58]. Then, we show instantiations of this framework on sequence and Directed Oriented Acyclic Graph domains in Section 2.3.2 and in 2.3.3 respectively.

2.3.1 General Framework for Processing Structured Data

The framework for adaptive processing of structured data adopted in this thesis has been introduced by Frasconi et al. to unify probabilistic and neural models for structured domains [58]. Before introducing the aforementioned framework, we formally define the concept of structured data.

Structured Data

A Directed Oriented Acyclic Graph (DOAG) is a directed acyclic graph where it exists a total order on the edges leaving from each vertex [58].

Definition 2.5 (DOAG [58])**.** A DOAG $\boldsymbol{\mathcal{D}}$ is a pair $(\text{vert}(\boldsymbol{\mathcal{D}}), \text{edg}(\boldsymbol{\mathcal{D}}))$, where $v \in \text{vert}(\boldsymbol{\mathcal{D}})$ is a *vertex* (or *node*) and $(v, u, l) \in \text{edg}(\boldsymbol{\mathcal{D}})$ is an *edge* from node v to node u in position l. Also, it holds:

$$(v, u, l) \in \text{edg}(\boldsymbol{\mathcal{D}}) \wedge (v, u', l) \in \text{edg}(\boldsymbol{\mathcal{D}}) \implies u = u'.$$

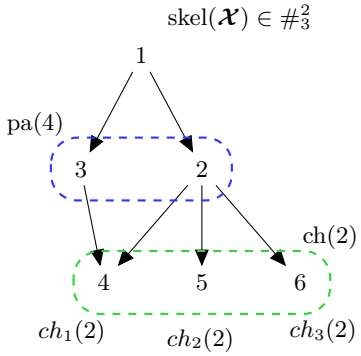
Let $\boldsymbol{\mathcal{D}}$ a DOAG and $v \in \text{vert}(\boldsymbol{\mathcal{D}})$ a node, we denote by $\text{ch}(v)$ the set of child nodes of v, by $\text{de}(v)$ the set of all descendants node of v and by $\text{pa}(v)$ the set of parent nodes of v. The cardinality of the set $\text{ch}(v)$ and $\text{pa}(v)$ are the *out-degree* and *in-degree* of v, respectively. The out-degree of $\boldsymbol{\mathcal{D}}$ is the maximum out-degree of all its nodes, i.e. the maximum number of children that a node in $\boldsymbol{\mathcal{D}}$ can have. Similarly, the in-degree of $\boldsymbol{\mathcal{D}}$ is the maximum in-degree of all its nodes, i.e. the maximum number of parents that a node in $\boldsymbol{\mathcal{D}}$ can have. A node $s \in \text{vert}(\boldsymbol{\mathcal{D}})$ is called *super-source* if every node $u \in \text{vert}(\boldsymbol{\mathcal{D}})$ can be reached by a directed path starting from s. On the contrary, a node which has no out-going edges is referred to as *sink*.

By definition, a DOAG imposes an order among child nodes of a given node v since all the edges leaving from v are ordered. Moreover, the position uniquely identify out-going edges. Hence, we can denote by $ch_l(v)$ (or more concisely vl) the child node of v in position l, i.e. the triplet $(v, ch_l(v), l) \in \text{edg}(\boldsymbol{\mathcal{D}})$. With no loss of generality, we assume $l \in [1, |\text{ch}(v)|]$ for each edge $(v, u, l) \in \text{edg}(\boldsymbol{\mathcal{D}})$.

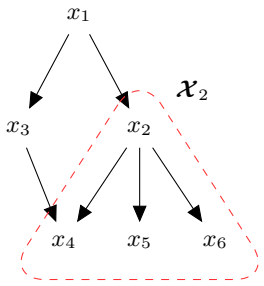
In Figure 2.3a, we show an example of DOAG. Node 1 is the super-source of $\boldsymbol{\mathcal{D}}$. Node 2 has three children: Node 4, Node 5 and Node 6, i.e. $\text{ch}(2) = \{4, 5, 6\}$. We always assume that children are ordered from left to right; hence, the left-most child is in the first position, i.e. $ch_1(2) = 4$. Similarly, $ch_2(2) = 5$ and $ch_3(2) = 6$.

We use the symbol $\#$ to denote the class of all DOAGs with a bounded (but unspecified) maximum in-degree and maximum out-degree. If such bounds are known, we denote by $\#_L^P$ the set of all DOAGs having maximum out-degree of L and a maximum in-degree of P. Also, we assume these sets contain only structures that possess a super-source or that are *empty*, i.e. without nodes. It is worth highlighting that we can always add a super-source node to a DOAG to satisfy this constraint [58].

In the context of learning, structures are used to store information. To this end, we assume that all structures are *labelled*, i.e. *labels* can be attached on edges and nodes.



(A) The skeleton of $\boldsymbol{\mathcal{X}}$.



(B) A labelled DOAG $\boldsymbol{\mathcal{X}}$.

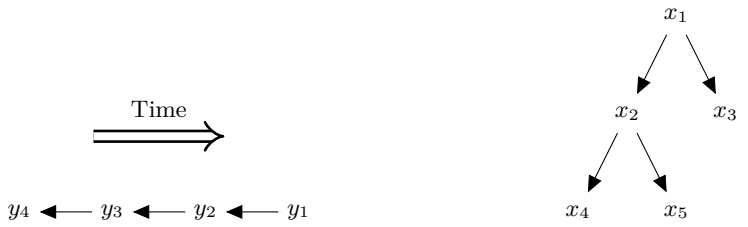
FIGURE 2.3: A labelled DOAG and its skeleton.

With no loss of generality, we assume that information are always stored in nodes [58]. The label associated with a node can be either numerical (it takes a continuous value, e.g. a scalar in $\mathbb{R}$ or a vector in $\mathbb{R}^n$) or categorical (i.e. its value is taken from a finite set). Let a node $v \in \text{vert}(\boldsymbol{\mathcal{D}})$, we denote by $x_v \in \mathcal{X}$ the label attached to v.

To simplify the notation introduced, we implicitly assume that the lowercase letter used to indicate a label indicates its domain, e.g. $x_v \in \mathcal{X}$ and $y_v \in \mathcal{Y}$. We extend this assumption also on the letter used to denote DOAGs, which indicates the domain of its labels. For example, we indicate by $\boldsymbol{\mathcal{X}}$ a labelled DOAG whose labels are taken from $\mathcal{X}$ and $\boldsymbol{\mathcal{Y}}$ a labelled DOAG whose labels are taken from $\mathcal{Y}$. We also denote by skel($\boldsymbol{\mathcal{X}}$) the *unlabelled* DOAG obtained ignoring all labels in $\boldsymbol{\mathcal{X}}$. We combine the notations used to indicate a structured space and a label domain in the symbol $\mathcal{X}^{\#_L^P}$, which denotes the set of all labelled DOAGs having maximum out-degree of L, maximum in-degree of P and whose labels are taken from $\mathcal{X}$. In Figure 2.3b, we show an example of labelled DOAG $\boldsymbol{\mathcal{D}} \in \mathcal{X}^{\#_3^2}$, hence all labels are taken from $\mathcal{X}$. Each node $v \in \text{vert}(\boldsymbol{\mathcal{D}})$ has the label $x_v \in \mathcal{X}$. Its skeleton skel($\boldsymbol{\mathcal{D}}$) is the DOAG depicted in Figure 2.3a. Moreover, we indicate by $\boldsymbol{\mathcal{X}}_v$ the sub-structure which contains only descendants nodes (with their labels) de(v) and v itself (see Figure 2.3b).

In the next paragraphs we introduce two particular type of DOAG class: *sequences* and *trees*.

Sequences. Sequences are DOAGs which belong to the class $\#_1^1$, i.e. DOAGs with a maximum out-degree and maximum-in degree of one. Sequences nodes are usually referred to as *elements*. Edges of a sequence define a total order among its elements; in temporal sequences, such a total order is used to associate a time-step for each element. In our definition, the temporal order is the opposite of the topological order of the structure (see Figure 2.4a). The super-source (the head of the chain) corresponds to the last time step in the sequence; thus, time indexes decrease following the direction of the arrows. In the rest of the work, we always use the topological order; to prevent confusion, we always use pa(v) and ch(v) and to indicate the predecessor and the



(A) A labelled sequence $\boldsymbol{\mathcal{Y}} \in \mathcal{Y}^{\#_1^1}$. (B) A labelled binary tree $\boldsymbol{\mathcal{X}} \in \mathcal{X}^{\#_2^1}$.

FIGURE 2.4: A labelled sequence and a labelled tree.

successor of an element v. Note that in the case of sequences, the child position is useless.

Trees. Trees are DOAGs which belong to the class $\#_L^1$, i.e. DOAGs with a maximum out-degree of L and a maximum in-degree of 1. The super-source tree node is referred to as *root*, while sink nodes are referred to as *leaves*. The maximum out-degree is also referred to as the *arity* of the tree. For example, in Figure 2.4b, we show a labelled binary tree (i.e. a tree with $L = 2$).

Structural Transductions

A *transduction* is a function between two structured domains; for example, $\tau : \mathcal{X}^{\#} \to \mathcal{Y}^{\#}$ denotes a transduction between two structured domains $\mathcal{X}^{\#}$ and $\mathcal{Y}^{\#}$ [58]. Hence, $\boldsymbol{\mathcal{Y}} = \tau(\boldsymbol{\mathcal{X}})$ is the structure obtained by applying the transduction $\tau(\cdot)$ to $\boldsymbol{\mathcal{X}}$.

A transduction $\tau(\cdot)$ is *isomorphic* if it generates an output structure with same skeleton as the input structure, i.e. $\text{skel}(\tau(\boldsymbol{\mathcal{X}})) = \text{skel}(\boldsymbol{\mathcal{X}})$ [58]. In other words, an isomorph transduction is a function which associates an output label for each input vertex. If such a function does not depend on the node on which it is applied (i.e. the same function is applied to each node), the transduction is *stationary*.

An isomorph transduction is *algebraic*, or *unstructured*, if the output label of each node $v \in \text{vert}(\boldsymbol{\mathcal{X}})$ depends only on the input label x_v [58]. In this case, the transduction can be computed applying a function to each input label; the input structure does not provide any further information necessary to compute the transduction. By contrast, in a non-algebraic isomorph transduction, the output label of each node v also depends on the *contextual* information provided from the structure.

Causal transductions are a particular case of non-algebraic isomorph transductions, in which the contextual information exploited to determine the output label of a node v belongs only to the descendant vertices of v [58].

In the context of learning, a key role is played by causal isomorph transductions which admit a *recursive state-representation*.

Definition 2.6 (Recursive transduction [58])**.** An isomorph stationary transduction $\tau : \mathcal{X}^{\#} \to \mathcal{Y}^{\#}$ admits a *recursive state-representation* if there exists a structured space

$\mathcal{H}^{\#}$ and two functions:

$$f : \mathcal{X} \times \mathcal{H} \times \cdots \times \mathcal{H} \to \mathcal{H}, \tag{2.14a}$$

$$g : \mathcal{H} \to \mathcal{Y}, \tag{2.14b}$$

such that, for all $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}} = \tau(\boldsymbol{\mathcal{X}})$, there is a $\boldsymbol{\mathcal{H}} \in \mathcal{H}^{\#}$ with $\text{skel}(\boldsymbol{\mathcal{X}}) = \text{skel}(\boldsymbol{\mathcal{Y}}) = \text{skel}(\boldsymbol{\mathcal{H}})$ and it holds:

$$h_v = f(x_v, h_{v1}, \ldots, h_{v|\text{ch}(v)|}), \tag{2.15a}$$

$$y_v = g(h_v), \tag{2.15b}$$

for each node $v \in \text{vert}(\boldsymbol{\mathcal{X}})$.

The function $f(\cdot)$ and $g(\cdot)$ are referred to as the *state-transition function* and the *output function*, respectively; the structure $\boldsymbol{\mathcal{H}}$ is referred to as the *hidden state-representation* of the transduction. Each label $h_v \in \mathcal{H}$ attached to a node $v \in \text{vert}(\boldsymbol{\mathcal{H}})$ is referred to as the *hidden state* of v.

The state-transition function and the output function state clearly the causal dependencies among input labels, hidden states and output labels assumed by the transduction. The function f indicates that the hidden state of a node depends on its input label and the hidden states of all its child nodes. The set $\{h_{v1}, \ldots, h_{v|ch(v)|}\}$ represents the contextual information which are exploited to determine the hidden state of the node v and it usually called *context*. The function g indicates that the output label of a node depends only on its hidden state.

The state-transition function allows computing hidden states recursively over the structure. The recursion scheme of $f(\cdot)$ is given by the input DOAG structure $\text{skel}(\boldsymbol{\mathcal{X}})$; however, while the DOAG defines a topological order which goes from the super-source to the sinks, the computation of the hidden states starts from the sinks and terminates at the super-source. The special hidden state $\perp$ is used to indicate the absence of a child node (necessary to define the recursion schema also on the sink nodes). The output function, instead, computes the output labels directly from the hidden states. Commonly, this recursive procedure can be represented graphically unfolding the recursive schema on the input structure. This representation is denoted by *encoding network* [58].

The concept of transduction can also be interpreted from a probabilistic point of view [58]. In this sense, a *probabilistic transduction* is a joint distribution $P(\boldsymbol{\mathcal{Y}} \mid \boldsymbol{\mathcal{X}}) \propto P(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}})$ defined over $\mathcal{X}^{\#} \times \mathcal{Y}^{\#}$, where both the input and the output labels are assumed to be realisations of different random variables. All the other properties introduced previously on deterministic transduction can be obtained by adding specific constraints on the joint distribution. If the probabilistic transduction admits a recursive state representation, also the hidden states are assumed to be realisations of random variables; the state-transition and the output function are usually referred to as *state-transition* and *output* distribution, respectively.

In some cases, it is useful to model a transduction which has a non-structural output (e.g. structure classification) [58]. Even if it is not an isomorph transduction, it can be easily implemented in the same framework by applying the output function only to the super-source node; the hidden states are always computed on the whole structure by the state-transition function. In some cases, the output function is applied to an aggregated representation of all hidden states rather than only on the super-source hidden state.

In the next paragraph, we show an example of recursive transduction.

Example of Recursive Transduction on Trees. Let us consider an isomorph transduction $eval : \mathcal{X}^{\#_2^1} \to \mathcal{Y}^{\#_2^1}$ between binary trees which evaluates mathematical expressions on integers. The input structure represents the parse tree of the expression; the input label domain is $\mathcal{X} = \mathbb{Z} \cup \{+, -, *, /\}$. The output of the transduction is a new structure with the same skeleton of the input one; the output labels represent all the intermediate results of the evaluation. Hence, the transduction $eval(\cdot)$ can be completely defined by the state-transition function ϕ:

$$h_v = \phi(x_v, h_{v1}, h_{v2}) = \begin{cases} x_v & \text{if} \quad h_{v1} = \bot \wedge h_{v2} = \bot \\ h_{v1} + h_{v2} & \text{if} \quad x_v = + \\ h_{v1} - h_{v2} & \text{if} \quad x_v = - \\ h_{v1} * h_{v2} & \text{if} \quad x_v = * \\ h_{v1} / h_{v2} & \text{if} \quad x_v = / \end{cases}, \tag{2.16}$$

and the output function ψ:

$$y_v = \psi(h_v) = h_v, \tag{2.17}$$

which is the identity function.

In Figure 2.5, we show the application of *eval* to the parse tree of the expression $4 + (2 * 3)$. We depict the interactions among the input labels, the output labels and the hidden states in the encoding network. In this case, the network has no particular meaning; it is simply a graphical support for the recursion schema.

The first step is the computation of the function ϕ applied to the sinks (the base case of the recursion): by the definition of ϕ in Eq. 2.16, the hidden state is equal to the input label on sink nodes. Then, also the sink output labels can be computed by applying the output function ψ on their hidden states.

In the second step, the function ϕ can be applied only to the node v which has the input label $*$. In fact, it is the only node such that all the hidden states of its children have been already computed; hence, by the definition of ϕ, $h_v = h_{v1} * h_{v2} = 2 * 3 = 6$. Applying the function ψ, its output label y_v is computed as well.

Finally, the hidden state of the super-source s can be computed, i.e. the node which has the input label $+$: by the definition of ϕ and ψ, $h_s = 4 + 6 = 10$ and $y_s = 10$. This terminates the recursion since all the hidden states and all the output labels are

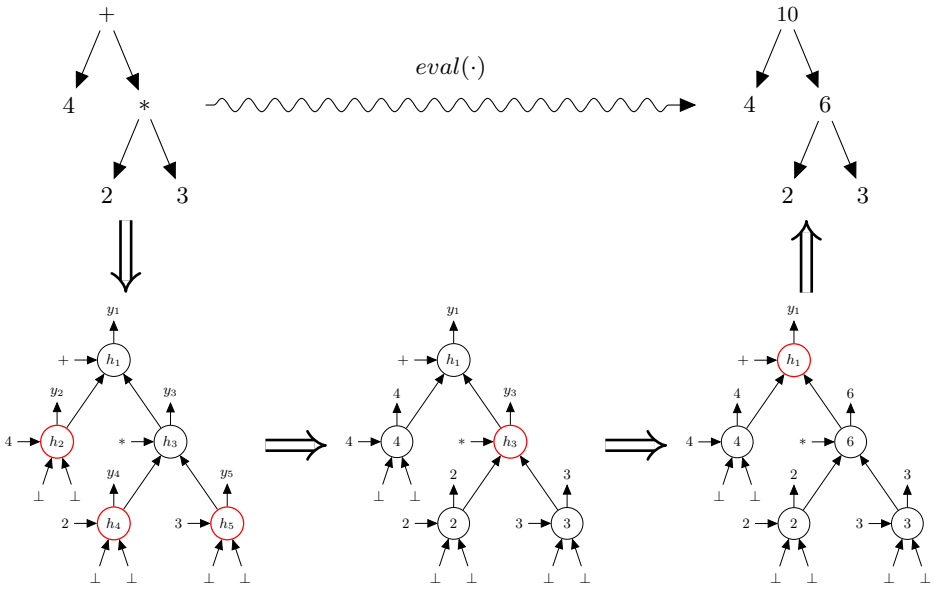


FIGURE 2.5: An isomorphic transduction which evaluates expression on integers.

computed. Hence, we can build the output structure extracting output labels from the encoding network.

We can define the equivalent super-source transduction of $eval(\cdot)$ as a transduction that computes only the final result of the expression (rather than a structure with all intermediate results). The hidden state are computed through the same recursion scheme depicted in Figure 2.5; nevertheless, the output function is applied on the super-source node, obtaining only the value 10.

Supervised Learning on Structured Domains

In our work, we mainly focus on supervised learning tasks on structured domain. As in the flat domain, we can define these tasks as the estimation of an unknown function $\tau : \mathcal{X}^{\#} \rightarrow \mathcal{Y}^{\#}$ from a set of input-output structure pairs $\hat{D} = \{(\boldsymbol{\mathcal{X}}^1, \boldsymbol{\mathcal{Y}}^1), \ldots, (\boldsymbol{\mathcal{X}}^N, \boldsymbol{\mathcal{Y}}^N)\}$. Clearly, the function $\tau(\cdot)$ is a transduction.

While learning general transductions is an interesting active research field (e.g. [158, 34, 54, 32]), in this thesis we focus on recursive transductions. Thanks to this constraint, we can reduce the learning of the whole transduction $\tau(\cdot)$ to learning of the state-transition function f and the output function g.

In the next two sections, we show instantiations of this framework on sequences and DOAGs which are particularly relevant for our work. In particular, we focus on probabilistic and neural instantiations where the functions f and g are implemented as probability distributions and neural networks, respectively.

Note that these two class of models also include other formalisms used in the context of structured data. For example, probabilistic automata are a particular case of the hidden Markov models that will be discussed in the next sections [52, 50].

Similarly, weighted automata [49] can be obtained removing the non-linearity in the definition of the neural networks that will be introduced in the next sections.

2.3.2 Recursive Models for Sequences

In this section, we describe examples of models which are able to learn recursive transductions on the sequence domain.

Hidden Markov Models

The Input-Output Hidden Markov Model (IO-HMM) [20] is a probabilistic model which is able to learn recursive transduction on sequences. To this end, it associates three random variables for each element v in the input sequence $\boldsymbol{\mathcal{X}}$: X_v is the random variable whose realisation is the input label x_v; Y_v is the random variable associated with the output label y_v; H_v is the random variable which models the hidden state h_v. The hidden random variables are discrete, while the type of the input and the output variables depends on their respective label domains.

The interactions among these variables is completely defined by the state-transition and the output (or emission) distribution. Since IO-HMM operates on sequences, the state-transition distribution $P(h_v \mid h_{\mathrm{ch}(v)})$ models the dependency between the hidden state h_v and its child hidden state $h_{\mathrm{ch}(v)}$. This assumption ensures that the hidden process is Markovian, i.e. its current state depends only on its previous (child) state. On the other hand, the output distribution $P(y_v \mid h_v)$ models the dependency between the output label y_v and the hidden state h_v. By applying these two distributions recursively over the input structure, we obtain the following complete data likelihood:

$$P(\boldsymbol{\mathcal{Y}}, \boldsymbol{\mathcal{H}} \mid \boldsymbol{\mathcal{X}}, \theta) = \prod_{v \in \mathrm{vert}(\boldsymbol{\mathcal{X}})} P(y_v \mid h_v, \theta_y) P(h_v \mid x_v, h_{\mathrm{ch}(v)}, \theta_h), \tag{2.18}$$

where $\boldsymbol{\mathcal{Y}}$ and $\boldsymbol{\mathcal{H}}$ are the set of all the output labels and the hidden states, respectively; $\theta = \{\theta_h, \theta_y\}$ are the IO-HMM parameters. This factorisation of the joint probability can be represented as a BN, which is the encoding network of the recursive model. Clearly, the Markovian property ensures that each hidden variable H_v block all paths between its ancestors and its descendants. Such independence assumptions reflect causalities assumed by recursive structural transduction. The likelihood of the output sequence can be obtained by marginalising all the hidden variables.

If the input structure is missing, IO-HMMs reduce to standard Hidden Markov Models (HMMs), making them suitable also for unsupervised tasks.

The parameters of IO-HMMs can be learned by applying a specialisation of the EM algorithm (see Section 2.2.1) [20]. This specialisation is obtained by a slight modification of the Baum-Welch algorithm [18] (i.e. the EM applied to HMMs). While the derivation of the M-step is straightforward, the computation of the posteriors in the E-step requires a recursive processing of the input sequence. Such an algorithm is usually referred to as *forward-backward* procedure [18] since it comprises two recursive

passes: a *forward* pass which goes from the sink element of the sequence to its super-source, and a *backward* pass which goes back from the super-source to the sink.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [53] represent a class of neural networks which are able to learn recursive transductions on sequences. The basic idea of RNNs is to implement the state-transition function and the output function by means of neural networks. The NN that implements the state-transition function is referred to as the *hidden* layer, since it computes the hidden states. Similarly, the NN that implements the output function is referred to as the *output* layer. RNNs assume that input labels, output labels and hidden states are vectors of possibly different size. Let v be an element of the input sequence $\boldsymbol{\mathcal{X}}$, then $\boldsymbol{x}_v \in \mathbb{R}^M$,$\boldsymbol{y}_v \in \mathbb{R}^K$ and $\boldsymbol{h}_v \in \mathbb{R}^C$ are its input label, output label and hidden state, respectively. Thus, the two neural layers are defined by the following equations:

$$\boldsymbol{h}_v = \sigma(\boldsymbol{W}^h \boldsymbol{x}_v + \boldsymbol{U}^h \boldsymbol{h}_{\mathrm{ch}(v)} + \boldsymbol{b}^h), \tag{2.19a}$$

$$\boldsymbol{y}_v = \phi(\boldsymbol{W}^y \boldsymbol{h}_v + \boldsymbol{b}^y), \tag{2.19b}$$

where $\theta_h = \{\boldsymbol{W}^h, \boldsymbol{U}^h, \boldsymbol{b}^h\}$ and $\theta_y = \{\boldsymbol{W}^y, \boldsymbol{b}^y\}$ are the parameters of the *hidden* and the *output* layer, respectively. The matrix $\boldsymbol{U}^h$ is usually referred to as the recurrent weight matrix since it is the matrix which is applied recursively to the hidden state. σ and ϕ are non-linear activation functions, e.g. the sigmoid function.

By unfolding these two layers on the input structure, we obtain a multi-layer feed-forward NN whose architecture matches the input structure and whose parameters are shared by each element. Such a feed-forward NN is the encoding network of the RNN and it reflects the causality assumptions induced by the recursive transduction.

RNNs are usually trained by minimising a loss through gradient descent. The computation of the loss gradients with respect to the model parameters is performed by the Back-Propagation Through Time (BPTT) algorithm [172]. The BPTT is essentially the application of the standard BP algorithm to the encoding network of RNNs. It is worth highlighting that a truncated version of BPTT is also used [115], where the gradient is not back-propagated for the whole sequence. Nevertheless, in this thesis we always use the standard BPTT algorithm.

Unfortunately, if the gradient propagated through many layers (in RNN, the number of layers depends on the number of elements in the input sequence), it tends to vanish or explode [85, 130]. In the next paragraph we show a very common solution to this problem which relies on the definition of a new neural state-transition function.

Long Short-Term Memory Networks. The Long Short-Term Memory (LSTM) network [85] is a particular type of RNN in which the state-transition function is implemented by a LSTM cell. The LSTM cell is defined by the following set of

equations:

$$\boldsymbol{i}_v = \sigma\left(\boldsymbol{W}^i\boldsymbol{x} + \boldsymbol{U}^i\boldsymbol{h}_{\mathrm{ch}(v)} + \boldsymbol{b}^i\right), \tag{2.20a}$$

$$\boldsymbol{o}_v = \sigma\left(\boldsymbol{W}^o\boldsymbol{x} + \boldsymbol{U}^o\boldsymbol{h}_{\mathrm{ch}(v)} + \boldsymbol{b}^o\right), \tag{2.20b}$$

$$\boldsymbol{u}_v = \sigma\left(\boldsymbol{W}^u\boldsymbol{x} + \boldsymbol{U}^u\boldsymbol{h}_{\mathrm{ch}(v)} + \boldsymbol{b}^u\right), \tag{2.20c}$$

$$\boldsymbol{f}_v = \sigma\left(\boldsymbol{W}^f\boldsymbol{x}_v + \boldsymbol{U}^f\boldsymbol{h}_{\mathrm{ch}(v)} + \boldsymbol{b}^f\right), \tag{2.20d}$$

$$\boldsymbol{c}_v = \boldsymbol{i}_v \odot \boldsymbol{u}_v + \boldsymbol{f}_v \odot \boldsymbol{c}_{\mathrm{ch}(v)}, \tag{2.20e}$$

$$\boldsymbol{h}_v = \boldsymbol{o}_v \odot \tanh(\boldsymbol{c}_v), \tag{2.20f}$$

where $\boldsymbol{i}_v \in \mathbb{R}^C$, $\boldsymbol{o}_v \in \mathbb{R}^C$, $\boldsymbol{f}_v \in \mathbb{R}^C$, $\boldsymbol{u}_v \in \mathbb{R}^C$ are referred to as the *input gate*, the *output gate*, the *forget gate*, the *update value*, respectively; $\boldsymbol{c} \in \mathbb{R}^C$ represents the *memory cell*. The non-linearity σ is usually the sigmoid function and the symbol $\odot$ denotes the element-wise multiplication.

The gates control the dynamic of the hidden state, allowing, for example, to ignore new elements in favour of previous ones. The idea of introducing gates to compute the hidden state of the current element has been firstly introduced in leaky units [121, 82]. In LSTMs, the gate values are computed by applying a single layer NN on the current input label $\boldsymbol{x}$ and on the hidden state of the child node $\boldsymbol{h}_{\mathrm{ch}(v)}$; hence, the gate values are adapted at each element of the sequence. By observing the Eq. (2.20), it is clear that each neural layer used to compute these values is equal to the RNN hidden layer defined in Eq. (2.19a). Since the LSTM cell contains four gates, it follows that the number of parameters required by a LSTM cell is four times the number of parameters required by a RNN.

LSTMs are usually trained by minimising a loss through gradient descent. The computation of loss gradients with respect to the model parameters can be computed using the BPTT algorithm. Also in the case of LSTM, truncated version of BPTT can be applied [72]. Nevertheless, we always use the standard BPTT algorithm.

2.3.3 Recursive Models for Highly-Structured Domains

In this section, we describe examples of models which are able to learn recursive transductions on the DOAG domain.

Hidden Recursive Model

The Hidden Recursive Model (HRM) [58] is a probabilistic model which can learn recursive transductions between two structured spaces $\mathcal{X}^{\#_L^P}$ and $\mathcal{Y}^{\#_L^P}$. As in IO-HMM, HRM defines a triplet of random variables (X_v, H_v, Y_v) for each node in the input structure. The interactions among these variables is completely defined by the state-transition distribution and the output distribution. By unfolding these distributions

on the input structure, we obtain the following complete likelihood:

$$P(\boldsymbol{\mathcal{Y}}, \boldsymbol{\mathcal{H}} \mid \boldsymbol{\mathcal{X}}, \theta) = \prod_{v \in \text{vert}(\boldsymbol{\mathcal{X}})} P(y_v \mid h_v, \theta_y) P(h_v \mid x_v, h_{v1}, \ldots, h_{vL}, \theta_h), \tag{2.21}$$

where again $\boldsymbol{\mathcal{Y}}$ and $\boldsymbol{\mathcal{H}}$ represent the set of all the output labels and the hidden states, respectively; $\theta = \{\theta_h, \theta_y\}$ are the HRM parameters. Note that since we are dealing with DOAGs which have a maximum out-degree of L, the state-transition function should consider the hidden state of L child nodes. As we will show more in details in Section 3.2.1, this leads to a state-transition distribution which requires a number of parameters which grows exponentially with respect to L.

Due to this limitation, HRM has been applied only to DOAGs with a limited maximum out-degree, e.g. sequences and binary trees [58]. It is worth highlighting that IO-HMM correspond to the application of HRM to the sequence domain.

The first practical approximation which overcomes this limitation is the Switching-Parent Bottom-Up Hidden Tree Markov Model (SP-BHTMM) [12]. SP-BHTMM approximates the state-transition distribution as a mixture of simpler distributions, requiring a number of parameters which grows linearly with respect to L. SP-BHTMM has been also extended to perform transductions [11]. We introduce this approximation in details in Section 3.3.1, with emphasis on the independence assumption introduced to reduce the number of model parameters.

The HRM learning procedure depends on the class of the input structure. If the input structure is not singly-connected (i.e. between two nodes in the structure there is at most one directed path), exact inferences cannot be computed on the encoding network [58]; the input structure must be "compiled" into a new structure, called junction-tree [17]. Note that we must apply this compilation step to each structure in the dataset every time it should be processed, making the learning algorithm computationally demanding. Moreover, in the case of densely connected DOAGs, even the inference on junction trees can be intractable.

On the other hand, if the input DOAG is singly connected, the HRM parameters can be learned by a specialisation of the EM algorithm. As in the case of IO-HMMs, the computation of the posteriors in the E-step requires a recursive processing of the input structure. In this case, this recursive procedure is usually referred to as *upward-downward* procedure [51, 12], since it requires an *upward* recursive pass which goes from the sink nodes to the super-source and a *downward* recursive pass which goes from the super-source to the sink nodes. Such a specialisation has been defined only for binary trees [58]. In the case of SP-BHTMMs, a tailored version of the EM algorithm has been introduced without any restriction on the maximum out-degree of the input tree structures [12].

Recursive Neural Networks

First-order Recursive Neural Networks (RecNNs) [58] are neural models which are able to learn recursive structural transductions between two structured spaces $\mathcal{X}^{\#_L^P}$

and $\mathcal{Y}^{\#_L^P}$. To this end, first-order RecNNs define two different single layer NNs to implement the state-transition and the output function of the recursive transduction. Let v be a vertex of the input structure $\boldsymbol{\mathcal{X}}$, $\boldsymbol{x}_v \in \mathbb{R}^M$ be its input label, $\boldsymbol{y}_v \in \mathbb{R}^K$ be its output label and $\boldsymbol{h}_v \in \mathbb{R}^C$ be its hidden state. Then, the two neural layers are defined by the following equations:

$$\boldsymbol{h}_v = \sigma\left(\boldsymbol{W}^h \boldsymbol{x}_v + \sum_{l=1}^{L} \boldsymbol{U}_l^h \boldsymbol{h}_{vl} + \boldsymbol{b}^h\right), \tag{2.22a}$$

$$\boldsymbol{y}_v = \phi(\boldsymbol{W}^y \boldsymbol{h}_v + \boldsymbol{b}^y), \tag{2.22b}$$

where $\theta_h = \{\boldsymbol{W}^h, \boldsymbol{U}_1^h, \ldots, \boldsymbol{U}_L^h, \boldsymbol{b}^h\}$ and $\theta_y = \{\boldsymbol{W}^y, \boldsymbol{b}^y\}$ are parameters of the *hidden* and the *output* layer respectively; σ and ϕ are the activation functions. The state-transition function requires L recurrent weight matrices, one for each child node. Hence, the number of parameters increases linearly with respect to the maximum out degree L. This is in contrast to HRMs parametrisation, which has an exponential relation with respect to L. As we will show in Section 3.3.2, this complexity reduction is obtained by implicitly imposing an independence assumption between child contributions.

Unfolding these two layers on the input structure, we obtain a multi-layer feed-forward NN whose architecture matches the input structure and whose parameters are shared by each node. Such a feed-forward NN is the encoding network of the RecNN. As feed-forward NNs, also RecNNs are usually trained by minimising a loss through gradient descent. The computation of the loss gradients with respect to the model parameters is performed by the Back-Propagataion Through Structure (BPTS) algorithm [67], i.e. the application of the BP algorithm to generic DOAGs.

The problem of vanishing and exploding gradient could also appear in RecNNs, but it is less frequent since usually the input structures do not have long dependencies. Nevertheless, the LSTM cell has been extended also for tree structure.

Tree-Structured LSTM. Tree-LSTM [159] are a specific type of RecNNs which implement the state-transition function through a LSTM cell. The cell definition is

modified as follows to consider the increased number of child nodes:

$$\boldsymbol{i}_v = \sigma\left(\boldsymbol{W}^i + \sum_{l=1}^{L} \boldsymbol{U}_l^i \boldsymbol{h}_{vl} + \boldsymbol{b}^i\right), \tag{2.23a}$$

$$\boldsymbol{o}_v = \sigma\left(\boldsymbol{W}^o + \sum_{l=1}^{L} \boldsymbol{U}_l^o \boldsymbol{h}_{vl} + \boldsymbol{b}^o\right), \tag{2.23b}$$

$$\boldsymbol{u}_v = \sigma\left(\boldsymbol{W}^u + \sum_{l=1}^{L} \boldsymbol{U}_l^u \boldsymbol{h}_{vl} + \boldsymbol{b}^u\right), \tag{2.23c}$$

$$\boldsymbol{f}_{vk} = \sigma\left(\boldsymbol{W}^f \boldsymbol{x}_v + \sum_{l=1}^{L} \boldsymbol{U}_{kl}^f \boldsymbol{h}_{vl} + \boldsymbol{b}_k^f\right) \quad \forall k \in [1, L], \tag{2.23d}$$

$$\boldsymbol{c}_v = \boldsymbol{i}_v \odot \boldsymbol{u}_v + \sum_{l=1}^{L} \boldsymbol{f}_{vl} \odot \boldsymbol{c}_{vl}, \tag{2.23e}$$

$$\boldsymbol{h}_v = \boldsymbol{o}_v \odot \tanh(\boldsymbol{c}_v), \tag{2.23f}$$

where $\boldsymbol{i}$, $\boldsymbol{o}$, $\boldsymbol{u}$ and $\boldsymbol{c}$ are the input gate, the output gate, the update value and the memory cell, respectively. Note that the Tree-LSTM cell introduces a different forget gate for each child node. Hence, the forget gate $\boldsymbol{f}_{vk}$ regulates the information flow between the node v and its k-th child node $\mathrm{ch}(v)_k$.

Since each gate is computed through a single layer NN equals to the first-order RecNN hidden layer (see Eq. (2.22a)), the number of parameters required by a Tree-LSTM is $L + 3$ times the number of parameters required by a first-order RecNN.

Tree-LSTMs are trained by gradient descent algorithms. The gradients of the cost function with respect to the parameters are computed by applying the BPTS algorithm.

2.4 Tensors

In this section we introduce tensors and the basic operations on them. A particular emphasis is placed on the tensor decompositions, which play a fundamental role in our thesis.

2.4.1 Definitions and Notations

Tensors are generally defined as multidimensional arrays.

Definition 2.7 (Tensor [99, 36, 90])**.** A D-way (or Dth order) tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ is a multi-dimensional array with D dimension (or way or mode), i.e. $\underline{\boldsymbol{T}}[i_1, \ldots, i_D]$ is an entry of $\underline{\boldsymbol{T}}$ such that for each $d \in [1, D]$ the index $i_d \in [1, I_d]$. I_d is the size associated with the d-th dimension.

From the definition, it is clear that tensors generalise the widespread concepts of vectors and matrices. In particular, a 1-way tensor is called *vector* and we denote it by a bold lowercase letter: $\boldsymbol{a} \in \mathbb{R}^N$ is a vector of size N. A 2-way tensor is called

matrix and we denote it by a bold uppercase letter: $\boldsymbol{U} \in \mathbb{R}^{N \times M}$ is a matrix of size $N \times M$. A tensor with more than 2 dimension is called *high-order tensor* and we denote it by an underlined bold uppercase letter, e.g. $\underline{\boldsymbol{T}}$. For the sake of simplicity, we use the word *tensor* to indicate a high-order tensor.

For example, let $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ be a three-way tensor (see Figure 2.6a); $\underline{\boldsymbol{T}}$ has three dimensions having size I_1, I_2 and I_3 respectively. We can access the entries of $\underline{\boldsymbol{T}}$ by specifying an index for each mode: $\underline{\boldsymbol{T}}[i_1, i_2, i_3]$ indicates the entry of $\underline{\boldsymbol{T}}$ at the i_1-th position on the first dimension, the i_2-th position on the second dimension and the i_3-th position on the third dimension.

A downside of the definition of tensors as multidimensional arrays is that it does not emphasise the relation between tensors and multi-linear functions, i.e. multi-variate functions that are linear in each argument [70]. It can be shown that every multi-linear function can be represented as a tensor [70]. Let $f : \mathbb{R}^{I_1} \times \cdots \times \mathbb{R}^{I_D} \to \mathbb{R}^K$ be a multi-linear function and let $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_D$ be a collection of vectors where each $a_d \in \mathbb{R}^{I_d}$; then, it exists a tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times \cdots \times I_D \times K}$ such that:

$$f(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_D) = \underline{\boldsymbol{T}}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_D) = \sum_{i_1=1}^{I_1} \cdots \sum_{i_D=1}^{I_D} \underline{\boldsymbol{T}}[i_1, \ldots, i_D, :] \boldsymbol{a}_1[i_1] \ldots \boldsymbol{a}_D[i_D], \quad (2.24)$$

where, for the sake of simplicity, we denote directly by $\underline{\boldsymbol{T}}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_D)$ the application of the multi-linear map induced by $\underline{\boldsymbol{T}}$ to the argument $(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_D)$. We always assume that the output dimension is the last one.

The duality between multi-linear functions and tensors is the natural generalisation of the best-known duality between linear functions and matrices. Nevertheless, such a duality has deeper justifications that lie on the tensor product definition and its universality as multi-linear map [70].

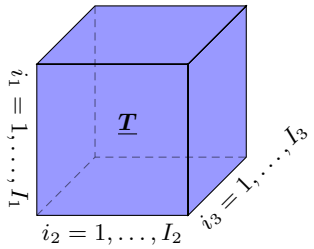
While we have shown that tensors are powerful objects, they suffer the so-called *curse of dimensionality* [36]. In this context, we use this expression to indicate the exponential relation between the number of entries, I^D, of a D-way tensor of size $I \times \cdots \times I$ and its order D [36]. Thus, working with high-order tensors becomes prohibitive due to the computational and memory resources necessary to process and to store such data.

For instance, let us consider a 12-order tensor $\underline{\boldsymbol{T}}$ which has size 8 in each dimension, i.e. $I_1 = \cdots = I_{12} = 8$. The space required to store $\underline{\boldsymbol{T}}$ is $4 \cdot 8^{12}$ bytes, where 8^{12} are the number of entries of $\underline{\boldsymbol{T}}$ and 4 is the number of bytes usually required to store floating point numbers. Hence, $\underline{\boldsymbol{T}}$ occupies 256 GB.

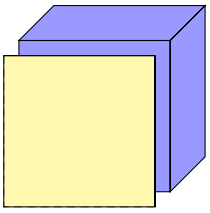
Before continuing with the introduction of tensor operations and tensor decompositions, it is useful to define particular types of tensors whose properties are used in the reminder of the thesis.

Definition 2.8 (Rank-one tensor [99])**.** A D-way tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ is *rank-one* if it can be obtained as the outer product of D vectors, i.e.:

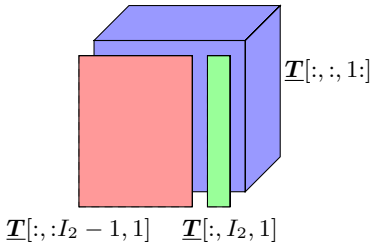
$$\underline{\boldsymbol{T}} = \boldsymbol{a}_1 \circ \boldsymbol{a}_2 \circ \cdots \circ \boldsymbol{a}_d, \quad (2.25)$$



(A) The tensor $\underline{\boldsymbol{T}}$.



(B) Slice of $\underline{\boldsymbol{T}}$.



(C) Sub-arrays of $\underline{\boldsymbol{T}}$.

FIGURE 2.6: A 3-way tensor $\underline{\boldsymbol{T}}$ and its sub-arrays.

where the symbol $\circ$ denotes the outer product operation. Hence, each entry of $\underline{\boldsymbol{T}}$ can be written as:

$$\underline{\boldsymbol{T}}[i_1, \ldots, i_d] = \boldsymbol{a}_1[i_1]\boldsymbol{a}_2[i_2] \ldots \boldsymbol{a}_d[i_d]. \tag{2.26}$$

Definition 2.9 (Symmetric tensor [99])**.** A D-way tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I \times \cdots \times I}$ is *symmetric* (or *super-symmetric*) if its entries remain constant under any permutation of the indices, i.e.:

$$\underline{\boldsymbol{T}}[i_1, \ldots, i_d] = \underline{\boldsymbol{T}}[\pi(i_1, \ldots, i_d)], \tag{2.27}$$

where π is a permutation of indices $i_1, \ldots, i_d$.

Definition 2.10 (Non-negative tensor)**.** A D-way tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ is *non-negative* if all its entries are positive or equal to zero, i.e.:

$$\underline{\boldsymbol{T}}[i_1, \ldots, i_d] \geq 0. \tag{2.28}$$

We denote a non-negative tensor as $\underline{\boldsymbol{T}} \in \mathbb{R}_{\geq 0}^{I_1 \times \cdots \times I_d}$.

2.4.2 Operations on Tensors

In the following paragraphs we introduce three basic operation on tensors: *indexing*, *reshaping* and *tensor contractions*.

Indexing

Indexing allow retrieving elements from a tensor. As we have already shown, we can access tensor elements specifying one index for each dimension, e.g. $\underline{\boldsymbol{T}}[i_1, i_2, i_3]$.

Moreover, we use indexing also to retrieve tensor sub-arrays, i.e. subsets of the tensor entries obtained fixing a subset of its indexes. We use the symbol ":" to indicate

that an index is unspecified. For example, the yellow matrix in Figure 2.6b represents the slice $\underline{\boldsymbol{T}}[:, 1, 1]$ while the green vector in Figure 2.6c represents the fiber $\underline{\boldsymbol{T}}[:, I_2, 1]$. We use the term *fiber* (*slice*) to indicate a sub-array that is a vector (matrix) [99].

With abuse of notation, we also allow to index a tensor by $i{:}j$, i.e. the list of consecutive index values ranging from i to j; when the starting value is not specified (i.e. $:j$), the range starts from 1; when the ending value is not specified (i.e. $i:$), the range ends at the dimension size. For example, in Figure 2.6c, we divide the tensor $\underline{\boldsymbol{T}}$ in two parts: the slice $A = \underline{\boldsymbol{T}}[:, :, 1]$ and the sub-tensor $\underline{\boldsymbol{T}}[:, :, 1{:}]$ obtained removing the slice A from $\underline{\boldsymbol{T}}$ (the blue cube in the figure); then, we split the slice A in two parts: its last column (i.e. the fiber $\underline{\boldsymbol{T}}[:, I_2, 1]$, the green vector in the figure) from the rest of columns (i.e. the matrix $\underline{\boldsymbol{T}}[:, :I_2 - 1, 1]$, the red matrix in the figure).

Reshaping

Matricisation and *vectorisation* are operations that transform tensors into matrices and vectors respectively. While these operations are very common in many scientific computation packages, their formal definition is clumsy since it lies on the mapping between lists of indices to a single index; we refer the reader to [36] for further details.

For our purposes, it is enough to understand the intuition of such operation. In Figure 2.7, we show two reshaping of the $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times 3 \times 4}$. In the bottom part of the figure, we reshape $\underline{\boldsymbol{T}}$ into a matrix with I_1 rows and 3×4 columns; such a matrix is obtained by concatenating all the mode-1 fibers of $\underline{\boldsymbol{T}}$. In general, we denote by $\underline{\boldsymbol{T}}_{(n)}$ the *n-mode matricisation* of $\underline{\boldsymbol{T}}$, i.e. the matrix with I_n rows and $\prod_{k \neq n} I_k$ columns obtained concatenating all the n-mode fibers of $\underline{\boldsymbol{T}}$. Also, in the right part of the figure, we transform $\underline{\boldsymbol{T}}$ into a vector with $I_1 \times 3 \times 4$ entries by simply stacking all its entries. We denote this reshaping operation by $\text{vec}(\underline{\boldsymbol{T}})$.

Tensor Contractions & Tensor Networks

The tensor contraction is a fundamental operation among tensors and can be considered as a generalisation of the matrix multiplication [36]. Let $\underline{\boldsymbol{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_R \times J_1 \cdots \times J_S}$ and $\underline{\boldsymbol{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_R \times K_1 \cdots \times K_T}$, the tensor contraction between $\underline{\boldsymbol{A}}$ and $\underline{\boldsymbol{B}}$ is a new $\underline{\boldsymbol{C}} \in \mathbb{R}^{J_1 \cdots \times J_S \times K_1 \cdots \times K_T}$ whose entries are obtained as:

$$\underline{\boldsymbol{C}}[j_1, \ldots, j_S, k_1, \ldots, k_T] = \sum_{i_1}^{I_1} \cdots \sum_{i_R}^{I_R} \underline{\boldsymbol{A}}[i_1, \ldots, i_R, j_1, \ldots, j_S] \underline{\boldsymbol{B}}[i_1, \ldots, i_R, k_1, \ldots, k_T]. \quad (2.29)$$

At first, the contraction operation multiplies entries of the two input tensors. Then, a summation is performed over "shared" dimensions which should have the same size.

For the sake of simplicity, we have assumed that the input tensors share the first R dimensions. The tensor contraction can be easily extended to the more general case in which shared dimensions can occupy any position; nevertheless, it requires

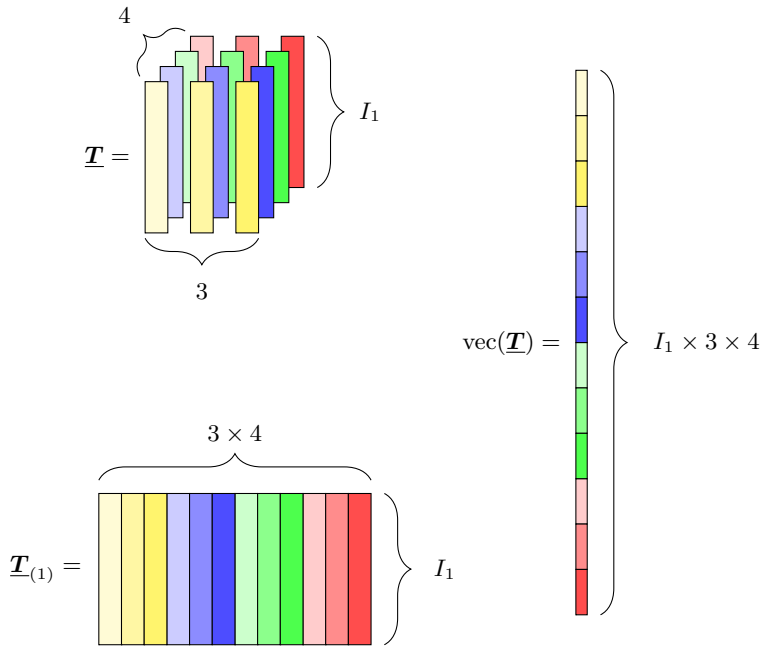


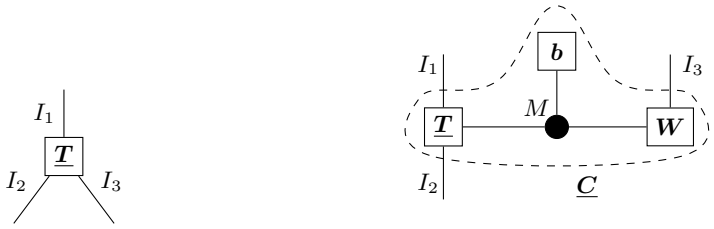
FIGURE 2.7: Different reshaping of $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times 3 \times 4}$: $\underline{\boldsymbol{T}}_{(1)}$ is its 1-mode matricisation while $\text{vec}(\underline{\boldsymbol{T}})$ is its vectorisation.

a cumbersome notation to identify such a mapping. To this end, it is convenient to represent tensors and contraction operations as *tensor networks* [36].

A tensor network is a diagrammatic representation which has been used in various context and therefore has been defined in multiple different ways. In our setting, it is convenient to define a tensor network as an undirected *hyper-graph* [6], i.e. a graph in which edges (called *hyper-edges*) can connect more than two nodes. Each vertex represents a tensor while each hyper-edge represents a mode. If a hyper-edge connects only one vertex, it is referred to as *dangling edge*. If a hyper-edge e connects more than one vertex, the mode represented by e is "shared" among the vertices (i.e. tensors) connected by e. Also, a label is attached to each hyper-edge to indicate the size of the mode. The underlying assumption of the tensor networks is that a contraction is always performed on all the shared hyper-edges. We clarify all these concepts by two examples.

In Figure 2.8a, we show the tensor network of a tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$: the hyper-graph contains only one vertex and three hyper-edges. The vertex is represented as a square box which contains the name of the tensor. Hyper-edges are lines and, in this case, are connected only to $\underline{\boldsymbol{T}}$; hence, they are dangling edges. We can deduce the order of a tensor counting the number of edges connected to its vertex. In particular, the dangling edge with the label I_1 indicate the first mode of $\underline{\boldsymbol{T}}$, the dangling edge with the label I_2 indicate the second mode of $\underline{\boldsymbol{T}}$ and the dangling edge with the label I_3 indicate the third mode of $\underline{\boldsymbol{T}}$. In this case, since there are no shared hyper-edges, no contraction is performed.

Tensor networks become interesting when there is a hyper-edge which connects more than one tensor. In Figure 2.8b, we show an example with three vertices: the



(A) $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ (B) $\underline{\boldsymbol{C}} = \sum_{m=1}^{M} \underline{\boldsymbol{T}}[:,:,m]\boldsymbol{W}[:,m]\boldsymbol{b}[m]$

FIGURE 2.8: Examples of tensor networks.

3-way tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times I_2 \times M}$, the matrix $\boldsymbol{W} \times \mathbb{R}^{I_3 \times M}$ and the vector $\boldsymbol{b} \in \mathbb{R}^{M}$. Moreover, there are three dangling edges (labelled with I_1, I_2 and I_3) and a hyper-edge which connects all three vertices. We graphically represent a hyper-edge as a black dot which is connected by a line to all the vertices in the hyper-edge; hence, we can retrieve the size of the hyper-edge (i.e. the number of vertices connected by it) by simply counting the number of lines attached to the black dot. The label near the black dot is the label attached to the hyper-edge. Hence, we have a hyper-edge of size 3 which connects $\underline{\boldsymbol{T}}$, $\boldsymbol{W}$ and $\boldsymbol{b}$. Such a connection implies that these three vectors share the dimension M. Therefore, the tensor network represents a tensor $\underline{\boldsymbol{C}}$ obtained contracting $\underline{\boldsymbol{T}}$, $\boldsymbol{W}$ and $\boldsymbol{b}$ along the mode of size M:

$$\underline{\boldsymbol{C}}[i_1, i_2, i_3] = \sum_{m=1}^{M} \underline{\boldsymbol{T}}[i_1, i_2, m]\boldsymbol{W}[i_3, m]\boldsymbol{b}[m]. \tag{2.30}$$

Note that the dangling edges of the tensor networks represents the mode of the tensor obtained after the contraction.

2.4.3 Tensor Decompositions

Tensor decompositions factorise tensors into a combination of simpler objects, referred to as *factors*. Again, we can think of tensor decompositions as the generalisation of matrix factorisations to the multi-dimensional case.

Tensor decompositions can be used to approximate tensors. As in the matrix case, the approximation is usually referred to as *low-rank approximation* since it is obtained constraining the decomposition rank. Each tensor decomposition define its own generalisation of the concept of rank; thus, the same tensor has different ranks according to different decompositions.

In the reminder of this section, we introduce three different tensor decompositions that are fundamental for the developing of the thesis: the *Canonical decomposition* [99], the *Higher-Order Singular Value Decomposition* [44] and the *Tensor Train decomposition* [129]. Nevertheless, it is worth highlighting that there are other tensor decompositions which play key roles in multiple contexts, e.g. the Tensor Ring decomposition [181], the Hierarchical Tucker decomposition [73, 71], Projected Entangled-Pair

States (PEPS) [168, 122]. We refer the reader to [36] for a comprehensive discussion of the tensor decompositions not considered in this thesis.

Canonical Decomposition

The canonical decomposition has been introduced multiple times in the literature. In 1927, Hitchcock was the first to propose the idea of the polyadic form of a tensor, i.e. expressing a tensor as the sum of a finite number of rank-one tensors [84, 83]. In 1944, a similar idea was proposed by Cattell in the context of psychological analysis [30, 31]. Nevertheless, the decomposition starts becoming popular in 1970 in the form of CANDECOMP [24] and PARAFAC [81]. Later, Kiers propose to standardise the name of the decomposition as Candecomp/Parafrac (CP) [93].

Definition 2.11 (CP approximation [99])**.** The CP decomposition approximates a D-way tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ into a sum of R rank-one tensors :

$$\underline{\boldsymbol{T}} \approx \sum_{r=1}^{R} \underline{\boldsymbol{X}}_r = \sum_{r=1}^{R} \boldsymbol{x}_{r1} \circ \boldsymbol{x}_{r2} \circ \cdots \circ \boldsymbol{x}_{rD}. \tag{2.31}$$

The last equality holds by the definition of rank-one tensor (see Definition 2.8). The value of R represents the rank of the canonical approximation.

If we collate together all the vectors which operates on the same dimension (e.g. the set $\boldsymbol{X}_1 = \{\boldsymbol{x}_{11}, \ldots, \boldsymbol{x}_{R1}\}$ on the first dimension), we can define the *factor matrices* $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_D$ of the decomposition. Hence, we can approximate entries of $\underline{\boldsymbol{T}}$ as:

$$\underline{\boldsymbol{T}}[i_1, \ldots, i_D] \approx \sum_{r=1}^{R} \boldsymbol{X}_1[i_1, r] \ldots \boldsymbol{X}_d[i_D, r], \tag{2.32}$$

where the k-th factor matrix $\boldsymbol{X}_k$ has shape $I_k \times R$.

The canonical approximation can be represented as a tensor network in which there is a vertex for each factor matrix and a hyper-edge that connects all the vertices along the mode of size R; all the other modes $I_1, \ldots, I_D$ are represented by dangling edges. For example, in Figure 2.9, we represent the canonical approximation of a 3-way tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.

CP-Rank. The CP-rank of a tensor $\underline{\boldsymbol{T}}$ is defined as the smallest number of rank-one tensors that generate $\underline{\boldsymbol{T}}$ as their sum, i.e. is the smallest value of R which makes the Eq. (2.32) an exact equality [99]. Hence, the CP-rank is the analogue of the matrix rank in high-order tensors. Nevertheless, they exhibit different properties. The first difference is that the rank of a real-valued tensors may actually be different over $\mathbb{R}$ and $\mathbb{C}$. The relation between real and complex rank is used to give an upper bound of the real CP rank, which is $\min_{i \in [1,D]} O(\prod_{k \neq i} I_k)$ in the worst case [16, 15]. Another major difference is that the computation of the CP-rank of a general tensor is a NP-Hard problem [99]. Hence, the exact CP decomposition can be obtained only attempting a

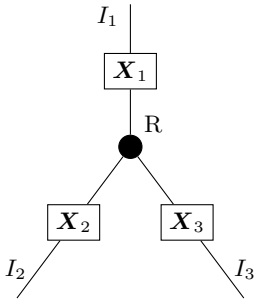


FIGURE 2.9: Tensor network of the CP approximation of a 3-way tensor.

perfect fit with values of $R \in \{1, 2, \dots\}$. Nevertheless, there is no perfect procedure for fitting a CP decomposition for a fixed rank value [99]; the most used one is the Alternating Least Square [36, 99].

Space required. The CP approximation with rank R of a tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ requires $O(DRI)$ space, where $I = \max_{k \in [1,D]} I_k$.

Higher-Order Singular Valued Decomposition

The High-Order Singular Value Decomposition (HOSVD) was first introduced by Tucker in [164] to search relations in a three-way tensor of psychometric data [164]. Later, the Tucker decomposition has been generalised to n-way tensors by [44] with the name HOSVD [44].

Definition 2.12 (HOSVD approximation [44])**.** The HOSVD decomposition approximates a D-way tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ as:

$$\underline{\boldsymbol{T}}[i_1, \dots, i_D] \approx \sum_{r_1}^{R_1} \cdots \sum_{r_D}^{R_D} \underline{\boldsymbol{G}}[r_1, \dots, r_D] X_1[i_1, r_1] \dots X_d[i_D, r_D], \tag{2.33}$$

where $\underline{\boldsymbol{G}} \in \mathbb{R}^{R_1 \times \cdots \times R_D}$ is a D-way tensor called *core tensor* and $X_1 \in \mathbb{R}^{I_1 \times R_1}, \dots X_D \in \mathbb{R}^{I_D \times R_D}$ are the *mode matrices* associated to each tensor dimension. The values $R_1, \dots, R_D$ are the ranks of the approximation.

In the original definition by De Lathauwer et al., orthogonality and unitary constraints are imposed on the core tensor and factor matrices, respectively. In our definition, we have removed such constraints because they are not necessary for the progress of our work. Nevertheless, we prefer to use the name HOSVD to emphasise the interpretation of this decomposition.

The HOSVD approximation can be represented as a tensor network in which there is a vertex for each mode matrix and a vertex for the core tensor. Then, the k-th mode matrix is linked to the core tensor along the shared dimension R_k; all the other modes $I_1, \dots, I_D$ are represented by dangling edges. For example, in Figure 2.10 we represent the HOSVD approximation of a 3-way tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times I_1 \times I_3}$.

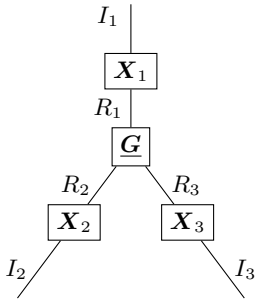


FIGURE 2.10: Tensor network of the HOSVD approximation of a 3-way tensor.

The HOSVD tensor network and the CP tensor network are similar: in the former, all mode matrices are contracted together with the core tensor while in the latter they are contracted together along the rank dimension. It can be shown that the CP approximation is equivalent to the HOSVD approximation with a diagonal core tensor [36].

n-ranks. The n-ranks of tensor $\underline{\boldsymbol{T}}$ are the smallest values of $R_1, \ldots, R_D$ that make the Eq. (2.33) an exact equality. The value of the rank along the n-th dimension can be computed exactly since it corresponds to the rank of the matrix $\underline{\boldsymbol{T}}_{(n)}$ (the n-mode matricisation of $\underline{\boldsymbol{T}}$, see reshaping operations in Section 2.4.2) [44]. This relation between the n-ranks and the n-mode matricisation of $\underline{\boldsymbol{T}}$ allows defining an algorithm for the computation of the HOSVD based on the singular value decomposition of each $\underline{\boldsymbol{T}}_{(n)}$ [44]. It follows that $R_n \leq I_n$ for each mode.

Space required. The HOSVD approximation with ranks $R_1, \ldots, R_D$ of a tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ requires $O(R^D + DIR)$ space, where $R = \max_{k \in [1,D]} R_k$ and $I = \max_{k \in [1,D]} I_k$. Hence, the space required still growths exponentially with respect to the order tensor D.

Tensor-Train Decomposition

The Tensor-Train (TT) decomposition was introduced by Oseledets, aiming to define a new tensor decomposition to overcome the limitations of the CP decomposition and the HOSVD. Nevertheless, in the quantum physics community, the TT decomposition has been known under the name Matrix Product States [55, 98, 133].

Definition 2.13 (TT approximation [129])**.** The TT decomposition approximates a D-way tensor into D core tensors $\underline{\boldsymbol{G}}_1, \ldots, \underline{\boldsymbol{G}}_D$ such that [129]:

$$\underline{\boldsymbol{T}}[i_1, \ldots, i_d] \approx \sum_{r_1}^{R_1} \cdots \sum_{r_{D-1}}^{R_{D-1}} \underline{\boldsymbol{G}}_1[i_1, r_1] \underline{\boldsymbol{G}}_2[r_1, i_2, r_2] \ldots \underline{\boldsymbol{G}}_D[r_{D-1}, i_d], \tag{2.34}$$

where each core tensor $\underline{\boldsymbol{G}}_k$ is a 3-way tensor of shape $R_{k-1} \times I_k \times R_k$ (assuming that $R_0 = R_D = 1$). The values of $R_1, \ldots, R_{D-1}$ are the decomposition ranks.

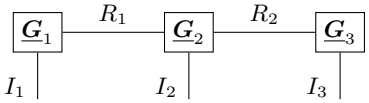


FIGURE 2.11: Tensor network of the TT approximation of a 3-way tensor.

The TT approximation can be represented as a tensor network that contains a vertex for each core tensor $\underline{\boldsymbol{G}}_1, \ldots, \underline{\boldsymbol{G}}_D$. Then, adjacent core tensors are linked together along the shared mode, i.e. $\underline{\boldsymbol{G}}_k$ is linked to $\underline{\boldsymbol{G}}_{k+1}$ along the mode R_k; all the other modes $I_1, \ldots, I_D$ are represented by dangling edges. For example, in Figure 2.11, we represent the TT approximation of a 3-way tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times I_1 \times I_3}$.

TT-ranks. The TT-ranks of a tensor $\underline{\boldsymbol{T}}$ are the smallest values of $R_1, \ldots, R_{D-1}$ which make the Eq. (2.34) an exact equality. Each of these values is related to the rank of a particular matricisation of $\underline{\boldsymbol{T}}$ [129]. This relation allows to define a straightforward algorithm for the computation of the TT decomposition based on the singular value decomposition of such matricisations [129]. Then, the upper bound of each TT-rank is related to the size of such a matrix.

Space required. The TT approximation with ranks $R_1, \ldots, R_D$ of a tenor $\underline{\boldsymbol{T}} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ requires $O(DIR^2 + IR)$ space, where $R = \max_{k \in [1,D]} R_k$ and $I = \max_{k \in [1,D]} I_k$.

2.4.4 Tensors and Machine Learning

In the last decade, an interest in tensors and their decompositions has emerged in the machine learning community. This interest is firstly motivated by the natural representation of multi-modal data, such as RGB images, videos or signals, using tensors. Certainly, the most popular approach to process such a rich data structure is the multi-way data analysis. In this area of research, tensor decompositions are used to process data and reveal complex relationships in them, without resorting to *external* learning models [99, 1, 37].

Although, also learning models has been adapted to handle this type of data. For example, in [160] the authors adapt most of the linear vector models (e.g. linear regression, Support Vector Machine) to tensor data (e.g. multi-linear regression, Support Tensor Machine). On the same line, the authors in [137] develop a low-rank model to handle output tensor data. Regarding neural networks, the Tensor Contraction Layer and Tensor Regression Layer has been introduced empowering NNs to handle tensor data [100]. Clearly, these approaches require parameters which are tensors themselves: due to their curse-of-dimensionality, low-rank approximation of the tensors parameters are necessary to make these approaches feasible in practice.

The usage of tensor decompositions with low rank constraints has been successfully applied also to compress NN layers such as feed-forward [125, 23], convolutional

[105] and recurrent [163]. In some cases, to take advantage of the decomposition, it is necessary to reshape weight matrices and input vectors in tensors (this process is usually called *tensorisation*). However, it is not always clear what is the best tensor shape to select. The compression ability of the tensor decompositions has been exploited also in probabilistic models [126, 176].

The usage of tensors as parameters is useful also to model higher-order interactions among inputs. While this possibility has been discussed in the literature with the introduction of the High-Order NN [65], less attention has been paid on the possibility to use tensor decompositions to reduce the complexity of such networks. Notable exceptions are [156, 127], where the TT decomposition is applied to reduce the complexity of models which consider higher-order interactions among vector entries. Hence, higher-order interactions are usually computed only for a small number of inputs. For example, in [19], the authors use tensors to fuse visual and textual representations; in [171], the authors use tensors to combine predicate, object and subject in event triplets.

If we turn our attention on learning models for structured data, seminal papers such as [116] and [58] highlight the possibility to model higher-order interactions among structure constituents. Nevertheless, in practice, only the interactions among a small number of constituents are modelled as tensors. For example, in [153], the authors proposed a model that leverage tensors to aggregate child information in binary trees. As far as we know, [178] is the only work which applies the TT decomposition to reduce the complexity of High-Order RNNs.

Another interesting connection between tensors and structured data arises considering weighted automata. For example, in [138], the authors show that weighted finite automata and linear second-order RNNs have the same expressive power when considering input sequences of discrete symbols. Also in the case of tree-structured data, weighted tree automata are naturally parametrised by tensors. In this case, low-rank approximations are required to limit the complexity of such models in practical applications [136, 40]. In particular, in [40], tensor decompositions have been used for this purpose.

Before continuing with the rest of our thesis, it is worth highlighting other two interesting application of tensors in ML. Even if they are not directly connected with our work, they underline the power of the tensor theory and the benefits that ML community can obtain from it. The former application is to use tensors and their decomposition as a tool in probabilistic models to estimate latent model parameters which exploit a certain tensor structure in their low-order observable moments [3]. The latter one, is to use tensors to study theoretical properties of deep learning models such as convolutional neural network [39, 38, 174] and recurrent neural network [92, 91]. In this perspective, the whole learning problem can be depicted as the estimation of a tensor claiming that the implicit regularisation of current learning algorithms leads to low-rank solutions rather than minimum norm solutions [4, 139].

2.5 Model Taxonomy

In the following chapters, we introduce several models for structured data. Thus, to facilitate the reading, it is worth introducing the model taxonomy followed in the rest of the thesis.

All the model names are in the form:

$$X - Y - Z,$$

where X, Y and Z are strings.

The string Z indicates the model class, i.e. if it is a probabilistic, neural or LSTM-based model.

The string Y indicates the parametrisation used to define the aggregation function. For example, it specifies the tensor decomposition leveraged by the model.

The string X provides additional information on the model. For example, it specifies if the model can handle unbounded data.

In Table 2.1, we report all the model names used in the rest of the thesis according to the introduced taxonomy.

Model	Data	Class	Model Parametrisation				
			Existing in Literature	Tensor Based			
				Full Tensor	Tensor Decompositions		
					Canonical	Higher-Order Singular Value	Tensor-Train
Recursive Models							
Bounded	Bounded	Probabilistic	SP-HRTM*	Full-HRTM	CP-HRTM	HOSVD-HRTM	TT-HRTM
		One layer NN	Sum-RecNTN*	Full-RecNTN	CP-RecNTN	HOSVD-RecNTN	TT-RecNTN
		LSTM-based	Sum-LSTM*	Full-LSTM	CP-LSTM	HOSVD-LSTM	TT-LSTM
	Unbounded	Probabilistic	Infinite-SP-HRTM	-	Infinite-CP-HRTM	-	Infinite-TT-HRTM
		One-Layer NN	-	-	Infinite-CP-RecNTN	-	Infinite-TT-RecNTN
		LSTM-based	Infinite-Sum-LSTM*	-	Infinite-CP-LSTM	-	Infinite-TT-LSTM
Unbounded	Bounded	Probabilistic	-	-	-	Bayesian-HOSVD-HRTM	-
Mixture Models							
Bounded	Bounded	Probabilistic	Mix-SP-HRTM	-	-	-	
Unbounded	Bounded	Probabilistic	BNP-SP-HRTM	-	-	-	

TABLE 2.1: Taxonomy of all the models introduced in this thesis. Models with * are already known in literature and are not this thesis contribution.

Chapter 3

A Tensor Framework for Recursive Models

3.1 Introduction

This chapter aims to build a connection between recursive models for structured data and tensors. In probabilistic models, such a connection arises naturally if the state-transition distribution considers all the possible joint configurations of input labels and child hidden states [58]. On the other hand, in the context of neural models, a tensor parametrisation of the state-transition function arises if we consider higher-order interactions among input labels and structural context [58, 104]. In both cases, the order of the tensor parameters grows linearly with respect to the maximum out-degree L of the structures taken into account. Thus, there is an exponential relation between the number of model parameters and L. Interestingly, this connection relates the complexity of recursive models (intended as the number of parameters) with the input structures complexity (intended as the maximum out-degree).

While a tensor parametrisation is feasible in practice only for small values of L, we propose to leverage it to build a framework for recursive models. In this framework, different recursive models can be defined by specifying different constraints on the tensor parameter. From this perspective, the constraints imposed allows approximating the full-tensor state-transition function and reducing the number of parameters required by the model. The price to pay for such an approximation is the introduction of a specific inductive bias into the recursive model. Consequently, the models with a full-tensor parametrisation have a low inductive bias since they do not add any constraint to the state-transition function.

Interestingly, the framework does not depend on the nature of the models and therefore can be applied in both the probabilistic and neural context. In this respect, we show how a probabilistic and a neural approximation commonly used in the literature can be cast in the proposed framework. The probabilistic approximation relies on the Switching Parent technique [147] to approximate the tensor state-transition distribution [12, 11]. The neural approximation consists in considering only first-order interactions among input labels and structural context. In both cases, we focus on the inductive bias introduced by the approximation.

In Section 3.2, we introduce a probabilistic and a neural model whose state-transition functions are parametrised by a tensor. These models will be used as a mould for recursive models which approximate the state-transition function. We also show that the tensor parametrisation proposed implies a low inductive bias. In Section 3.3, we show how two well-known literature models can be framed in our framework, highlighting the inductive bias they introduce by approximating the tensor parameter. Finally, in Section 3.4, we draw our conclusion.

3.2 General Tensor Framework

In this section, we lay the foundation of our framework presenting a probabilistic and a neural model whose state-transition functions are parametrised by a tensor. For both models, we discuss their inductive bias and how their parameters can be learned from data. For the sake of simplicity, we do not specialise the output functions since they are not related to the tensor parametrisation of the models.

3.2.1 Hidden Recursive Tensor Models

The Hidden Recursive Tensor Model (HRTM) is a recursive model for learning structured transductions which is equivalent to HRM introduced in Section 2.3.3. Nevertheless, we add the term tensor to highlight that it defines a state-transition function parametrised by a tensor. As in HRM, the input and output labels as well as the hidden states are modelled as realisations of random variables. In particular, the hidden states are modelled as realisations of discrete random variables with C states.

Tensor State-Transition Distribution

The state-transition distribution models the information flow from the child hidden variables to the parent hidden variable. Let v be a node of the input structure $\boldsymbol{\mathcal{X}}$. If we assume that the input labels are categorical with M values, the state-transition distribution $P(h_v \mid x_v, h_{v1}, \dots, h_{vL})$ is a categorical distribution. Recalling the parametrisation of categorical conditional distributions introduced in Section 2.2.1, the state-transition distribution is parametrised as:

$$P(H_v = k \mid X_v = i, H_{v1} = j_1, \dots, H_{vL} = j_L, \underline{\boldsymbol{P}}) = \underline{\boldsymbol{P}}[i, j_1, \dots, j_L, k], \tag{3.1}$$

where $\underline{\boldsymbol{P}}$ is a $(L+2)$-tensor of size $(M+1) \times (C+1)^L \times C$. We add the special state $\perp$ to indicate the absence of the input label or a child node. The entry $\underline{\boldsymbol{P}}[i, j_1, \dots, j_L, k]$ indicates the probability to observe the hidden variable H_v in the state k, given that X_v is in the state i and each child hidden variable H_{vl} is in state j_l. Since $\underline{\boldsymbol{P}}$ represents a probability distribution, it must satisfy the probability constraints (see Section 2.2.1).

In this formulation, we assume that the input labels are categorical. However, this is not always the case. If the input labels are numerical (e.g. real vectors), we cannot

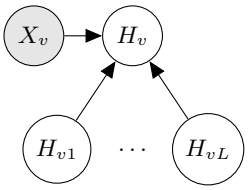


FIGURE 3.1: BN depicting the HRTM state-transition distribution.

enumerate all the possible joint configurations of the input label and the child hidden states. Nevertheless, the state-transition function can be rewritten as:

$$\begin{aligned} &P(h_v \,|x_v, h_{v1}, \ldots, h_{vL}) \\ &\qquad = P(x_v \mid h_v, h_{v1}, \ldots, h_{vL}) P(h_v \mid h_{v1}, \ldots, h_{vL}) \\ &\qquad = P(x_v \mid h_v) P(h_v \mid h_{v1}, \ldots, h_{vL}), \end{aligned} \tag{3.2}$$

where the last equality holds assuming that the input label and the child hidden states are independent given the parent hidden state h_v. The distribution $P(h_v \mid h_{v1}, \ldots, h_{vL})$ is still parametrised by a $(L+1)$-way tensor (since all hidden variables are categorical); thus, also in this case, we obtain a number of parameters which grows exponentially wit respect to L. The distribution $P(x_v \mid h_v)$ is defined as a continuous distribution where the value h_v indicates a different parametrisation of the distribution itself, e.g. $x_v \sim \mathcal{N}(\mu_{h_v}, \sigma_{h_v})$.

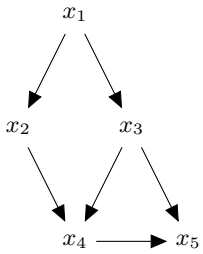
Inductive Bias

The inductive bias introduced by HRTM is strictly related to the independence assumptions imposed by the model. Since the HRTM encoding network can be graphically represented as a BN, we can state the independence assumptions induced by HRTM directly from the BN.

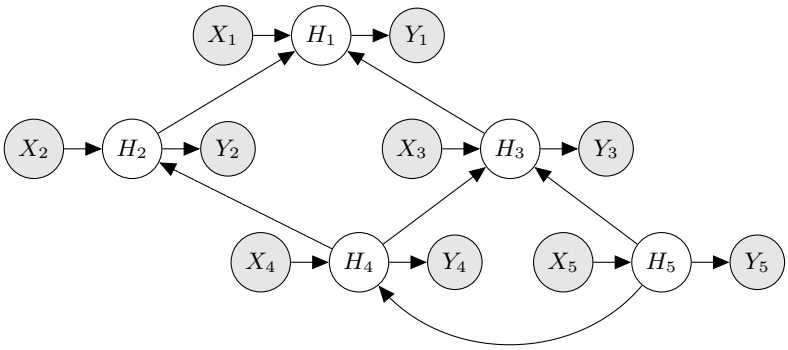
In Figure 3.1, we depict the BN associated to the HRTM state-transition distribution. The parent hidden variable H_v is a collider in every path among child hidden variables $H_{v1}, \ldots, H_{vL}$ and the input variable X_v. Hence, by definition of d-separation (see Definition 2.3), we can state that the variables $\{X_v, H_{v1}, \ldots, H_{vL}\}$ are pairwise independent. This independence assumption ensures us that the hidden state of a node is independent of its siblings (given that their parent hidden state is not observed). Thus, the hidden state of a node depends only on its descendants, reflecting the causality of the transduction process (see Section 2.3.1).

Moreover, the parent variable H_v is independent of its descendants variables (except its child variables) given its child variables, i.e. $H_v \perp\!\!\!\perp \boldsymbol{\mathcal{H}}_v \setminus \{H_v, H_{v1}, \ldots, H_{vL}\} \mid H_{v1}, \ldots, H_{vL}$. This independence assumption is intrinsic in the recursive nature of the transduction: it ensures us that, if the child hidden states are known, we have all the information necessary to determine the parent hidden state.

Thus, we can state that HRTM has a low inductive bias since it contains only the independence assumptions derived from modelling the learning task as a causal



(A) An isomorphic transduction $\boldsymbol{\mathcal{Y}} = \tau(\boldsymbol{\mathcal{X}})$



(B) Bayesian network induced by HRTM. Shaded nodes indicate visible variables.

FIGURE 3.2: BN induced by HRTM on pair of observed trees $(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}})$.

recursive transduction.

It is worth to point out that in the general case, the structure can induce dependencies also among child nodes. In Figure 3.2a, we show an example of input structure $\boldsymbol{\mathcal{X}} \in \mathcal{X}^{\#_2^2}$ which contains links between child nodes: e.g. x_4 and x_5 are child nodes of x_3, but x_5 is also a child of x_4. Note that this is a valid DOAG since there are no directed cycles in the structure. In Figure 3.2b, we show the encoding network induced by the HRTM on the structure of $\boldsymbol{\mathcal{X}}$. Clearly, the hidden variable H_4 and H_5 are not independent even if they are children of H_3 since a directed link connects them. The same holds between H_2 and H_3: due to the common child H_4, they are not independent even if their parent H_1 is not observed. Nevertheless, the independence assumptions stated before are still true if we condition over the descendant variables. For example, given the state of H_4 and H_5, the variable H_2 and H_3 are independent (i.e. $H_2 \perp\!\!\!\perp H_3 \mid \{H_4, H_5\}$). Thus, also in this case, the hidden state of a node depends only on its descendants.

Learning

The parameters of HRTM can be learned from the observed data by posterior inference. Nevertheless, as we have already pointed out in Section 2.3.3, the learning procedure worsens if the structures are not singly connected. In this section, we present the learning algorithm which can be applied only when the observed structures are singly-connected DOAGs.

Let $\hat{D} = \{(\boldsymbol{\mathcal{X}}^1, \boldsymbol{\mathcal{Y}}^1), \ldots, (\boldsymbol{\mathcal{X}}^N, \boldsymbol{\mathcal{Y}}^N)\}$ be a training set with N pairs of singly-connected DOAGs, the goal of the learning algorithm is to find the model parameters $\theta = \{\underline{\boldsymbol{P}}, \theta_g\}$ which maximise the likelihood of the output structures. Since we have not detailed the output distribution, we use a generic parametrisation θ_g. The algorithm is based on a specialisation of the EM algorithm (see Section 2.2.1).

E-step. The aim of the E-step is the computation of the hidden variables posterior given the visible ones. For the sake of simplicity, we introduce the procedure to compute the posterior on a generic pair $(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}})$. Clearly, it must be applied to each training pair $(\boldsymbol{\mathcal{X}}^i, \boldsymbol{\mathcal{Y}}^i)$.

The posteriors that should be computed are:

$$\epsilon_{v1,\ldots,vL,v} = P(h_{v1}, \ldots, h_{vL}, h_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \quad \forall v \in \mathrm{vert}(\boldsymbol{\mathcal{X}}), \tag{3.3a}$$

$$\epsilon_v = P(h_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \quad \forall v \in \mathrm{vert}(\boldsymbol{\mathcal{X}}). \tag{3.3b}$$

The computation is efficiently carried out by a tailored version of the upward-downward algorithm.

Upward pass. The upward pass is a recursive procedure which computes the following quantities:

$$\beta_v = P(h_v \mid \boldsymbol{\mathcal{X}}_v, \boldsymbol{\mathcal{Y}}_v). \tag{3.4}$$

The recursion goes form the sinks to the super-source. Hence, the base case of the recursion is defined on the sinks nodes. Let v be a sink node, the quantity β_v is computed as:

$$\beta_v[h_v] = \frac{P(y_v \mid h_v, \theta_g)\underline{\boldsymbol{P}}[x_v, \perp, \ldots, \perp, h_v]}{Z}, \tag{3.5}$$

where $Z = P(y_v \mid x_v)$ is a normalising constant. Note that all the child variables are in the state $\perp$ since v does not have child nodes.

Let v be an internal node, the computation of β_v can be obtained recursively as:

$$\beta_v[h_v] = \frac{\sum_{h_{v1}} \cdots \sum_{h_{vL}} P(y_v \mid h_v, \theta_g)\underline{\boldsymbol{P}}[x_v, h_{v1}, \ldots, h_{vL}, h_v] \prod_{l=1}^{L} \beta_{vl}[h_{vl}]}{Z}, \tag{3.6}$$

where $Z = \frac{P(\boldsymbol{\mathcal{Y}}_v|\boldsymbol{\mathcal{X}}_v)}{\prod_{l=1}^{L} P(\boldsymbol{\mathcal{Y}}_{vl}|\boldsymbol{\mathcal{X}}_{vl})}$ is a normalising constant.

Multiplying together all the normalisation constants computed on each node, we obtain the likelihood of the output structure given the input one $P(\boldsymbol{\mathcal{Y}} \mid \boldsymbol{\mathcal{X}})$. This quantity is exactly the one we would like to increase during the training algorithm. Hence, it is useful to check its value during the training phase.

Downward pass. The downward pass is a recursive procedure which computes the posterior for each node. The recursion goes from the super-source node s to the sink nodes. The super-source is the base case of the recursion since $\beta_s = \epsilon_s$.

The recursive computation of the posteriors can be divided in two steps. In the first one, we compute the joint posterior $\epsilon_{v1,\dots,vL,v}$ given the parent posterior ϵ_v:

$$\epsilon_{v1,\dots,vL,v}[h_{v1},\dots,h_{vl},h_v] = \frac{\epsilon_v[h_v]\underline{\boldsymbol{P}}[x_v,h_{v1},\dots,h_{vL},h_v]\prod_{l=1}^{L}\beta_{vl}[h_{vl}]}{\sum_{h_{v1}}\cdots\sum_{h_{vL}}\underline{\boldsymbol{P}}[x_v,h_{v1},\dots,h_{vL},h_v]\prod_{l=1}^{L}\beta_{vl}[h_{vl}]}. \tag{3.7}$$

Then, we can obtain the posterior of each child hidden variable $\epsilon_{v1},\dots,\epsilon_{vL}$ by marginalisation:

$$\epsilon_{vl}[h_{vl}] = \sum_{h_{v1}}\cdots\sum_{h_{v(l-1)}}\sum_{h_{v(l+1)}}\cdots\sum_{h_{vL}}\epsilon_{v,v1,\dots,vL}[h_{v1},\dots,h_{vL},h_v]. \tag{3.8}$$

This terminates the upward-downward procedure.

M-step. In the M-step, we aim to find the model parameters which increase the expectation of the complete log-likelihood $\mathcal{L}(\underline{\boldsymbol{P}},\theta_g)$ of the observed data with respect to the posteriors computed during the E-step:

$$\begin{aligned}\mathcal{L}(\underline{\boldsymbol{P}},\theta_g) = &\sum_{n=1}^{N}\sum_{v\in\mathrm{vert}(\boldsymbol{\mathcal{X}}^n)}\mathbb{E}\left[\log P(y_v \mid h_v,\theta_g)\right]_{\epsilon_v}\\ &+\sum_{n=1}^{N}\sum_{v\in\mathrm{vert}(\boldsymbol{\mathcal{X}}^n)}\mathbb{E}\left[\log P(h_v \mid x_v,h_{v1},\dots,h_{vL},\underline{\boldsymbol{P}})\right]_{\epsilon_{v,v1,\dots,vL}}.\end{aligned} \tag{3.9}$$

Thus, the parameter updates are obtained by solving the following optimisation problems:

$$\underline{\boldsymbol{P}} = \arg\min_{\underline{\boldsymbol{P}}}\sum_{n=1}^{N}\sum_{v\in\mathrm{vert}(\boldsymbol{\mathcal{X}}^n)}\mathbb{E}\left[\log P(h_v \mid x_v,h_{v1},\dots,h_{vL},\underline{\boldsymbol{P}})\right]_{\epsilon_{v,v1,\dots,vL}}, \tag{3.10a}$$

$$\theta_g = \arg\min_{\theta_g}\sum_{n=1}^{N}\sum_{v\in\mathrm{vert}(\boldsymbol{\mathcal{X}}^n)}\mathbb{E}\left[\log P(y_v \mid h_v,\theta_g)\right]_{\epsilon_v}. \tag{3.10b}$$

The optimisation problem in Eq. (3.10a) can be solved analytically, obtaining the following state-transition parameter update:

$$\underline{\boldsymbol{P}}[i,j_1,\dots,j_L,k] \propto \sum_{n=1}^{N}\sum_{v\in\mathrm{vert}(\boldsymbol{\mathcal{X}}^n)}\epsilon_{v1,\dots,vL,v}[j_1,\dots,j_L,k]\times\mathbb{I}[x_v=i], \tag{3.11}$$

where the indicator function $\mathbb{I}[p]$ is equal to 1 if and only if the predicate p is true.

3.2.2 Recursive Neural Tensor Networks

A Recursive Neural Tensor Network (RecNTN) is a neural recursive model for learning structured transductions which is based on a tensor parametrisation of the neural state-transition function. The tensor formulation arises defining a multi-affine map which considers higher-order interactions among the input label and the structural

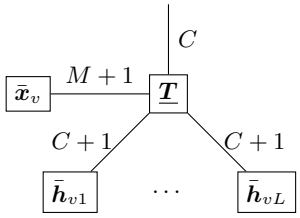


FIGURE 3.3: Tensor network representing the multi-affine map in the RecNTN state-transition function.

context of a node. In the following, we assume that the input label and the hidden state associated to each node v are real vectors of size M and C, respectively (i.e. $\boldsymbol{x}_v \in \mathbb{R}^M$ and $\boldsymbol{h}_v \in \mathbb{R}^C$).

Neural Tensor State-Transition Function

The neural state-transition function is a neural layer which computes the hidden state of a node given its input label and its child hidden states. A neural layer is commonly defined as the composition of a non-linear activation function with an affine operator (see Section 2.2.2). Thus, a generic neural state transition function can be defined as:

$$h_v = \sigma(\psi(x_v, h_{v1}, \dots, h_{vL})), \tag{3.12}$$

where v is a generic node of the input structure, σ is the non-linear activation function and ψ is an affine function. Interestingly, the existence of the structural context (i.e. the hidden state of the child nodes of v) complicates the definition of ψ since it becomes a multi-variate function. In fact, in the case of flat data, the ψ function is univariate since it has only the vector x_v as argument (see Section 2.2.2). Thus, it seems natural to define ψ as a multi-variate affine function, or simply *multi-affine* function. As in the definition of multi-linear function, a multi-affine function is a multi-variate function which is affine in each dimension.

The neural tensor state-transition function defines the multi-affine operator by means of a tensor:

$$\boldsymbol{h}_v = \sigma\left(\underline{\boldsymbol{T}}(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{vL})\right), \tag{3.13}$$

where σ is a non-linear activation function, $\{\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{vL}\}$ are homogeneous coordinates of the input label and the child hidden states (i.e. $\bar{\boldsymbol{a}} = [\boldsymbol{a}; 1]$); $\underline{\boldsymbol{T}} \in \mathbb{R}^{(M+1)\times(C+1)\times\cdots\times(C+1)\times C}$ is an *augmented tensor* which represents the multi-affine map ψ. We use the term *augmented tensor* by analogy to augmented matrices that parametrise affine maps. If a child node or an input label is missing, they are commonly substituted by the zero vector. Thus, the tensor $\underline{\boldsymbol{T}}$ can be applied also when the out-degree of the node is less than L. In Figure 3.3, we represent the aggregation performed by $\underline{\boldsymbol{T}}$ as a tensor network.

The function induced by $\underline{\boldsymbol{T}}$ is clearly a multi-affine map. In fact, fixing all the arguments of $\underline{\boldsymbol{T}}$ except one we obtain a univariate linear function. Since such a linear

function is applied on a homogeneous coordinate vector, it represents an affine function (see Section 2.2.2).

Interestingly, the augmented tensor $\underline{\boldsymbol{T}}$ aggregates the input label and the child hidden states considering interactions of any order among them. For the sake of simplicity, let us consider a node v of a binary tree (i.e. $\boldsymbol{\mathcal{X}} \in \mathcal{X}^{\#_2^1}$). By applying the definition of the function induced by a tensor introduced in Eq. (2.24), we obtain:

$$\underline{\boldsymbol{T}}(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{h}}_{v1}, \bar{\boldsymbol{h}}_{v2}) = \sum_{i=1}^{M+1} \sum_{j_1=1}^{C+1} \sum_{j_2=1}^{C+1} \underline{\boldsymbol{T}}[i, j_1, j_2, :] \bar{\boldsymbol{x}}[i] \bar{\boldsymbol{h}}_{v1}[j_1] \bar{\boldsymbol{h}}_{v2}[j_2]. \tag{3.14}$$

Since the input label and the child hidden states are represented in homogeneous coordinate, their last entry is equal to 1 by definition. Thus, the contribution of such vectors can be neglected in the multiplications when the last entry is indexed. This allows rewriting the application of the augmented tensor as:

$$\begin{aligned}
\underline{\boldsymbol{T}}(\bar{\boldsymbol{x}}_v,\bar{\boldsymbol{h}}_{v1}, \bar{\boldsymbol{h}}_{v2}) &= \sum_{i=1}^{M+1} \sum_{j_1=1}^{C+1} \sum_{j_2=1}^{C+1} \underline{\boldsymbol{T}}[i, j_1, j_2, :] \bar{\boldsymbol{x}}[i] \bar{\boldsymbol{h}}_{v1}[j_1] \bar{\boldsymbol{h}}_{v2}[j_2] \\
&= \underline{\boldsymbol{T}}[M+1, C+1, C+1, :] + \sum_{i=1}^{M} \underline{\boldsymbol{T}}[i, C+1, C+1, :] \boldsymbol{x}[i] \\
&+ \sum_{i=1}^{C} \underline{\boldsymbol{T}}[M+1, j_1, C+1, :] \boldsymbol{h}_{v1}[j_1] + \sum_{i=1}^{C} \underline{\boldsymbol{T}}[M+1, C+1, j_2, :] \boldsymbol{h}_{v2}[j_2] \\
&+ \sum_{i=1}^{M} \sum_{j_1=1}^{C} \underline{\boldsymbol{T}}[i, j_1, C+1, :] \boldsymbol{x}[i] \boldsymbol{h}_{v1}[j_1] + \sum_{i=1}^{M} \sum_{j_2=1}^{C} \underline{\boldsymbol{T}}[i, C+1, j_2, :] \boldsymbol{x}[i] \boldsymbol{h}_{v2}[j_2] \\
&+ \sum_{j_1=1}^{C} \sum_{j_2=1}^{C} \underline{\boldsymbol{T}}[M+1, j_1, j_2, :] \boldsymbol{h}_{v1}[j_1] \boldsymbol{h}_{v2}[j_2] \\
&+ \sum_{i=1}^{M} \sum_{j_1=1}^{C} \sum_{j_2=1}^{C} \underline{\boldsymbol{T}}[i, j_1, j_2, :] \boldsymbol{x}[i] \boldsymbol{h}_{v1}[j_1] \boldsymbol{h}_{v2}[j_2],
\end{aligned} \tag{3.15}$$

where each terms in the equation refers to a different sub-tensor of $\underline{\boldsymbol{T}}$. By renaming these sub-tensors, we obtain:

$$\begin{aligned}
&\underline{\boldsymbol{T}}(\bar{\boldsymbol{x}}_v,\bar{\boldsymbol{h}}_{v1}, \bar{\boldsymbol{h}}_{v2}) \\
&\quad = \boldsymbol{b} && \text{(bias vector)} \\
&\quad + \boldsymbol{W}\boldsymbol{x}_v + \boldsymbol{U}_1\boldsymbol{h}_{v1} + \boldsymbol{U}_2\boldsymbol{h}_{v2} && \text{(first-order interactions)} \\
&\quad + \underline{\boldsymbol{A}}(\boldsymbol{x}_v, \boldsymbol{h}_{v1}) + \underline{\boldsymbol{B}}(\boldsymbol{x}_v, \boldsymbol{h}_{v2}) + \underline{\boldsymbol{C}}(\boldsymbol{h}_{v1}, \boldsymbol{h}_{v2}) && \text{(second-order interactions)} \\
&\quad + \underline{\boldsymbol{D}}(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \boldsymbol{h}_{v2}), && \text{(third-order interactions)}
\end{aligned} \tag{3.16}$$

where we clearly report the interaction order modelled by each sub-tensor. The generalisation to the case of a node with L children is straightforward. Thus, RecNTN is able to model the interactions of any order among input labels and child hidden states.

In some cases, it can be useful to avoid representing input labels in homogeneous coordinates. For example, if the input labels are categorical with M states, we would like to use them to select a different parametrisation of the state-transition function. To this end, we can define $\boldsymbol{x}_v \in \mathbb{R}^M$ as the one-hot encoding of the observed input label; removing the homogeneous coordinates representation of $\boldsymbol{x}_v$, the contraction between $\underline{\boldsymbol{T}}$ and $\boldsymbol{x}_v$ is equivalent to selecting the sub-array $\underline{\boldsymbol{T}}[i, :, \dots, :]$, where $\boldsymbol{x}_v[i] = 1$. This sub-array defines the multi-affine operator associated to the label i and it is used to combine child hidden states.

Inductive Bias

The inductive bias introduced by RecNTN depends on the assumptions made by the state-transition function to aggregate input labels and contextual information. The first step is to show that every multi-affine function can be represented by an augmented tensor. While this property can be directly derived from the universality of the tensor product [74], we provide an alternative derivation based on the representation of multi-affine maps as a sum of multi-linear functions [61].

Theorem 1 (Augmented tensors)**.** *Let $\psi : \mathbb{R}^{I_1} \times \cdots \times \mathbb{R}^{I_D} \to \mathbb{R}^K$ be a multi-affine function. There exists an* augmented tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{(I_1+1)\times\cdots\times(I_D+1)\times K}$ *such that, for every $\boldsymbol{a}_d \in \mathbb{R}^{I_d}$, it holds:*

$$\psi\left(\boldsymbol{a}_1, \dots, \boldsymbol{a}_d\right) = \underline{\boldsymbol{T}}\left(\bar{\boldsymbol{a}}_1, \dots, \bar{\boldsymbol{a}}_d\right),$$

where $\bar{\boldsymbol{a}}_d = [\boldsymbol{a}_d; 1]$ are the homogeneous coordinate of d-th input vector $\boldsymbol{a}_d$.

Proof sketch (complete proof in Appendix C.1). From Lemma 4.1.3 in [61], it follows that every multi-affine function ψ is completely determined by the sum of 2^{D-1} unique multi-linear functions; each function is applied on a different non-empty subset of $\{\boldsymbol{a}_1, \dots, \boldsymbol{a}_D\}$ plus a constant:

$$\psi(\boldsymbol{a}_1, \dots, \boldsymbol{a}_D) = \psi(\boldsymbol{0}, \dots, \boldsymbol{0}) + \sum_{\substack{\mathcal{S}\subseteq\{1,\dots,d\} \\ \mathcal{S}=\{j_1,\dots,j_M\}, M\geq 1}} f_{\mathcal{S}}(\boldsymbol{a}_{j_1}, \dots, \boldsymbol{a}_{j_M}), \tag{3.17}$$

where $\boldsymbol{0}$ is the zero vector. Hence, it is enough to show that (1) all these multi-linear functions completely determine the entries of $\underline{\boldsymbol{T}}$ and (2) the application of the multi-linear function defined by $\underline{\boldsymbol{T}}$ on homogeneous coordinates of the input vectors induces the summation of such functions. □

Theorem 1 ensure us that any recursive model which use a multi-affine map to aggregate the input label and the structural context in its state-transition function can be simulated by a RecNTN with the same hidden state size. It is worth highlighting that we obtain a tensor parametrisation also if we constrain ψ to be a multi-linear function, as has been done in seminal papers such as [58, 104]. However, the multi-linear function is not able to represent the summation (i.e. the function which returns

the sum of all its argument) since it is a multi-affine map. As it will be clear in Section, 3.3.2, the summation plays a key role in the definition of the first-order RecNTN.

Note that there are also more complex aggregation functions, such as the multivariate polynomial functions, that cannot be represented by augmented tensors since they are not multi-affine maps. Nevertheless, as far as we know, these functions have been used only in the context of flat data [65, 140, 109, 126].

Thus, constraining ψ to be a multi-affine map seems to be a reasonable choice. Under this assumption, we can state that RecNTN has a low inductive bias since it does not impose any further constraint to the state-transition function definition.

Learning

The parameters of RecNTNs are learned by minimising a loss function through gradient descent. The gradients of the loss functions with respect to the model parameters can be computed by applying the BPTS algorithm (see Section 2.3.3).

To better characterise the model, it is worth showing the gradients computation. First, we recall the definition of the function induced by a tensor (see Section 2.4):

$$\begin{aligned} h_v[k] &= \underline{\boldsymbol{T}}(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{vL})[k] \\ &= \sum_{i}^{M+1} \sum_{j_1}^{C+1} \cdots \sum_{j_L}^{C+1} \underline{\boldsymbol{T}}[i, j_1, \dots, j_L, k] \bar{\boldsymbol{x}}_v[i] \bar{\boldsymbol{h}}_{v1}[j_1] \dots \bar{\boldsymbol{h}}_{vL}[j_L]. \end{aligned} \tag{3.18}$$

Hence, the gradient of the output value $\underline{\boldsymbol{T}}(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{vL})[k]$ with respect of the entry $\underline{\boldsymbol{T}}[i, j_1, \dots, j_L, k]$ is given by:

$$\frac{\delta \underline{\boldsymbol{T}}(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{vL})[k]}{\delta \underline{\boldsymbol{T}}[i, j_1, \dots, j_L, k]} = \bar{\boldsymbol{x}}_v[i] \bar{\boldsymbol{h}}_{v1}[j_1] \dots \bar{\boldsymbol{h}}_{vL}[j_L]. \tag{3.19}$$

Similarly, the gradient of $\underline{\boldsymbol{T}}(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{vL})[k]$ with respect to the argument $\boldsymbol{h}_{v1}[j_1]$ is obtained as:

$$\frac{\delta \underline{\boldsymbol{T}}(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{vL})[k]}{\delta \boldsymbol{h}_{v1}[j_1]} = \sum_{i}^{M+1} \sum_{j_2}^{C+1} \cdots \sum_{j_L}^{C+1} \underline{\boldsymbol{T}}[i, j_1, \dots, j_L, k] \bar{\boldsymbol{x}}_v[i] \bar{\boldsymbol{h}}_{v2}[j_1] \dots \bar{\boldsymbol{h}}_{vL}[j_L], \tag{3.20}$$

and it allows back-propagating the error to child hidden states.

Both gradients require the multiplication of L terms; hence, for large values of L, the computation of such gradients can be numerically unstable.

3.3 Existing Approximation

In this section, we show that a probabilistic and a neural recursive model commonly used in the literature can be interpreted as approximation of the proposed tensor models. Both approximations rely on the summation of the input label and the child nodes contributions. Such a summation introduces a strong inductive bias since they

impose independence among the input label and the structural context of a node. Moreover, these approximations cannot be interpreted as decompositions of the tensor parameters.

3.3.1 Switching-Parent

The Switching Parent (SP) [147] approximation has been introduced to approximate the state-transition distribution of high-order Markov chain and factorial hidden Markov model [147]. Nevertheless, it has been also used to reduce the complexity of HRTM state-transition distribution [12, 11].

The idea of the SP approximation is to decompose a conditional state-transition distribution into a mixture of simpler distributions. To this end, it introduces a mixture variable which allows decoupling the contribution of conditioning variables. In the following, we show the application of the SP approximation to reduce the number of parameters required by the HRTM state-transition distribution. We denote this model as SP-HRTM and it is equivalent to Input-Output SP-BHTMM [11].

Let v be a node of the input structure $\boldsymbol{\mathcal{X}} \in \mathcal{X}^{\#_L^P}$, the SP approximation defines a new discrete random variable S_v which conditioning the hidden variable H_v. Moreover, the following equality is imposed:

$$P(h_v \mid x_v, h_{v1}, \dots, h_{vL}, S_v = l) = P(h_v \mid S_v = l, x_v, h_{vl}). \tag{3.21}$$

When the mixture variable S_v is observed, it allows to separate the contribution of the child hidden variables. In particular, if $S_v = l$, the variable H_v depends only on the input label x_v and the child variable H_{vl}. Unfortunately, we cannot represent graphically this relation; hence, the BN is equal to the BN of the full state-transition distribution (see Figure 3.4).

Finally, we can derive the approximation of the state-transition distribution:

$$\begin{aligned} P(h_v \mid x_v, h_{v1}, \dots, h_{vL}) &= \sum_{l=l}^{L} P(h_v, S_v = l \mid x_v, h_{v1}, \dots, h_{vL}) \\ &= \frac{\sum_{l=l}^{L} P(h_v, S_v = l, x_v, h_{v1}, \dots, h_{vL})}{P(x_v, h_{v1}, \dots, h_{vL})} \\ &= \sum_{l=l}^{L} P(h_v \mid S_v = l, x_v, h_{v1}, \dots, h_{vL}) P(S_v = l \mid x_v, h_{v1}, \dots, h_{vL}) \\ &= \sum_{l=l}^{L} P(S_v = l \mid x_v) P(h_v \mid S_v = l, x_v, h_{vl}), \end{aligned} \tag{3.22}$$

where we have assumed that the mixture variable S_v depends only on the input label x_v. This allows defining a different approximated state-transition function for each input label.

The Eq. (3.22) makes clear the connection between the SP approximation and mixture models. In fact, the hidden state h_v can be obtained by a generative process

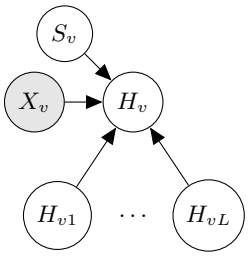


FIGURE 3.4: BN representing the SP-HRTM state-transition distribution.

which comprises two steps: (1) a state l of the variable S_v is drawn; (2) the hidden state h_v is sampled from the distribution $P(h_v \mid S_v = l, x_v, h_{vl})$ selected accordingly to the state l of S_v. Hence, the variable S_v is commonly denoted as the *mixture variable*, while distributions $P(h_v \mid S_v = l, h_{vl}, x_v)$ are referred to as the *mixture components*. Each mixture component describes the relation between the hidden variable H_v and its l-th child hidden variable. Clearly, the state of S_v is not observed. Hence, its marginalisation mixes the contribution of child hidden variables. Unfortunately, when the mixture variable is marginalised, no independence assumptions can be established.

The SP approximation is completely determined by the mixture variable distribution and the mixture components distributions. Assuming that the input label is modelled as a discrete random variable with M states, the mixture variable distribution is parametrised by the matrix $\boldsymbol{S} \in \mathbb{R}_{\geq 0}^{(M+1)\times L}$: $P(S_v = l \mid x_v, \boldsymbol{S})$. On the other hand, each mixture component is parametrised by a 3-rd order tensor $\underline{\boldsymbol{U}}_l \in \mathbb{R}_{\geq 0}^{(M+1)\times(C+1)\times C}$: $P(h_v \mid S_v = l, h_{vl}, x_v, \underline{\boldsymbol{U}}_l)$. Hence, the SP approximation requires $(M+1)L + L(M+1)(C+1)C = O(LMC^2)$ parameters.

While the SP approximation has a nice probabilistic interpretation, it does not correspond to a particular decomposition of the tensor $\underline{\boldsymbol{P}}$ which parametrises the HRTM state-transition distribution. In fact, expressing the Equation 3.22 in terms of parameters, we obtain:

$$\underline{\boldsymbol{P}}[x_v, h_{v1}, \ldots, h_{vL}, h_v] = \sum_{l=1}^{L} \underline{\boldsymbol{U}}_l[x_v, h_{v_l}, h_v]\boldsymbol{S}[x_v, l]. \tag{3.23}$$

Equivalently, we can write:

$$\underline{\boldsymbol{P}} = \sum_{l=1}^{L} \underline{\boldsymbol{U}}_l^*, \tag{3.24}$$

where $\underline{\boldsymbol{U}}_l^*[x_v, h_{v1}, \ldots, h_{vL}, h_v]$ are tensors with the same order of $\underline{\boldsymbol{P}}$ which are obtained by replicating the entries of $\underline{\boldsymbol{U}}_l$ and $\boldsymbol{S}$ opportunely, i.e.:

$$\underline{\boldsymbol{U}}_l^*[x_v, h_{v1}, \ldots, h_{vL}, h_v] = \underline{\boldsymbol{U}}_l[x_v, h_{vl}, h_v]\boldsymbol{S}[x_v, l], \quad \forall l' \neq l. \forall h_{vl'} \in [1, C]. \tag{3.25}$$

Hence, the SP approximation impose that the $\underline{\boldsymbol{P}}$ is obtained as a sum of L tensors which are obtained applying a proper replication of the entries in $\underline{\boldsymbol{U}}_l$ and $\boldsymbol{S}$. Clearly, there are tensors that cannot be represented using such a summation.

Moreover, the SP approximation is not able to propagate information from child to the parent efficiently. Let v be a node with only two child nodes without common descendants:

$$\begin{aligned} P(h_v \,|\boldsymbol{\mathcal{X}}_v) &= \sum_{h_{v1}} \sum_{h_{v2}} P(h_v, h_{v1}, h_{v2} \mid x_v, \boldsymbol{\mathcal{X}}_{v1}, \boldsymbol{\mathcal{X}}_{v2}) \\ &= \sum_{h_{v1}} \sum_{h_{v2}} P(h_v \mid x_1, h_{v1}, h_{v2}) P(h_{v1} \mid \boldsymbol{\mathcal{X}}_{v1}) P(h_{v2} \mid \boldsymbol{\mathcal{X}}_{v2}) \\ &= \sum_{h_{v1}} \sum_{h_{v2}} \left[\sum_{l=1}^{L} P(S_v = l \mid x_v)(h_v \mid S_v = l, h_{vl}, x_v) \right] P(h_{v1} \mid \boldsymbol{\mathcal{X}}_{v1}) P(h_{v2} \mid \boldsymbol{\mathcal{X}}_{v2}) \\ &\neq \sum_{l=1}^{2} P(S_1 = l \mid x_v) P(h_1 \mid x_1, h_{1l}) P(h_{1l} \mid x_{1l}). \end{aligned} \tag{3.26}$$

The summation over the mixture variable cannot be distributed over the product of $P(h_1 \mid x_1, h_{1l})P(h_{1l} \mid x_{1l})$. Hence, we still need the full state-transition distribution $P(h_v \mid x_1, h_{v1}, h_{v2})$ to perform the correct information propagation. The full distribution can be reconstructed using the Eq. (3.23), losing the computational advantage of the approximation.

The information propagation from child variables to the parent variable is crucial for the learning procedure. In particular, it is useful for the computation of the β values in Eq. (3.6). As we show in Appendix D.1, the last equality of Eq. (3.26) is usually imposed in the learning procedure. This allows to avoid the exponential size of the state-transition tensor but removes information exchanges between the child variables. Hence, they become completely independent.

3.3.2 First-Order Approximation

First-order RecNNs are clearly an approximation of RecNTNs. Nevertheless, it is interesting to show the connection between these two class of models from the point of view of the tensor $\underline{\boldsymbol{T}}$ which parametrises the RecNTN neural state-transition layer. In particular, we show that first-order RecNNs can be obtained imposing a zero-constraint on the entries of $\underline{\boldsymbol{T}}$.

Recalling the definition of the first-order state-transition function:

$$\boldsymbol{h}_v = \sigma\left(\boldsymbol{W}\boldsymbol{x}_v + \sum_{l=1}^{L} \boldsymbol{U}_l \boldsymbol{h}_{vl} + \boldsymbol{b}\right), \tag{3.27}$$

the hidden state $\boldsymbol{h}_v$ is obtained by applying a linear transformation $\boldsymbol{W}$ to the input label $\boldsymbol{x}_v$ and a linear transformation $\boldsymbol{U}_l$ to each child hidden state $\boldsymbol{h}_{vl}$. The results of such transformations are summed together with a bias vector $\boldsymbol{b}$, and fed to a non-linear function σ.

Let ψ^+ be the summation of each linear map:

$$\psi^+(\boldsymbol{x}, \boldsymbol{h}_1, \ldots, \boldsymbol{h}_L) = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}_v + \boldsymbol{U}_1 \boldsymbol{h}_{v1} + \cdots + \boldsymbol{U}_L \boldsymbol{h}_{vL}, \tag{3.28}$$

ψ^+ is clearly a multi-affine map. Hence, according to the Theorem 1, it exists a tensor $\underline{\boldsymbol{T}}^+$ such that:

$$\psi^+(\boldsymbol{x}, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL}) = \sum_{i=1}^{M+1} \sum_{j_1=1}^{C+1} \cdots \sum_{j_L=1}^{C+1} \underline{\boldsymbol{T}}^+[i, j_1, \dots, j_L, :] \bar{\boldsymbol{x}}[i] \bar{\boldsymbol{h}}_{v1}[j_1] \dots \bar{\boldsymbol{h}}_{vL}[j_L]. \tag{3.29}$$

The key observation is that the entry $\underline{\boldsymbol{T}}^+[i, j_1, \dots, j_L, :]$ is multiplied by the elements $\bar{\boldsymbol{x}}[i], \bar{\boldsymbol{h}}_{v1}[j_1], \dots, \bar{\boldsymbol{h}}_{vL}[j_L]$. However, the function ψ^+ have no terms where vectors are multiplied together. Hence, all those entries of $\underline{\boldsymbol{T}}^+$ which are multiplied by more than one vector are set to 0.

Recalling that we are using homogeneous coordinates (hence the last element of each input vector is equal to 1), the only non-zero entries in $\underline{\boldsymbol{T}}^+$ are the ones which have all indexes (except one) pointing to the last entry. For example, the entry $\underline{\boldsymbol{T}}^+(i, C{+}1, \dots, C{+}1, :)$ multiplies solely by $\boldsymbol{x}[i]$, since $\bar{\boldsymbol{h}}_{v1}[C{+}1] = \cdots = \bar{\boldsymbol{h}}_{vL}[C{+}1] = 1$. Therefore, we obtain the following equation:

$$\begin{aligned} \boldsymbol{h}_v = &\sum_{i=1}^{M} \underline{\boldsymbol{T}}^+[i, C+1, \dots, C+1, :] \boldsymbol{x}[i] + \sum_{j_1=1}^{C} \underline{\boldsymbol{T}}^+[M+1, j_1, \dots, C+1, :] \boldsymbol{h}_{v1}[j_1] \\ &+ \cdots + \sum_{j_L=1}^{C} \underline{\boldsymbol{T}}^+[M+1, C+1, \dots, j_L, :] \boldsymbol{h}_{vL}[j_L] + \underline{\boldsymbol{T}}^+[M+1, C+1, \dots, C+1, :], \end{aligned} \tag{3.30}$$

where each term (except the last one) is the application of a linear map. Finally, by comparing Eq. (3.29) with Eq. (3.30), we can derive the entries of $\underline{\boldsymbol{T}}^+$:

$$\underline{\boldsymbol{T}}^+[i, j_1, \dots, j_L, :] = \begin{cases} \boldsymbol{W}[i, :] & \text{if } j_1 = \cdots = j_L = C+1 \\ \boldsymbol{U}_l[j_l, :] & \text{if } i = M+1 \wedge j_{\neq l} = C+1 \\ \boldsymbol{b}[:] & \text{if } i = M+1 \wedge j_1 = \cdots = j_L = C+1 \\ 0 & \text{otherwise} \end{cases}. \tag{3.31}$$

This approximation requires $MC + LC^2 + C = O(LC^2)$ parameters. The exponential dependence between the number of parameters and the hidden state size is removed annihilating the higher-order interactions. Nevertheless, in this formulation, the input label and the child hidden states contribute separately to the computation of the hidden state h_v; therefore, each of them is completely independent of the others. This induces a strong inductive bias into the model.

In the rest of the work, we refer to this model as Sum-RecNTN to underline the key role of the summation in the aggregation of structural context.

3.4 Conclusion

In this chapter, we have introduced a tensor framework for learning with structured data. The proposed framework is based on the definition of a probabilistic and a

neural recursive model whose state-transition function is parametrised by a tensor.

In particular, HRTM defines a categorical state-transition distribution which is parametrised by a tensor. Such a distribution allows capturing all the possible interactions among the input label and the structural context of a node. To this end, we have shown that HRTM has a low inductive bias since it imposes only the independence assumptions derived from modelling the learning task as recursive transduction.

RecNTN aggregates the input label and the structural context of a node by means of a tensor-based multi-affine map. The tensor parametrisation allows considering higher-order interactions during the aggregation. Then, we have argued that RecNTNs has a low inductive bias since it is able to simulate any neural recursive model whose state-transition function is based on a multi-affine map using the same hidden state size.

Finally, we have introduced two approximations commonly used in literature: the SP approximation for HRTMs and first-order approximation for RecNTNs. While they have different interpretations, these two approximations share the following two characteristics: (1) they impose a complete independence assumption among input label and child nodes contributions and (2) they cannot be interpreted as a tensor decomposition of the tensor parameter.

Chapter 4

Tensor Decompositions for Recursive Tensor Models

4.1 Introduction

This chapter puts forward the application of tensor decompositions on full-tensor models to reduce the number of parameters required by tensor state-transition functions. To this end, we propose novel recursive models which use tensor decompositions to represent tensors parameters in a compressed format. Hence, decomposition factors become the new model parameters which are learned from data. This approach has been already used in the literature to compress neural layers (e.g. [105, 125, 23, 163]). Nevertheless, in these works tensor decompositions are used as mere tools to reduce the number of parameters required by neural layers. A key point of our work is to show the connection between tensor decompositions and models inductive bias.

We argue that each decomposition introduces a different inductive bias to the model, which can be exploited during the learning. In the case of probabilistic models, tensor decompositions are interpreted as BNs. This leads to the introduction of new variables and new independence assumptions which factorise the full state-transition distribution. In the case of neural models, tensor decompositions define a specific procedure to compute the state-transition output. Such a procedure is defined by the tensor network associated with each tensor decomposition.

Moreover, we show that the expressiveness of these novel models depends on decompositions ranks. From this perspective, the decompositions rank can be interpreted as a model hyper-parameter which regulates the complexity of the state-transition function. Thus, tensor decompositions allow decoupling model expressiveness from the dimension of the hidden encoding space. While the dimension of the hidden encoding space reflects the complexity of the input structures, the value of the rank reflects the complexity of the state-transition function.

We experimentally assess the effectiveness of the tensor decompositions in recursive models for structured data on two structural transduction tasks. In particular, we compare our proposal with approximated models existing in the literature, showing the advantages of the inductive biases introduced by tensor decompositions. Moreover,

we also analyse the computational complexity required by the models which leverage tensor decompositions.

In Section 4.2.2, we show the application of the CP decomposition to define a probabilistic and a neural model which approximate HRTM and RecNTN, respectively. In Section 4.2.3, we leverage the HOSVD to define a probabilistic and a neural approximated models. In Section 4.2.4, we apply the TT approximation for the same purpose. In Section 4.4, we discuss our experimental setting and the results obtained. Finally, in Section 4.5, we summarise the main results.

4.2 Approximated Recursive Tensor Models

In this section, we show how tensor decompositions can be used to approximate full-tensor models introduced in Section 3.2.

4.2.1 Tensor Decompositions and Model Approximations

The definition of the tensor-based models HRTM and RecNTN paves the way to the application of tensor decompositions on their tensor parameters, representing them in a compressed format. By imposing such a succinct representation, we obtain new recursive models having decomposition factors as parameters. The decomposition rank becomes a model hyper-parameter which regulates the trade-off between compression and model expressiveness.

In probabilistic models, tensor decompositions provide a compressed representation of the tensor which parametrises the state-transition distribution. Imposing a positivity constraint on the decomposition factors (i.e. they contain only positive entries), tensor decompositions can be interpreted as graphical models [142]. Thus, the compressed format becomes a factorisation of the distribution $P(h_v \mid x_v, h_{v1}, \ldots, h_{vL})$. This observation is the key to depict the inductive bias introduced by each tensor decomposition.

In neural models, the tensor decompositions compress the augmented tensor which parametrises the multi-affine map underlying the neural tensor state-transition function. The compression induces the definition of a new multi-affine map. From this point of view, tensor decompositions modify the process which aggregates input label and contextual information to obtain the node hidden state. Thus, they introduce a specific bias in the recursive model.

In the following sections, we apply these general concepts to define new recursive models. These models leverage the tensor decompositions introduced in Section 2.4.3 to approximate the state-transition functions. For the sake of conciseness, we report only the approximated state-transition function. Moreover, in their definition, we ignore the input label. As we have shown in Section 3.2, the input label can be used to select a specific parametrisation of the state-transition function. Thus, also in the case of approximated models, we use the input label to select a different parametrisation of the tensor approximation.

Also, we do not detail learning algorithms. All probabilistic models can be trained by minor modifications of the EM algorithm derived for HRTM. We detail these specialised procedures in Appendix D. Neural models are trained using gradient descent methods since the tensor decompositions are used to represent multi-affine (thus differentiable) functions.

4.2.2 Canonical Approximation

Let $\underline{\boldsymbol{T}}$ be a $(L+1)$-way tensor which parametrises a state-transition function. Recalling the definition of the CP decomposition in Eq. (2.32), we can approximate $\underline{\boldsymbol{T}}$ as:

$$\underline{\boldsymbol{T}}[j_1, \ldots, j_L, k] \approx \sum_{r=1}^{R} \boldsymbol{U}_1[j_1, r] \ldots \boldsymbol{U}_L[j_L, r] \boldsymbol{Q}[k, r], \tag{4.1}$$

where $\{\boldsymbol{U}_1, \ldots, \boldsymbol{U}_L\}$ are the factor matrices associated to the first L dimensions, while the factor matrix $\boldsymbol{Q}$ is associated to the last dimension. All factor matrices have size $(C+1) \times R$, where R is the rank of the approximation. Thus, the number of parameters required by this approximation is $O(CLR)$.

The general idea of the CP approximation is to aggregate the child contributions with a simple operation like the element-wise multiplication. In principle, this multiplication does not allow modelling higher-order interactions among children (i.e. the contribution of each child nodes does not depend on its siblings). Nevertheless, the CP approximation is able to capture complex interactions because it performs the aggregation in a new R-dimensional space. Since the size of this new space can be much larger than the original one, a simple operation is enough to model such relations. Thus, the value of R (i.e. the decomposition rank) regulates models expressiveness. Larger values of R allow defining state-transition functions that capture complex interactions among child nodes even if they are aggregated by element-wise multiplication. On the other hand, larger values of R lead to state-transition functions that require more parameters.

Probabilistic Interpretation (CP-HRTM)

Let $\underline{\boldsymbol{P}}$ be the tensor which parametrises the HRTM state-transition distribution. If we assume it is decomposed according to the CP approximation in Eq. (4.1), we obtain the following factorisation of the state-transition distribution:

$$P(h_v \mid h_{v1}, \ldots, h_{vL}, \underline{\boldsymbol{P}}) \approx \sum_{r_v} P(h_v \mid r_v, \boldsymbol{Q}) \prod_{l=1}^{L} P(r_v \mid h_{vl}, \boldsymbol{U}_l), \tag{4.2}$$

where we make explicit the relation between distributions and factor matrices. Since factor matrices parametrise categorical distributions, they must contain only non-negative elements. The value of r_v represents the state of a new discrete random variable R_v which has exactly R states. In Figure 4.1a, we show the BN associated with Equation (4.2).

The state r_v can be interpreted as an encoding of the joint configuration of the child hidden states $(h_{v1}, \ldots, h_{vL})$. If the number of states R is less than the number of possible joint configurations, there are configurations mapped to the same state r_v. Joint configurations mapped to the same state r_v are indistinguishable for the parent variable H_v. Suppose the value of R is big enough to build a one-to-one relationship between r_v and all the child joint configurations. In this case, the expressiveness of this approximation is equivalent to the HRTM. It is enough to set each distribution $P(r_v \mid h_{vl})$ equal to 1, if the joint configuration associated with r_v contains h_{vl}, and equal to 0 otherwise. By computing the product $\prod_{l=1}^{L} P(r_v \mid h_{vl})$, we obtain a distribution a $P(r_v \mid h_{v1}, \ldots, h_{vL})$ which is always zero except in the state r_v associated with the joint configuration $(h_{v1}, \ldots, h_{vL})$. Thus, $\sum_{r_v=1} P(h_v \mid r_v) \prod_{l=1}^{L} P(r_v \mid h_{vl})$ is equivalent to $P(h_v \mid h_{v1}, \ldots, h_{vL})$. The price to pay for such expressiveness is a number of parameters which again grows exponentially with respect to L. In fact, we must set $R = C^L$ to build a one-to-one relationship between r_v and all the child joint configurations. This is consistent with the upper bound of the tensor rank (see Section 2.4.3).

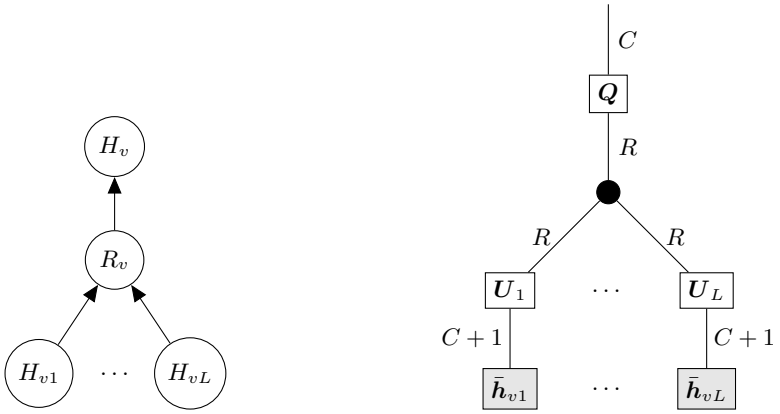
This interpretation of the rank variable gives an intuition of the approximation's inductive bias, reflecting the independence assumptions imposed among variables. The rank variable R_v blocks all paths between child variables $H_{v1}, \ldots, H_{vL}$ and the parent variable H_v (see Figure 4.1a); hence, $H_v \perp\!\!\!\perp \{H_{v1}, \ldots, H_{vL}\} \mid R_v$. Moreover, child variables are independent given R_v. Unfortunately, this conditional independence cannot be deduced from the BN. Hence, we show that it holds analytically. For the sake of simplicity, we show it holds only in the case of two child variables:

$$\begin{aligned} P(h_{v1}, h_{v2} \mid r_v) &\propto P(h_{v1}, h_{v2}, r_v) \\ &\propto P(r_v \mid h_{v1}) P(r_v \mid h_{v2}) P(h_{v1}) P(h_{v1}) \\ &= P(h_{v1}, r_v) P(h_{v2}, r_v) \propto P(h_{v1} \mid r_v) P(h_{v2} \mid r_v). \end{aligned} \tag{4.3}$$

The generalisation to the case of a node with L child nodes is straightforward. This assumption ensures that the contribution of each child variable to the state of R_v is independent of the others. Note that this independence assumption also guarantees that the information propagation from the child variables to the parent variable does not require the full distribution (as in the case of the SP-HRTM, see Section 3.3.1). Thus, the propagation can be carried out efficiently.

Neural Interpretation (CP-RecNTN)

Let $\underline{\boldsymbol{T}}$ be the augmented tensor which parametrises the multi-affine map of RecNTN state-transition function. If we assume it is decomposed according to the CP



(A) CP-HRTM state-transition distribution. (B) CP-RecNTN multi-affine map.

FIGURE 4.1: Graphical representation of the probabilistic and the neural CP state-transition function.

approximation in Eq. (4.1), we can approximate the multi-affine map as:

$$
\begin{aligned}
\underline{\boldsymbol{T}}(\bar{\boldsymbol{h}}_{v1},\dots,\bar{\boldsymbol{h}}_{vL}) &= \sum_{j_1=1}^{C+1}\cdots\sum_{j_L=1}^{C+1}\underline{\boldsymbol{T}}[j_1,\dots,j_L,:]\times\bar{\boldsymbol{h}}_{v1}[j_1]\dots\bar{\boldsymbol{h}}_{vL}[j_L] \\
&\approx \sum_{j_1=1}^{C+1}\cdots\sum_{j_L=1}^{C+1}\sum_{r=1}^{R}\boldsymbol{U}_1[j_1,r]\dots\boldsymbol{U}_L[j_L,r]\boldsymbol{Q}[:,r]\bar{\boldsymbol{h}}_{v1}[j_1]\dots\bar{\boldsymbol{h}}_{vL}[j_L] \\
&= \sum_{r=1}^{R}\left(\left(\sum_{j_1=1}^{C+1}\boldsymbol{U}_1[j_1,r]\bar{\boldsymbol{h}}_{v1}[j_1]\right)\dots\left(\sum_{j_L=1}^{C+1}\boldsymbol{U}_L[j_L,r]\bar{\boldsymbol{h}}_{vL}[j_L]\right)\boldsymbol{Q}[r,:]\right) \\
&= \sum_{r=1}^{R}\left((\boldsymbol{U}_1^\intercal\bar{\boldsymbol{h}}_{v1})[r]\dots(\boldsymbol{U}_L^\intercal\bar{\boldsymbol{h}}_{vL})[r]\right)\boldsymbol{Q}[:,r] \\
&= \boldsymbol{Q}\left(\boldsymbol{U}_1^\intercal\bar{\boldsymbol{h}}_{v1}\odot\cdots\odot\boldsymbol{U}_L^\intercal\bar{\boldsymbol{h}}_{vL}\right).
\end{aligned}
\tag{4.4}
$$

The computation of the multi-affine map in Eq. (4.4) follows the intuition of the CP approximation (see the tensor network in Figure 4.1b). At first, factor matrices $\{\boldsymbol{U}_1,\dots,\boldsymbol{U}_L\}$ are used to map child hidden states in vectors in $\mathbb{R}^R$. Then, these new vectors are aggregated together by the element-wise multiplication. Finally, the result of the aggregation is mapped back to the hidden state space $\mathbb{R}^C$ thanks to the factor matrix $\boldsymbol{Q}$. The independence relationship among child contributions is given by the element-wise multiplication.

4.2.3 Higher-Order Singular Value Decomposition Approximation

Let $\underline{\boldsymbol{T}}$ be a $(L+1)$-way tensor which parametrises a state-transition function. Recalling the definition of the HOSVD in Eq. (2.33), we can approximate $\underline{\boldsymbol{T}}$ as:

$$\begin{aligned}&\underline{\boldsymbol{T}}[h_{v1},\dots,h_{vL},h_v]\\&\approx\sum_{r_v=1}^{R}\sum_{r_{v1}=1}^{R}\cdots\sum_{r_{vL}=1}^{R}\underline{\boldsymbol{G}}[r_{v1},\dots,r_{vL},r_v]\boldsymbol{U}_1[h_{v1},r_{v1}]\dots\boldsymbol{U}_L[h_{vL},r_{vL}]\boldsymbol{Q}[h_v,r_v],\end{aligned}\tag{4.5}$$

where R is the rank of the approximation along all dimensions of $\underline{\boldsymbol{T}}, \boldsymbol{U}_1,\dots,\boldsymbol{U}_L$ are the mode matrices associated to the first L dimensions and $\boldsymbol{Q}$ is the mode matrix along the last dimension. The tensor $\underline{\boldsymbol{G}}$ is the core tensor of the approximation. Each factor matrix requires $O(CR)$ parameters, while the core tensor requires $O(R^{L+1})$ parameters. Thus, the total number of parameters required by the HOSVD approximation is $O(LCR+R^{L+1})=O(R^{L+1})$.

The HOSVD approximation explicitly models higher-order interactions among child hidden states thanks to the core tensor. Nevertheless, these interactions are computed on a succinct representations of the child hidden states. The value of R (i.e. the decomposition rank) indicates the size of these succinct representations. Thus, small values of R lead to the definition of state-transition functions that heavily compress child information. Nevertheless, the compression reduces the ability to model complex interactions among constituents.

Probabilistic Interpretation (HOSVD-HRTM)

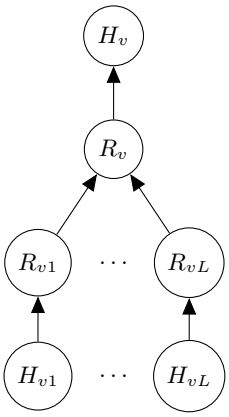
Let $\underline{\boldsymbol{P}}$ be the tensor which parametrises the HRTM state-transition distribution. By applying the HOSVD approximation in Eq. (4.5), we obtain the following approximation of the state-transition distribution:

$$\begin{aligned}&P(h_v\mid h_{v1},\dots,h_{vL},\underline{\boldsymbol{P}})\\&\quad\approx\sum_{r_v}P(h_v\mid r_v,\boldsymbol{Q})\sum_{r_{v1}}\cdots\sum_{r_{vL}}P(r_v\mid r_{v1},\dots,r_{vL},\underline{\boldsymbol{G}})\prod_{l=1}^{L}P(r_{vl}\mid h_{vl},\boldsymbol{U}_l),\end{aligned}\tag{4.6}$$

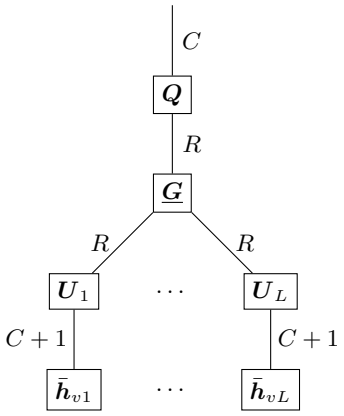
where we make explicit the relation between distributions and approximation factors. Since the mode matrices and the core tensor are interpreted as parameters of categorical distributions, they must contain only non-negative entries.

The values $r_v, r_{v1},\dots,r_{vL}$ represent the realisations of new random variables $R_v, R_{v1},\dots,R_{vL}$. Each of these new random variables has R states. In Figure 4.2a, we show the BN associated with Equation (4.6).

The rank variables $\{R_{v1},\dots,R_{vL}\}$ are the core of the probabilistic interpretation of the HOSVD approximation. In fact, the states of each R_{vl} can be interpreted as clusters of the child hidden states [176, 146, 145]. The hidden states in the same cluster are indistinguishable from the state-transition point of view since the parent state h_v depends only on the cluster (i.e. the state of R_{vl}). Hence, each cluster r_{vl}



(A) HOSVD-HRTM state-transition distribution.



(B) HOSVD-RecNTN multi-affine map.

FIGURE 4.2: Graphical representation of the probabilistic and the neural HOSVD state-transition function.

contains all the states of H_{vl} that bring the same information to the parent state transition. This behaviour is consistent with the independence assumptions introduced by the approximation. Observing the Bayesian network in Figure 4.2a, it is clear that H_{vl} and H_v are independent given R_{vl}, i.e. $H_{vl} \perp\!\!\!\perp H_v \mid R_{vl}$.

If all the states of each random variable H_{vl} are informative, no clustering can be performed; thus, the number of clusters (i.e. the number of states of the random variable R_{vl}) must be equal to the number of states of H_{vl}. Hence, no compression is performed. This is consistent with the upper bound of the n-mode rank (see Section 2.4.3).

Neural Interpretation (HOSVD-RecNTN)

Let $\underline{\boldsymbol{T}}$ be the augmented tensor which parametrises the multi-affine map of RecNTN state-transition function. If we assume it is decomposed according to the HOSVD

approximation in Eq. (4.5), we can approximate the multi-affine map as:

$$
\begin{aligned}
\underline{\boldsymbol{T}}(\bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{vL}) &= \sum_{j_1=1}^{C+1} \cdots \sum_{j_L=1}^{C+1} \underline{\boldsymbol{T}}[j_1, \dots, j_L, :] \times \bar{\boldsymbol{h}}_{v1}[j_1] \dots \bar{\boldsymbol{h}}_{vL}[j_L] \\
&\approx \sum_{j_1=1}^{C+1} \cdots \sum_{j_L=1}^{C+1} \sum_{r_1=1}^{R} \cdots \sum_{r_L=1}^{R} \sum_{r=1}^{R} \underline{\boldsymbol{G}}[r_1, \dots, r_L, r] \boldsymbol{U}_1[j_1, r_1] \dots \boldsymbol{U}_L[j_L, r_L] \\
&\quad \times \boldsymbol{Q}[:, r] \bar{\boldsymbol{h}}_{v1}[j_1] \dots \bar{\boldsymbol{h}}_{vL}[j_L] \\
&= \sum_{r_1=1}^{R} \cdots \sum_{r_L=1}^{R} \sum_{r=1}^{R} \underline{\boldsymbol{G}}[r_1, \dots, r_L, r] \\
&\quad \times \left(\sum_{j_1=1}^{C+1} \boldsymbol{U}_1[j_1, r_1] \bar{\boldsymbol{h}}_{v1}[j_1] \right) \dots \left(\sum_{j_L=1}^{C+1} \boldsymbol{U}_L[j_L, r_L] \bar{\boldsymbol{h}}_{vL}[j_L] \right) \boldsymbol{Q}[:, r] \\
&= \sum_{r_1=1}^{R} \cdots \sum_{r_L=1}^{R} \sum_{r=1}^{R} \underline{\boldsymbol{G}}[r_1, \dots, r_L, r] (\boldsymbol{U}_1^\intercal \bar{\boldsymbol{h}}_{v1})[r_1] \dots (\boldsymbol{U}_L^\intercal \bar{\boldsymbol{h}}_{vL})[r_L] \boldsymbol{Q}[:, r] \\
&= \sum_{r=1}^{R} \boldsymbol{Q}[:, r] \underline{\boldsymbol{G}} \left(\boldsymbol{U}_1^\intercal \bar{\boldsymbol{h}}_{v1}, \dots, \boldsymbol{U}_L^\intercal \bar{\boldsymbol{h}}_{vL} \right) [r] \\
&= \boldsymbol{Q} \left(\underline{\boldsymbol{G}} \left(\boldsymbol{U}_1^\intercal \bar{\boldsymbol{h}}_{v1}, \dots, \boldsymbol{U}_L^\intercal \bar{\boldsymbol{h}}_{vL} \right) \right).
\end{aligned}
\tag{4.7}
$$

The computation of the multi-affine map in Eq. (4.7) reflects the intuition of the HOSVD approximation (see Figure 4.2b). At first, factor matrices $\boldsymbol{U}_1, \dots, \boldsymbol{U}_L$ are used to compute a R-dimensional succinct representation of the child hidden states. Then, these new vectors are combined using the core tensor $\underline{\boldsymbol{G}}$. Finally, the result of the aggregation is mapped back to the hidden state space $\mathbb{R}^C$ thanks to the factor matrix $\boldsymbol{Q}$. Higher-order interactions are captured thanks to the core tensor $\underline{\boldsymbol{G}}$.

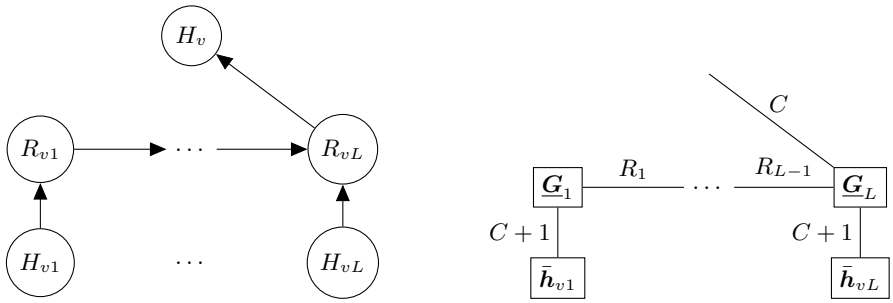
4.2.4 Tensor Train Approximation

Let $\underline{\boldsymbol{T}}$ be a $(L+1)$-way tensor which parametrises a state-transition function. Recalling the definition of the TT in Eq. (2.34), we can approximate $\underline{\boldsymbol{T}}$ as:

$$
\begin{aligned}
&\underline{\boldsymbol{P}}[h_{v1}, \dots, h_{vL}, h_v] \\
&\qquad \approx \sum_{r_{v1}=1}^{R} \cdots \sum_{r_{vL}=1}^{R} \boldsymbol{G}_1[h_{v1}, r_{v1}] \dots \underline{\boldsymbol{G}}_L[r_{vL-1}, h_{vL}, r_{vL}] \boldsymbol{Q}[r_{vL}, h_v],
\end{aligned}
\tag{4.8}
$$

where R is the approximation rank which we assume equal along all dimensions, and $\boldsymbol{G}_1, \dots, \underline{\boldsymbol{G}}_L, \boldsymbol{Q}$ are the core tensors. All the core tensors (except the first and the last one), have size $R \times (C+1) \times R$. Thus, the number of parameters required by the approximation is $O(LCR^2)$.

The idea of the TT approximation is to aggregate child hidden states iteratively, considering one child at a time. At each step, the child information is combined with the result of the previous aggregation, obtaining the new intermediate result. All temporary results are stored in vectors of size R (i.e. the decomposition rank). By



(A) TT-HRTM state-transition distribution. (B) TT-RecNTN multi-affine map.

FIGURE 4.3: Graphical representation of the probabilistic and the neural TT state-transition function.

increasing the value of R, we increase the capacity of the approximation to store intermediate results. Thus, larger values of R lead to more expressive state-transition functions.

Probabilistic Interpretation (TT-HRTM)

Let $\underline{\boldsymbol{P}}$ be the tensor which parametrises HRTM state-transition distribution, its TT approximation following Eq. (4.8) yields:

$$P(h_v \mid h_{v1}, \ldots, h_{vL}, \underline{\boldsymbol{P}}) \approx \sum_{r_{v1}} \cdots \sum_{r_{vL}} P(r_{v1} \mid h_{v1}, \boldsymbol{G}_1) \ldots P(r_{vL} \mid r_{vL-1}, h_{vL}, \underline{\boldsymbol{G}}_L) P(h_v \mid r_{vL} \boldsymbol{Q}), \quad (4.9)$$

where we make explicit the relation between distributions and core tensors of the approximation. The values $r_v, r_{v1}, \ldots, r_{vL-1}$ represent the states of the new discrete random variables $R_v, R_{v1}, \ldots, R_{vL}$. Each of them has R states. In Figure 4.3a, we show the BN associated with Equation (4.9). The BN depicted is very similar to the BN of a HMM. Hence, rank variables can be interpreted as the hidden process which regulates the interaction among hidden child state variables. The Markovian property is ensured by the conditional independence $R_{vl+1} \perp\!\!\!\perp R_{vl-1} \mid R_{vl}$. In fact, as in HMMs, this independence assumption ensures us that the state of R_{vl} encodes all the useful information of previous elements (i.e. $\{H_1, \ldots, H_l\}$). Moreover, each variable R_{vl} is a collider in all paths between H_l and $H_{l'}$, where $l' < l$. Thus, if we observe the rank variable R_{vl}, the variable H_l is not independent of all variables $\{H_1, \ldots, H_{vl-1}\}$. Note that if the state R_{vL} (or H_v) is observed, all hidden child state variables $\{H_1, \ldots, H_{vL}\}$ are dependent.

Neural Interpretation (TT-RecNTN)

Let $\underline{\boldsymbol{T}}$ be the augmented tensor which parametrises the multi-affine map of RecNTN state-transition function, its TT approximation is

$$\begin{aligned}
\underline{\boldsymbol{T}}(\bar{\boldsymbol{h}}_{v1},\dots,\bar{\boldsymbol{h}}_{vL}) &= \sum_{j_1=1}^{C+1}\cdots\sum_{j_L=1}^{C+1}\underline{\boldsymbol{T}}[j_1,\dots,j_L,:]\times\bar{\boldsymbol{h}}_{v1}[j_1]\dots\bar{\boldsymbol{h}}_{vL}[j_L]\\
&\approx \sum_{j_1=1}^{C+1}\cdots\sum_{j_L=1}^{C+1}\sum_{r_1=1}^{R}\cdots\sum_{r_L=1}^{R}\boldsymbol{G}_1[j_1,r_1]\dots\underline{\boldsymbol{G}}_L[r_{L-1},j_L,r_L]\times\boldsymbol{Q}[r_L,:]\\
&\quad\times\bar{\boldsymbol{h}}_{v1}[j_1]\dots\bar{\boldsymbol{h}}_{vL}[j_L]\\
&= \sum_{r=1}^{R}\sum_{r_L=1}^{R}\cdots\sum_{r_2=1}^{R}\left(\sum_{r_1=1}^{R}\left(\sum_{j_1=1}^{C+1}\boldsymbol{G}_1[j_1,r_1]\bar{\boldsymbol{h}}_{v1}[j_1]\right)\sum_{j_2=1}^{C+1}\underline{\boldsymbol{G}}_2[r_1,j_2,r_2]\bar{\boldsymbol{h}}_{v2}[j_2]\right)\\
&\quad\dots\left(\sum_{j_L=1}^{C+1}\underline{\boldsymbol{G}}_L[r_{L-1},j_L,r_L]\bar{\boldsymbol{h}}_{vL}[j_L]\right)\boldsymbol{Q}[r,:]\\
&= \sum_{r=1}^{R}\boldsymbol{Q}[r,:]\underline{\boldsymbol{G}}_L\left(\dots\underline{\boldsymbol{G}}_2\left(\boldsymbol{G}_1^{\mathsf{T}}\bar{\boldsymbol{h}}_{v1},\bar{\boldsymbol{h}}_{v2}\right)\dots,\bar{\boldsymbol{h}}_{vL}\right)[r]\\
&= \boldsymbol{Q}^{\mathsf{T}}\left(\underline{\boldsymbol{G}}_L\left(\dots\underline{\boldsymbol{G}}_2\left(\boldsymbol{G}_1^{\mathsf{T}}\bar{\boldsymbol{h}}_{v1},\bar{\boldsymbol{h}}_{v2}\right)\dots,\bar{\boldsymbol{h}}_{vL}\right)\right).
\end{aligned} \tag{4.10}$$

The computation of the multi-affine map in Eq. (4.10) follows the intuition of the TT approximation (see Figure 4.3b). At the first step, the first hidden state vector $\bar{\boldsymbol{h}}_{v1}$ is mapped into a R-dimensional vector thanks to core matrix $\boldsymbol{G}_1$. This vector represents the first intermediate result. At the second step, such a result is combined with the second hidden state vector $\bar{\boldsymbol{h}}_{v2}$ through the multi-linear function $\underline{\boldsymbol{G}}_2$ which yields the new R-dimensional intermediate result. Hence, each core tensor can be interpreted as a linear operator which combines the previous intermediate result with the current child hidden state, obtaining the new intermediate result. The last core tensor $\boldsymbol{Q}$ maps the final result to the hidden state space.

4.3 Approximated LSTM-based Recursive Models

In Section 2.3.3, we have introduced Tree-LSTMs. They are a specific architecture of RecNN that use a LSTM cell to approximate the state-transition function. The computation of all the LSTM gates values is performed through different neural networks whose architectures are identical to the first-order RecNN hidden layer. Hence, we can define tensor LSTM cells by applying tensor-based multi-affine maps

to compute the gates values:

$$\begin{aligned}
\boldsymbol{i}_v &= \sigma\left(\psi^i(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL})\right), & \boldsymbol{o}_v &= \sigma\left(\psi^o(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL})\right), \\
\boldsymbol{u}_v &= \sigma\left(\psi^u(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL})\right), & \boldsymbol{f}_{vl} &= \sigma\left(\psi^l(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL})\right), \forall l \in [1, L], \\
\boldsymbol{c}_v &= \boldsymbol{i}_v \odot \boldsymbol{u}_v + \sum_{l=1}^{L} \boldsymbol{f}_{vl} \odot \boldsymbol{c}_{vl}, & \boldsymbol{h}_v &= \boldsymbol{o}_v \odot \tanh(\boldsymbol{c}_v),
\end{aligned} \tag{4.11}$$

where $\{\psi^i, \psi^o, \psi^u\}$ and $\{\psi^1, \dots, \psi^L\}$ are multi-affine maps. In the following paragraphs, we introduce new tensor Tree-LSTMs defining multi-affine maps based on the tensor framework developed in this thesis. Note that we use the same architecture for all ψ functions. Hence, in the following paragraphs we define a single function ψ^t where $t \in \{i, o, u, 1, \dots, L\}$ is the superscript which specifies the gate. We apply the same superscript also on the multi-affine parameters, to underline that each multi-affine map has a different parametrisation.

As in the other approximated models, we assume that input labels are categorical and their values specify a different parametrisation of the state-transition function. Hence, each ψ^t has a different parametrisation for each input label. For the sake of simplicity, in the following equations we ignore this relation between input label and parameters.

Full-TensorLSTM. In the Full-LSTM, each multi-affine map ψ^t is implemented through an augmented tensor:

$$\psi^t(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL}) = \underline{\boldsymbol{T}}^t(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{vL}). \tag{4.12}$$

Sum-TensorLSTM. The Sum-LSTM is equivalent to Tree-LSTM defined in Section 2.3.3. Each multi-affine map ψ^t is implemented considering only first-order interaction among hidden child states:

$$\psi^t(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL}) = \sum_{l=1}^{L} \boldsymbol{U}_l^t \boldsymbol{h}_{vl} + \boldsymbol{b}^t. \tag{4.13}$$

CP-TensorLSTM. In the CP-LSTM, each multi-affine map ψ^t is obtained by applying the CP approximation (see Eq. (4.1)) on its augmented tensor:

$$\psi^t(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL}) = \boldsymbol{Q}^t \left({\boldsymbol{U}_1^t}^\mathsf{T} \bar{\boldsymbol{h}}_{v1} \odot \cdots \odot {\boldsymbol{U}_L^t}^\mathsf{T} \bar{\boldsymbol{h}}_{vL} \right) \tag{4.14}$$

HOSVD-TensorLSTM. In the HOSVD-LSTM, each multi-affine map ψ^t is obtained by applying the HOSVD approximation (see Eq. (4.5)) on its augmented tensor:

$$\psi^t(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL}) = \boldsymbol{Q}^t \left(\underline{\boldsymbol{G}}^t \left({\boldsymbol{U}_1^t}^\mathsf{T} \bar{\boldsymbol{h}}_{v1}, \dots, {\boldsymbol{U}_L^t}^\mathsf{T} \bar{\boldsymbol{h}}_{vL} \right) \right), \tag{4.15}$$

	Recursive Models		
Approx.	Probabilistic	One-layer NN	LSTM-based
None	Full-HRTM	Full-RecNTN	Full-LSTM
CP	CP-HRTM	CP-RecNTN	CP-LSTM
HOSVD	HOSVD-HRTM	HOSVD-RecNTN	HOSVD-LSTM
TT	TT-HRTM	TT-RecNTN	TT-LSTM
Existing	SP-HRTM [12]	Sum-RecNTN [58]	Sum-LSTM [159]

TABLE 4.1: List of all the models assessed.

TT-TensorLSTM. In the TT-LSTM, each multi-affine map ψ^t is obtained by applying the TT approximation (see Eq. (4.8)) on its augmented tensor:

$$\psi^t(\boldsymbol{x}_v, \boldsymbol{h}_{v1}, \dots, \boldsymbol{h}_{vL}) = \boldsymbol{Q}^{t\mathsf{T}} \left(\underline{\boldsymbol{G}}_L^t \left(\dots \underline{\boldsymbol{G}}_2^t \left(\boldsymbol{G}_1^{t\mathsf{T}} \bar{\boldsymbol{h}}_{v1}, \bar{\boldsymbol{h}}_{v2} \right) \dots, \bar{\boldsymbol{h}}_{vL} \right) \right). \tag{4.16}$$

4.4 Experimental Analysis

In this section, we experimentally assess the effectiveness of the proposed tensor-based models. In Table 4.1, we report all models we evaluated in the experimental assessment. For the sake of clarity, we refer to models which do not approximate the tensor parameter as *full* models. To ensure a fair comparison between different models, we always use a label-dependent parametrisation of the state-transition function.

Model predictive performance is assessed using classification accuracy. In particular, we use the term *root accuracy* to indicate the accuracy computed only on root nodes. This quantity measures the ability of the models to predict the correct class of the whole input structure (as in super-source transductions)

$$\text{ACC}_r = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}[y_r^n = \hat{y}_r^n], \tag{4.17}$$

where N is the number of output structures; y_r^n and $\hat{y}_r^n$ are the predicted and the true output labels attached to the root of the n-th structure, respectively.

On the other hand, we use the term *node accuracy* to indicate the accuracy computed on each node of the structure, i.e.:

$$\text{ACC} = \frac{1}{N} \sum_{n=1}^{N} \frac{\sum_{v \in \text{vert}(\boldsymbol{\mathcal{Y}})} \mathbb{I}[y_v^n = \hat{y}_v^n]}{|\text{vert}(\boldsymbol{\mathcal{Y}})|}, \tag{4.18}$$

where N is the number of output structures; y_v^n and $\hat{y}_v^n$ are the predicted and the true output labels attached to the node v of the n-th output structure, respectively. All the accuracy values reported in the next sections are averaged over three runs to account for randomisation effects due to model parameters initialisation.

4.4.1 Implementation Details

We have implemented all the models using PyTorch [131] and Deep Graph Library (DGL) [170]. The code is publicly available.[1]

Neural models implement the neural state-transition layer and the output layer as PyTorch modules. The recursive processing of the input structures leverages the DGL message-passing primitives. All the gradients are computed automatically using PyTorch differentiation tool.

We also implement probabilistic models using PyTorch and DGL; in particular, we use PyTorch to perform the numerical computations and DGL to implement the recursive processing of the input structures. All the probabilistic operations are performed in log-space to avoid numerical issues. However, when a marginalisation occurs, it is necessary to remove the logarithm. We implement the so-called *log-sum-exp* trick to avoid numerical issues in the marginalisation step:

$$\sum_x P(x,y) = Z \sum_x \frac{P(x,y)}{Z} = \log Z + \log \sum_x \exp[\log(P(x,y)) - \log Z],$$

where $Z = \max P(x,y) \implies \log Z = \max \log P(x,y)$. This ensures that the exponential function does not shrink to zero all values in $\log P(x,y)$ since $\log(P(x,y)) - \log Z$ always has an entry which is equal to 0.

4.4.2 Experimental Settings

We test all the models on two transduction tasks on tree-structured data, i.e. the input structured space is $\mathcal{X}^{\#^1_L}$. In the following, we outline the experimental setting used in both tasks by the probabilistic and the neural models. We assume that input and output labels have M and K different states, respectively.

Probabilistic Models Configurations

Probabilistic models tackle both tasks as structural transductions. Hence, an output label is generated for every node in the input structure. Thus, all output labels are visible during the training.

The generation of the output labels is handled through a categorical emission distribution since the output labels are categorical. The input labels on internal nodes are used to select a specific parametrisation of the state-transition distribution. On leaf nodes, the input labels are used to select a specific prior distribution: $P(h_v \mid x_v)$. Note that this prior distribution is equivalent to a full state-transition distribution where all the child hidden states are $\perp$.

Since the input structures are singly connected, probabilistic models are trained through tailored versions of the EM algorithm detailed in Appendix D. We execute the EM algorithm for a maximum of 200 iterations. Also, we use an early-stopping

[1] https://github.com/danielecastellana22/tensor-tree-nn

	Hyper-parameters values			
	BoolSent		ListOps	
	C	R	C	R
SP-HRTM	$\{5, 10, 20, 50\}$	-	$\{20, 50\}$	-
Full-HRTM	$\{2, 3\}$	-	$\{5, 10\}$	-
CP-HRTM	$\{5, 10\}$	$\{10, 20\}$	$\{20, 50, 100\}$	$\{20, 50, 100\}$
HOSVD-HRTM	$\{5, 10\}$	$\{2, 3\}$	$\{50, 100, 150\}$	$\{2, 3\}$
TT-HRTM	$\{5, 10\}$	$\{2, 4\}$	$\{10, 20, 30\}$	$\{5, 10\}$
Sum-RecNTN	$\{10, 20, 50, 100, 300\}$	-	$\{25, 88, 214\}$	-
Full-RecNTN	$\{2, 5, 10\}$	-	$\{3, 5, 7\}$	-
CP-RecNTN	$\{10, 20, 50\}$	$\{10, 50, 100\}$	$\{100\}$	$\{6, 50, 370\}$
HOSVD-RecNTN	$\{10, 20\}$	$\{2, 5, 10\}$	$\{10, 20\}$	$\{3, 5, 7\}$
TT-RecNTN	$\{10, 20\}$	$\{5, 10, 20\}$	$\{50\}$	$\{4, 12, 32\}$
Sum-LSTM	$\{10, 20, 50, 100, 300\}$	-	$\{25, 88, 214\}$	-
Full-LSTM	$\{2, 5, 10\}$	-	$\{3, 5, 7\}$	-
CP-LSTM	$\{10, 20, 50\}$	$\{10, 50, 100\}$	$\{100\}$	$\{6, 50, 370\}$
HOSVD-LSTM	$\{10, 20\}$	$\{2, 5, 10\}$	$\{10, 20\}$	$\{3, 5, 7\}$
TT-LSTM	$\{10, 20\}$	$\{5, 10, 20\}$	$\{50\}$	$\{4, 12, 32\}$

Table 4.2: Hyper-parameters values validated on the BoolSent and the ListOps task.

criteria on the validation root accuracy to stop the training if the root accuracy does not increase for 50 consecutive iterations.

For each model, we perform a model selection on the validation set to find the best hyper-parameters configuration. We validate only two hyper-parameters: the hidden-state size C and the rank R (if it is used). We choose to validate only these two hyper-parameters because they are strictly related to the model expressiveness and the number of parameters it requires. In Table 4.2, we report the hyper-parameters values validated in all the tasks.

Neural Models Configurations

Neural models (i.e. RecNTNs and LSTM-based models) tackle both tasks as structure classification problems. Hence, they implement a super-source transduction applying the output layer only on the root nodes. Thus, only root labels are visible during the training.

The output layer is implemented as a softmax layer, i.e. $\boldsymbol{y}_v = \sigma(\boldsymbol{W}^y \boldsymbol{h}_v + \boldsymbol{b}^y)$ where σ is the softmax activation function.

The input labels on the internal nodes are used to select a specific parametrisation of the state-transition neural layer. On leaf nodes, the state-transition function is not applied. Hence, we define a new layer to infer the hidden state of the leaf nodes starting from their input labels. The layer is defined as $\boldsymbol{h}_v = \sigma(\boldsymbol{W}^x \boldsymbol{x}_v + \boldsymbol{b}^x)$, where v is a leaf node, σ is the sigmoid activation function and $\boldsymbol{x}_v \in \mathbb{R}^M$ is the one-hot encoding of the input label. Note that this layer is equivalent to a full tensor neural layer where all child hidden states are zero vectors.

All the neural models are trained by minimising the negative log-likelihood of the observed labels. The parameters are updated using the AdaDelta algorithm [179]. Thus, no learning rate is set. The batch size is set to 50 for all the models. The training is performed for at most 100 epochs. Also, we use early-stopping if the root validation accuracy does not increase for 5 consecutive iterations.

For each model, we perform a model selection on the validation set to find the best hyper-parameters configuration. Again, we validate only two hyper-parameters: the hidden-state size C and the rank R (when used). In Table 4.2, we report the hyper-parameters values validated in all the tasks.

4.4.3 Boolean Sentences Task

We build the BoolSent task on purpose to assess performances of tensor-based models. The goal of BoolSent task is to predict the output of a sequence of operations on boolean values. Each sequence is represented with its parse tree generated according to the grammar:

$$\begin{aligned} \mathrm{S} := &\mathrm{OR}(S, \dots, S) \mid \mathrm{AND}(S, \dots, S) \mid \mathrm{XOR}(S, \dots, S) \\ &\mid \mathrm{IMPLY}(S, \dots, S) \mid \{0, 1\}, \end{aligned} \tag{4.19}$$

where $\{0, 1\}$ are the truth values which appear only on leaf nodes, and OR, AND, XOR, IMPLY are the logical operators which appear only on internal nodes. The result of the logical operators is obtained by folding the input list with the logical operator from left to right (i.e. the fold left reduction is applied). For example, the expression $\mathrm{OR}(0, 0, 1)$ is evaluated applying the OR operator on the first two input elements; the result obtained (i.e. 0 since $0 \vee 0 = 0$) is used to compute the final results: $0 \vee 1 = 1$. In the following, we briefly describe the properties of each operator.

OR: its output is 1 if and only if there is at least one true value among inputs.

AND: its output is 1 if and only if all the input values are true.

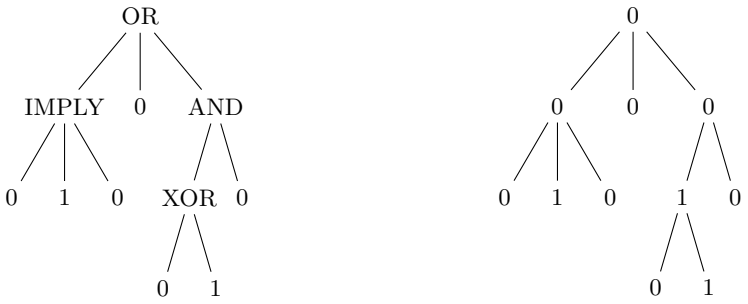
XOR: its output is 1 if and only if the number of true values in its input list is odd.

IMPLY: its output depends on the order of the input values; hence, cannot be deduced by counting the number of 0 and 1 in the input list.

In Figure 4.4, we report an example of input-output tree pair of the BoolSent task. Clearly, all the input labels are categorical. There are 4 possible labels on internal nodes and 2 possible labels on leaf nodes. The output labels are also categorical since there are only two possible outcomes: 0 and 1.

The tree maximum out-degree L denotes the number of inputs for each operator: we build four different dataset setting $L \in \{2, 3, 4, 5\}$. Each dataset contains 10k trees: 7000 in the training set, 1000 in the validation set and 2000 in the test set. The depth of each tree is between 4 and 8. The BoolSent datasets is publicly available.[2]

[2] https://github.com/danielecastellana22/tensor-tree-nn/tree/main/data/BoolSent/raw



(A) Input tree $\boldsymbol{\mathcal{X}}$. (B) Output tree $\boldsymbol{\mathcal{Y}}$.

FIGURE 4.4: Example of a BoolSent input-output tree pair.

	ACC_r%			
	L=2	L=3	L=4	L=5
SP-HRTM	73.8 (0.7)	69.8 (0.2)	69.0 (0.0)	70.4 (0.9)
Full-HRTM	**100.0** (0.0)	**100.0** (0.0)	**100.0** (0.0)	**100.0** (0.0)
CP-HRTM	60.6 (0.0)	63.9 (0.1)	66.9 (0.0)	69.1 (0.0)
HOSVD-HRTM	**100.0** (0.0)	**100.0** (0.0)	**100.0** (0.0)	**100.0** (0.0)
TT-HRTM	**100.0** (0.0)	**100.0** (0.0)	**100.0** (0.0)	**100.0** (0.0)
Sum-RecNTN	**99.9** (0.0)	95.3 (2.9)	71.5 (0.7)	73.2 (0.9)
Full-RecNTN	86.3 (0.2)	78.4 (0.9)	71.6 (1.2)	72.3 (0.7)
CP-RecNTN	**100.0** (0.0)	**100.0** (0.0)	**99.9** (0.1)	**99.9** (0.1)
HOSVD-RecNTN	**100.0** (0.0)	**100.0** (0.0)	**99.9** (0.1)	**99.8** (0.2)
TT-RecNTN	**100.0** (0.0)	**100.0** (0.0)	**99.9** (0.1)	**100.0** (0.0)
Sum-LSTM	**100.0** (0.0)	**99.5** (0.1)	81.2 (2.0)	74.0 (0.3)
Full-LSTM	**99.6** (0.2)	96.3 (0.2)	76.0 (0.8)	75.1 (1.0)
CP-LSTM	**99.9** (0.1)	**99.9** (0.0)	**99.7** (0.2)	**99.5** (0.3)
HOSVD-LSTM	**99.9** (0.1)	**99.9** (0.0)	**99.9** (0.0)	**99.9** (0.1)
TT-LSTM	**99.9** (0.1)	**100.0** (0.0)	**99.8** (0.0)	**99.8** (0.1)

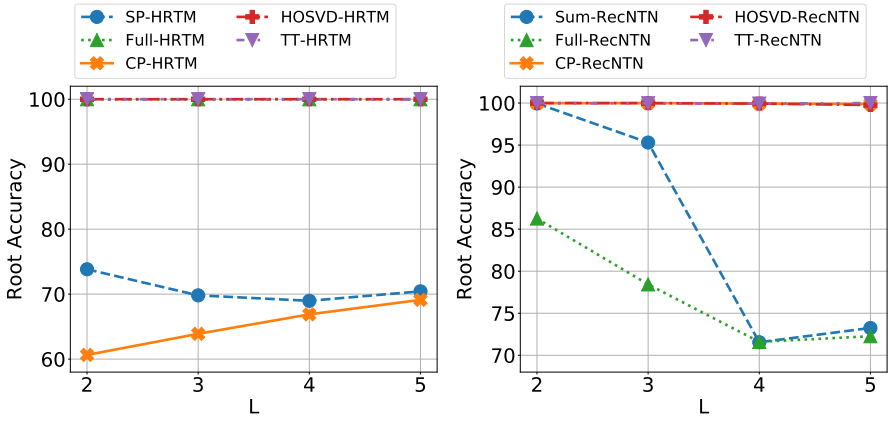
TABLE 4.3: Test root accuracy obtained by all the evaluated models on the BoolSent datasets with different maximum out-degree L. Values reported are averaged over three runs (standard deviation in brackets).

Results

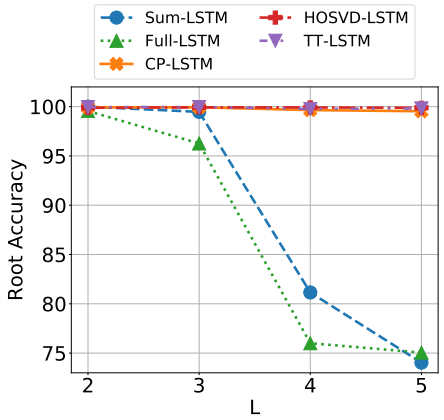
In Table 4.3, we report the test root accuracy obtained by all the models on the BoolSent dataset by varying the maximum tree out-degree L. The same results are depicted in Figure 4.5. The advantage of using tensor decompositions is evident: the models which leverage tensor decompositions to define their state-transition functions reach an accuracy of 100% regardless of the value of L. On the other hand, the sum-based and the full-tensor models struggle, especially when $L > 3$. Interestingly, this behaviour is shared between probabilistic and neural models, emphasising that the state-transition function inductive bias is a key design factor for recursive models.

The only exceptions to this trend are CP-HRTM and Full-HRTM (see Figure 4.5a). The former one is not able to achieve high accuracy for any value of L, even if its state-transition distribution is based on the canonical approximation. The latter always reaches an accuracy of 100%, even if it is a full-tensor model.

The advantages of using complex neural architecture as the LSTM cell are negligible



(A) HRTMs results. (B) RecNTNs results.



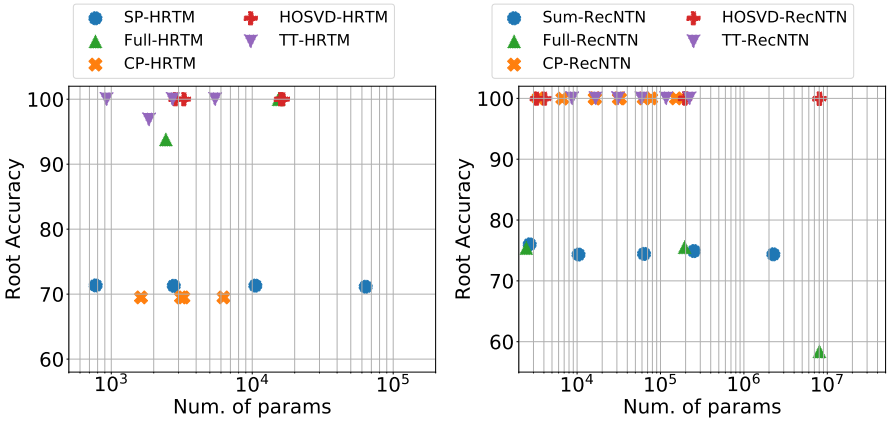
(C) LSTM-based models results.

FIGURE 4.5: Test root accuracy on the BoolSent task in relation to the input structure maximum out-degree L.

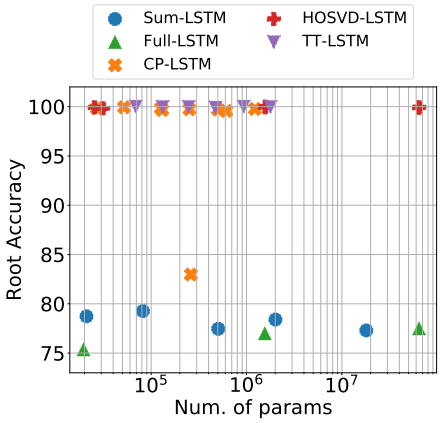
(see Figure 4.5b and Figure 4.5c). The sum-based and the full-tensor neural models achieve higher accuracies using a LSTM cell rather than a single neural layer as a state-transition function. Nevertheless, when the value of L increases, they still fail to reach the performances of approximated models.

In Figure 4.6, we report the validation accuracy achieved by all the model configurations evaluated on the BoolSent task with $L = 5$. The models which leverage tensor decompositions achieve 100% accuracy using far fewer parameters than the sum-based and the full-tensor approaches. This behaviour emphasises again the advantages of the models with the right inductive bias for the task considered. We discuss in deep this aspect in Section 4.4.5.

There is only one configuration of CP-LSTM which does not achieve an accuracy of 100%. Further investigation has shown that the learning process of such configuration stopped after few iterations due to the early-stopping criteria. Thus, in some cases, the early-stopping criteria selected can be too strict.



(A) HRTMs results. (B) RecNTNs results.



(C) LSTM-based models results.

FIGURE 4.6: Validation root accuracy for all the configurations tested on the BoolSent task with $L = 5$. For each configuration, we report the validation accuracy reached in relation to the number of parameters required.

4.4.4 List Operations Task

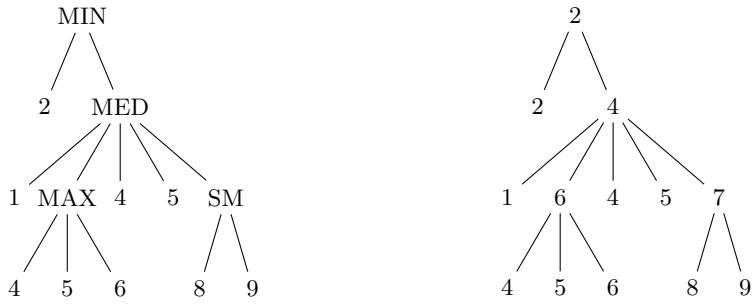
The goal of the ListOps task [123] is to predict the solution of a sequence of summary operations on lists of single-digit integers, written in prefix notation. Each element in the dataset consists of a sequence of operations and its solution. See [123] for more details on the dataset generation. For our purposes, we represent each sequence as a tree using its syntax tree built according to the following grammar:

$$S := \text{MAX}(S, \dots, S) \mid \text{MIN}(S, \dots, S) \mid \text{SM}(S, \dots, S) \mid \text{MED}(S, \dots, S) \mid \{0, \dots, 9\}, \tag{4.20}$$

where $\{0, \dots, 9\}$ are the digits that appear only on the leaf nodes; MAX, MIN, SM and MED are the logical operators which appear only on the internal nodes. In the following, we briefly describe the semantic of each operator.

MAX: its output is the maximum value among its input values.

MIN: its output is the minimum value among its input values.



(A) Input tree $\boldsymbol{\mathcal{X}}$. (B) Output tree $\boldsymbol{\mathcal{Y}}$.

FIGURE 4.7: Example of a ListOps input-output tree pair.

SM: its output is the summation of its input values modulo 10.

MED: its output is the median of its input values.

The dataset is already divided into training and test splits containing respectively 90% and 10% of the data [123]. We further sample 11% of the training set in order to build a validation set. Hence, we obtain a training set which contains 80k trees; both validation and test set contain 10k trees. The tree maximum out-degree is equal to five, i.e. $L = 5$.

In Figure 4.7, we report an example of input-output tree pair of the ListOps task. Clearly, all the input labels are categorical. There are 4 possible labels on the internal nodes and 10 possible labels on the leaf nodes. The output labels are also categorical since there are ten possible outcomes: the digits from 0 to 9. Note that in the original dataset in [123], the output label is attached only on the root nodes. We have extended it considering also intermediate results.

Following the experimental setting used in [123], we use a 2-layer neural network as output function of the neural recursive models. The first layer is a hidden layer with 20 hidden units, while the second layer is a softmax layer.

Results

In Table 4.4, we report the test root accuracy obtained by all the models on the ListOps task. In this task, the difference between the probabilistic and the neural models is evident. This is mainly motivated by the increased complexity of the task; in fact, in ListOps all the labels and the intermediate results are digits. Hence, there are 10^5 possible input configurations to consider for each operator.

In probabilistic models, TT-HRTM outperforms all the other models reaching an accuracy of 75%. Nevertheless, we were not able to terminate the model selection process on Full-HRTM due to the computational time required by the training.

In neural models, the use of tensor decompositions always leads to accuracies higher than 90%. On the other hand, the sum-based and the full-tensor neural models struggle to reach an accuracy of 80%. We argue that, also on this task, the inductive

	ACC_r%
SP-HRTM	27.2 (2.5)
Full-HRTM	Out Of Resources
CP-HRTM	22.7 (0.1)
HOSVD-HRTM	33.9 (0.8)
TT-HRTM	**75.6** (2.7)
Sum-RecNTN	76.4 (0.1)
Full-RecNTN	60.1 (1.0)
CP-RecNTN	**94.3** (0.8)
HOSVD-RecNTN	**96.2** (0.1)
TT-RecNTN	**93.0** (0.4)
Sum-LSTM	79.9 (1.0)
Full-LSTM	75.5 (1.2)
CP-LSTM	**94.2** (0.2)
HOSVD-LSTM	**97.8** (0.6)
TT-LSTM	**97.2** (0.2)

TABLE 4.4: Test root accuracy obtained by all the evaluated models on the ListOps task. All values are averaged over three runs (standard deviation in brackets).

bias introduced by the sum-based models does not match the task nature. We deepen this aspect in Section 4.4.5.

All the neural models benefit from a LSTM-based state-transition function. This is particularly evident when comparing the results obtained by Full-RecNTN and Full-LSTM.

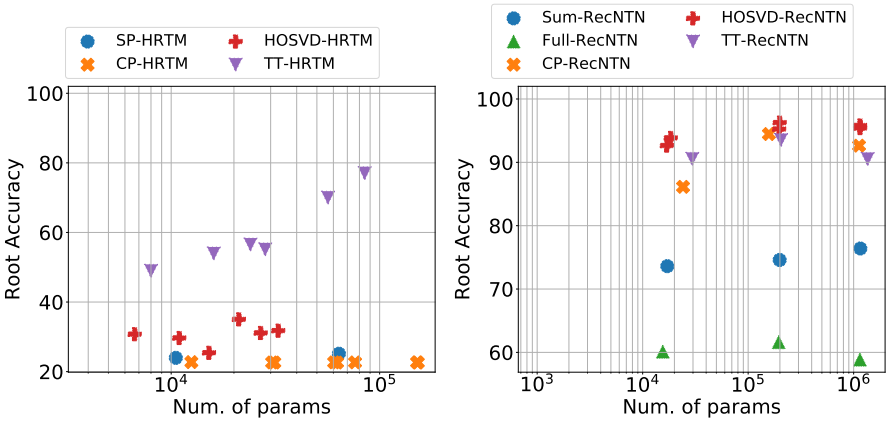
In Figure 4.8, we report the validation accuracy achieved in all configurations. Neural models (see Figure 4.8b and Figure 4.8c) follow the same trend highlighted in the BoolSent task. The models which leverage tensor decompositions achieve higher accuracy using fewer parameters than the sum-based and the full-tensor approaches. This trend is less evident in the probabilistic models (see Figure 4.8a). Nevertheless, TT-HRTM outperforms other approaches using fewer parameters.

Note that we cannot compare our results with the one reported in [123] due to the different parsing of input expressions.

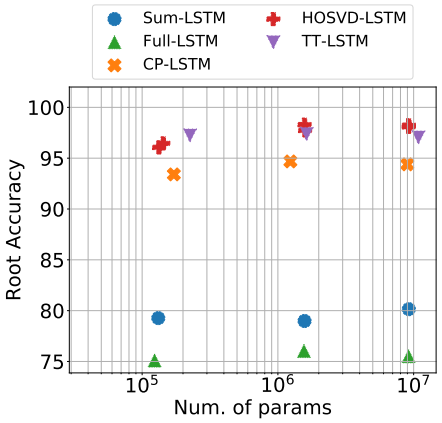
4.4.5 The Importance of the Inductive Bias

The results obtained on the BoolSent and the ListOps tasks show clearly the advantage of tensor decompositions when the maximum out-degree of the input structures increases. This advantage is independent of the class of the recursive model used to the tackle transduction task since both probabilistic and neural models show the same behaviour. Hence, we argue that the models which leverage tensor decompositions are able to outperform other models thanks to (1) the inductive bias introduced by the tensor decompositions and (2) the decoupling of models expressiveness from the hidden state size. To support our intuition, we study the ability of each model to learn the operators introduced in both tasks.

In Figure 4.9a, we show the test accuracy for each logical operator obtained by all the probabilistic models on the BoolSent dataset with $L = 5$. We also report the



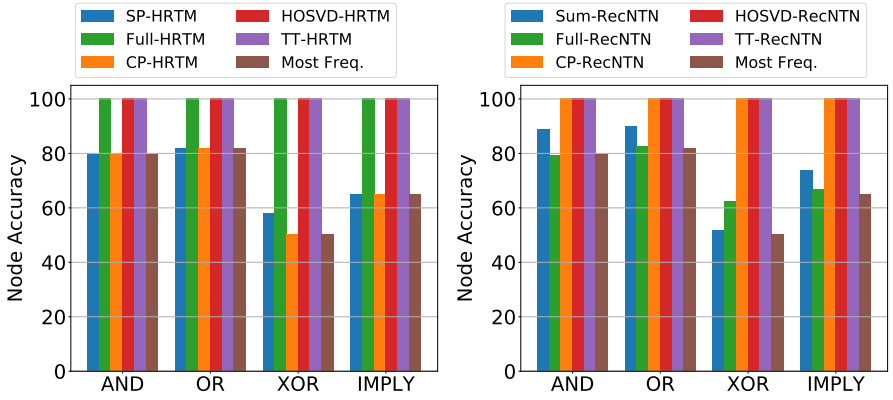
(A) HRTMs results. (B) RecNTNs results.



(C) LSTM-based models results.

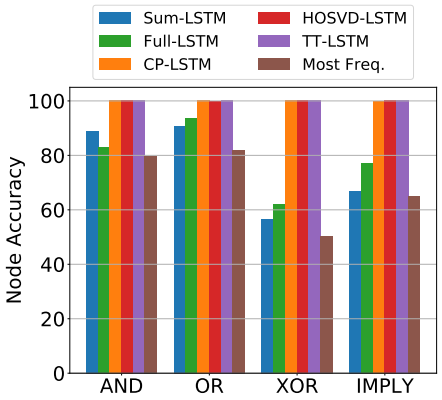
FIGURE 4.8: Validation root accuracy of all the configurations tested on the ListOps task. For each configuration, we report the validation accuracy reached in relation to the number of parameters required.

accuracy obtained by a dummy model which simply outputs the most frequent class for each operator[3]. By observing the plot, it is clear that SP-HRTM and CP-HRTM have the same performances as the dummy model; thus, they are not able to learn the input-output relation induced by each operator. SP-HRTM and CP-HRTM are the only models who impose an independence assumption among child nodes. Clearly, this assumption does not match the task characteristics.



(A) HRTMs results.

(B) RecNTNs results.



(C) LSTM-based models results.

FIGURE 4.9: Test node accuracy for each operator in the BoolSent task with $L = 5$.

In Figure 4.9b and Figure 4.9c, we report the test accuracy for each logical operator obtained by the neural models on the same dataset. Again, Sum-RecNTN and Sum-LSTM achieve performances that are comparable with the dummy model. In theory, with a sufficiently large hidden state size, such models should be able to implement a solution of the task. Nevertheless, in practice, the learning algorithm is not able to find such a solution even with a hidden state size of 300. Note that the state-transition functions of these two models can only capture first-order interactions among children; hence, the contribution of each child node is independent of the others. We argue that this assumption does not match the characteristics of the task; thus, the learning algorithm fails to find the correct solution.

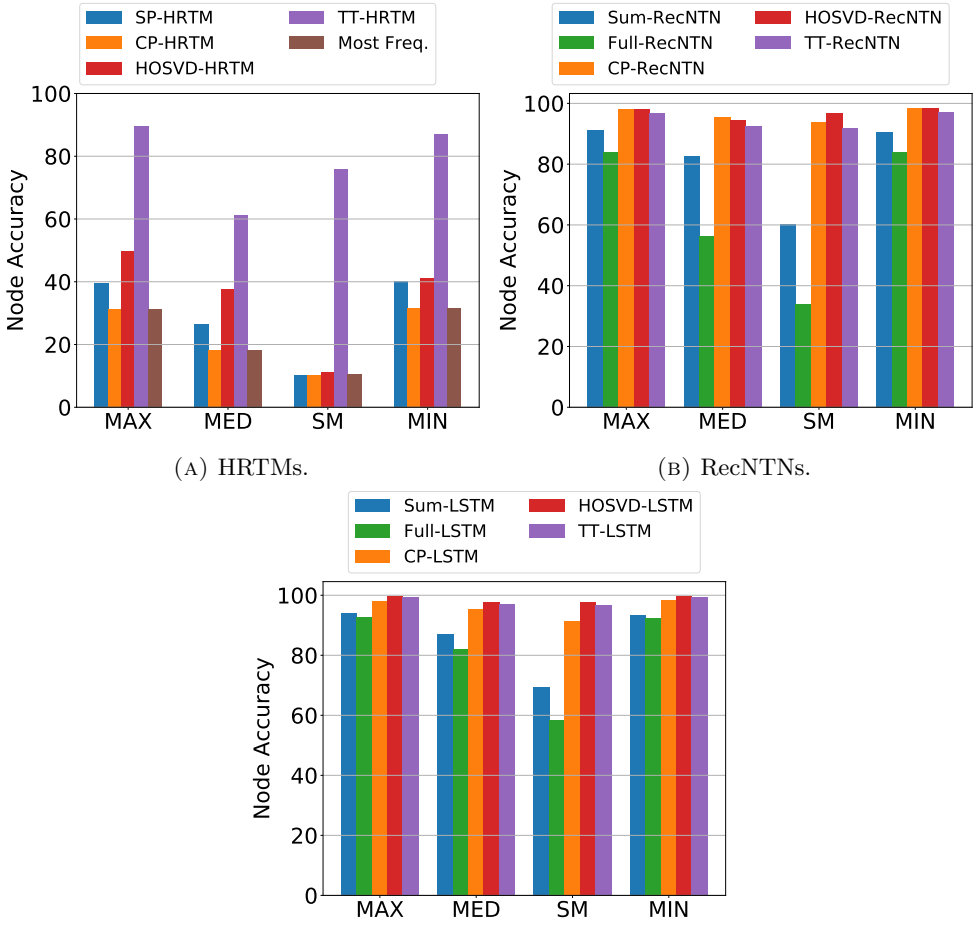
[3]Note that the solution implemented by the dummy model maximises the likelihood of output labels given the input one, i.e. $P(y_v \mid x_v)$. The structural information are ignored.

The full-tensor models (e.g. Full-RecNTN and Full-LSTM) are not able to reach an accuracy of 100% on any logical operators. In this case, we believe that the full-tensor models cannot solve the task due to the small hidden state size. In fact, in both models we validate only hidden state spaces of size 2,5 and 10 (see Table 4.2) to limit the number of model parameters which is $O(C^L)$. In theory, full tensor models can solve this task using a small hidden state since it is enough to encode the intermediate results (0 or 1) on internal nodes. However, in practice, neural models fail to find such a solution. We argue that this failure is due to the lack of the immediate feedback on internal nodes. In fact, neural models observe only the output labels on root nodes (see Section 4.4.2). Hence, we believe that, in this setting, the learning algorithm is not able to infer that hidden states should represent the intermediate results rather than an encodings of the structure. This hypothesis is supported by the results of HOSVD-RecNTN and HOSVD-LSTM; both models are able to reach an accuracy of 100% even when the decomposition rank is equal to 2. Thanks to the mapping induced by the mode matrices, the models are able to extract from the hidden child states all the necessary information to implement the logical operators (i.e. the intermediate results).

All the other neural and probabilistic models which leverage tensor decompositions achieve an accuracy of 100% on logical operators. The only exceptions is CP-HRTM, which obtains same performances as the dummy model due to the independence assumption imposed among child variables. The CP decomposition is the only decomposition which behaves differently when it is applied on neural or probabilistic models. We argue that this difference is due to the element-wise multiplication which is the basis of the decomposition. In probabilistic models, the probability constraints imposed on the factor matrices ensure us that all the elements multiplied are positive. Hence, we lose the negation property of the multiplication (i.e. $-1 \cdot -1 = 1$). Indeed, such a property can play a key role to implement the XOR operator. In fact, let us consider a mapping which transforms 0 and 1 values to positive and negative real numbers, respectively. Then, the output of the XOR operator applied on a list of boolean values can be obtained (1) by mapping all boolean values into real numbers according to the map just defined, and (2) by multiplying all the results together. If the result of the multiplication is negative, we are sure that the input list contains an odd number of values equal to 1. Such function cannot be easily encoded by CP-HRTM, resulting in poor performances on this task. On the other, can be straightforwardly implemented by the neural models based on the CP approximation.

If we turn our attention to the ListOps task, the performances of the probabilistic models heavily degrade. In Figure 4.10a, we report the test accuracy for each list operator obtained by all the probabilistic models on ListOps dataset. Also, we report the performances of a dummy model which simply outputs the most frequent digit for each operator. Observing the figure, it is clear that CP-HRTM obtains the same performances of the dummy model; thus, it is not able to learn the semantic of any digit operator. SP-HRTM and HOSVD-HRTM achieve better results than CP-HRTM;

nevertheless, these results are not satisfactory.



(A) HRTMs.

(B) RecNTNs.

(C) LSTM-based models.

FIGURE 4.10: Test node accuracy for each operator in the ListOps task.

In the case of SP-HRTM, these poor results are mainly motivated by the independence assumptions introduced by the SP approximation. The separation of the child contributions makes difficult the learning of operators such as MED and SM, where the final output is highly dependent on all the inputs of the operator.[4]

HOSVD-HRTM fails to solve the ListOps task due to the clustering performed by the rank variables. In the experiments, we validate $R \in \{2, 3\}$ (see Table 4.2); hence, hidden states are clustered in 2 or 3 groups before being combined by the probability distribution induced by the core tensor. The clustering does not allow learning the behaviour of the digit operators. For example, we expect that in MAX e MIN operator the clustering can be useful to separate higher digit (e.g. $\{7, 8, 9\}$) form lower digit (e.g. $\{0, 1, 2\}$). Nevertheless, to achieve an accuracy of 100%, the rank should be set equals to 10, losing the compression ability of the HOSVD approximation.

[4]On the contrary, in MAX e MIN operator the value of a single input element can be strongly correlated with the output operator. Let us consider the output of the following expression: $\text{MIN}(0, x, y, z)$: regardless the value of x, y and z, the output is always 0.

TT-HRTM outperforms all the other probabilistic models on all the operators. We argue that this result is due to the inductive bias of the TT decomposition which allows computing the parent hidden state through an iterative process. Each rank variable can be interpreted as an intermediate result of the computation. Hence, each operator can be decomposed considering one input element at a time: at each step, we combine the previous intermediate result with the current input to obtain the new updated intermediate result. It is worth highlighting that MIN, MAX and SM operators can be easily computed iteratively. On the other hand, the MED operator cannot. In fact, if we observe Figure 4.10a, we note that TT-HRTM has the lowest accuracy in the MED operator. Interestingly, TT-HRTM is the only model which does not perform as the dummy model on the SM operator.

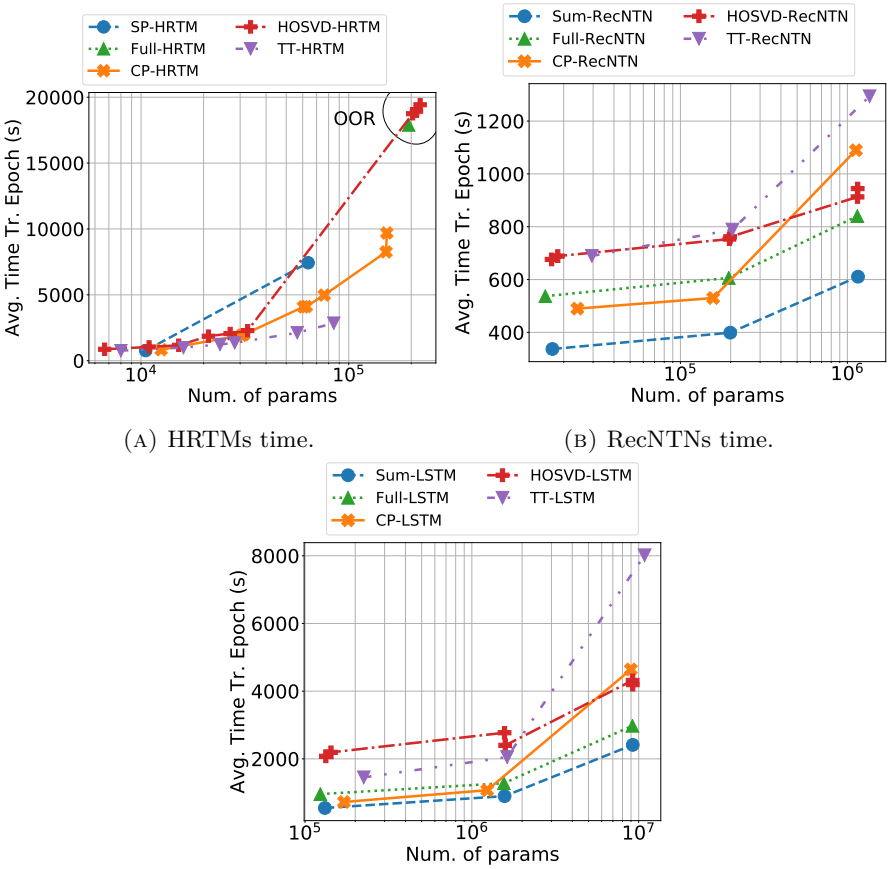
In Figure 4.10b and Figure 4.10c, we report the test accuracy for each digit operator obtained by RecNTNs and LSTM-based models on ListOps, respectively. The behaviour of neural models follows the trend highlighted in the BoolSent task. Interestingly, most of the failures are concentrated on the SM operator. We argue that such an operator is difficult to learn due to its non-monotonic behaviour.

4.4.6 Computational Complexity Analysis

To assess the computational complexity of the proposed models, we analyse the time required by all the models to complete a training epoch on the ListOps task. Figure 4.11a reports the time required by each probabilistic model in relation to the number of parameters. Note that, in this context, a training epoch comprises an upward and a downward recursive pass on the input structure (see Appendix D). For the sake of fairness, we also report time required by the configurations that do not complete training. Full-HRTM and HOSVD-HRTM are the most demanding models due to the exponential relation between the number of parameters and the maximum out-degree. Both models were not able to terminate the training when the hidden size (in Full-HRTM) or the rank (in HOSVD-HRTM) exceed 4.

Interestingly, SP-HRTM is more demanding than tensor based models such CP-HRTM and TT-HRTM. This is motivated by the exponentiation required to marginalise the switching variable (see Section 4.4.1).

In Figure 4.11b and Figure 4.11c, we report the epoch running time of RecNTNs and LSTM-based models, respectively. Note that, in neural models, a training epoch comprises a forward pass to compute the model output and a backward pass to compute the gradients and to update the parameters. Clearly, LSTM-based models are more demanding than RecNTNs. In both contexts, the sum-based models are the least demanding ones. Surprisingly, the full-tensor models are more efficient than the models which leverage tensor decompositions (for the same number of parameters). We argue that this behaviour is due to the optimisation of the back-end code. Hence, less computation on large operands (as in the case of full tensor models) are faster than more computation on small operands (as in the case of tensor decompositions).Neural



(A) HRTMs time.

(B) RecNTNs time.

(C) LSTM-based models time.

FIGURE 4.11: Average time required by all the configurations tested to complete a training epoch on the ListOps task. For each configuration, we report the average time needed to complete a single training epoch in relation to the number of parameters.

Models based on the TT approximation are the most demanding one due to the sequential processing of hidden child nodes imposed by the decomposition itself.

Even if the neural models which leverage tensor decompositions are slower than the full-tensor models when they have the same number of parameters, it is worth recalling that the tensor decompositions are fundamental to compress full-tensor models. In fact, for high values of L, the full tensor parameter cannot be even stored in the memory (see Section 2.4).

4.5 Conclusion

In this chapter, we have shown how tensor decomposition can be used to represent tensor state-transition functions in a compressed format. By imposing such a succinct representation, we obtain novel recursive models having decomposition factors as parameters. We have introduced nine different recursive models applying three tensor decomposition (i.e. CP, HOSVD, TT) on three model classes (i.e. probabilistic, neural and LSTM-based).

A key point of our analysis has been the study of the inductive bias introduced by each decomposition. In probabilistic models, the inductive bias can be interpreted as a set of independence assumptions imposed among the child hidden states. On the other hand, in neural models, the inductive bias is obtained by modifying the computational graph of the multi-affine map in the neural state-transition function. Regarding the different interpretations, all the recursive models which leverage tensor decompositions define a new hyper-parameter which regulates the trade-off between their expressiveness and their compression ability. Such a hyper-parameter corresponds to the decomposition rank.

Finally, we have experimentally assessed the advantages of the tensor-based models. To this end, we have introduced an ad-hoc task on boolean expressions. Despite the simplicity of the task, the results obtained have shown that the sum-based and the full-tensor neural models achieve performances comparable with a dummy model which simply outputs the most probable answer. In the sum-based models, we have argued that these performances are due to their strong inductive bias which impose an independence assumption among the child hidden states. In the full-tensor models, we have argued that the poor performances are due to the strict relation between their model complexity and the hidden state size. Even if small hidden state size should be sufficient to solve the task, the learning procedure is not able to infer that the hidden states should represent intermediate results rather than structural encodings. On the other hand, all the models based on the tensor decompositions (except CP-HRTM) always achieve 100% accuracy, independently of the model class, using a small number of parameters.

We have also conducted a second set of experiments on a benchmark from literature. The results obtained have confirmed the behaviour highlighted in the previous task. Moreover, we have analysed the computational cost of the proposed models, showing

that the neural models which leverage tensor decompositions are surprisingly slower than the full-tensor models when they have the same number of parameter. We have argued that this behaviour is attributable to optimisations in the computational backend, which favour execution of fewer operations on larger operands rather than more operations on small operands. Nevertheless, for large values of L, the full-tensor models could not even be stored in memory and the tensor decompositions are fundamental to reduce the number of parameters.

Chapter 5

Tensor Models for Unbounded Structured Data

5.1 Introduction

In many application domains, the maximum out-degree L of the input structures is not known. For example, in programming language processing, programs are represented through their abstract syntax trees. In this representation, each internal node represents a construct of the programming language and its child nodes represent the arguments of the construct. Some constructs do not have a fixed number of arguments, e.g. the sequential construct takes a list of commands as input and it executes all of them in the given order. Thus, abstract syntax trees are *unbounded* structures, i.e. their maximum out-degree L cannot be determined. Another domain where unbounded structures arise is natural language processing. In this context, sentences can be represented as trees where internal nodes represent syntactic categories and leaf nodes represent words. Especially in this domain, unbounded structures are converted to binary trees through a procedure called *binarisation*.

The recursive models introduced in the previous chapters cannot be used in this context. For example, the full-tensor models are parametrised by a L-way tensor; if the value of L is not known, their tensor parameters cannot be defined. Also, the recursive models based on tensor decompositions cannot be defined since they require the definition of a parameter for each child position.

This chapter aims to introduce new tensor-based recursive models which can learn from unbounded structures. To this end, we propose to combine tensor decompositions with a weight sharing constraint (see Section 2.2.2) imposed on decomposition factors. Unfortunately, this strategy cannot be applied to all tensor decompositions. For example, the HOSVD approximation still requires the definition of the core tensor (whose size depends on L) even if the weight sharing constraint is imposed.

The usage of weight sharing constraints on the state-transition parameters has been already proposed in the literature (e.g. [59, 159]). However, the weight sharing constraint usually leads to the definition of a state-transition function which ignores the child nodes order. As far as we know, there are two models in the literature which are able to handle unbounded structures without ignoring children order: the

Tree-Based Convolutional Neural Network [120] and the Multi-way Tree-LSTM [149]. The former achieves such a result imposing a softer weight sharing constraint: the parameters used to process a child node are obtained as a convex combination of two matrices. The combination coefficients depend on the child node position. The latter combines child information by using a bi-directional LSTM.

When weights sharing constraints are combined with tensor decompositions, the ability to exploit child nodes order depends on the approximation inductive bias. While the CP approximation leads to a state-transition function that ignores children order, the TT approximation exploits such information.

We experimentally assess the effectiveness of the proposed models on natural language processing tasks. In particular, we compare our models with baseline models that manage both binary and unbounded constituency trees.

In Section 5.2, we discuss in detail the formulation of the tensor decompositions with weight sharing constraints. Then, we introduce new probabilistic and neural recursive models for unbounded structures that leverage the CP and the TT approximation. In Section 5.3, we deeply analyse the performances of such models on natural language processing tasks. In particular, we compare their results with well-known baseline models highlighting the advantages of tensor decomposition bias. Finally, in Section 5.4, we draw our conclusion.

5.2 Infinite Recursive Tensor Models

In this section, we discuss how tensor decompositions can be combined with weight sharing constraints. Then, we introduce two infinite tensor approximations by imposing a weight sharing constraint to the CP and TT decomposition factors. These new approximations are the bases for defining new tensor models which learn from unbounded structures.

5.2.1 Tensor Decompositions and Weight Sharing

In Section 4.2, we have shown how tensor decompositions can be used to approximate tensor-based state-transition functions, obtaining recursive models whose parameters are the decomposition factors. Unfortunately, such models cannot be applied on unbounded structures since they associate a different parametrisation for each child node position. Thus, the number of child nodes (i.e. the structure out-degree L) must be known.

To overcome this limitation, we impose a weight sharing constraint on model parameters. In particular, we impose that the parameters used to process each child node is *shared* across all positions. Due to the relation between model parameters and decomposition factors, this is equivalent to impose a weight sharing constraint on the decomposition factors.

In the case of the HOSVD approximation, this constraint is not enough to guarantee the independence between the model definition and L. In fact, the HOSVD decomposition combines child information through a multi-linear operator parametrised by a L-th order tensor (i.e. the core tensor). Thus, even if we impose a weight sharing constraint on the mode matrices, the definition of the core tensor still depends on the value of L.

In the next two sections, we show how the CP and TT approximation can be combined with a weight sharing constraint. Moreover, we show how these approximations can be used to define probabilistic and neural recursive models for unbounded structures. As in the previous chapter, we focus only on the state-transition functions ignoring the input label. Note that the weight sharing constraint does not affect the learning algorithm.

5.2.2 Infinite Canonical Approximation

Let $\underline{\boldsymbol{T}} \in \mathbb{R}^{(C+1)\times\cdots\times(C+1)\times C}$ be a $(L+1)$-way tensor, its infinite CP approximation is defined as:

$$\underline{\boldsymbol{T}}[j_1, \ldots, j_L, k] \approx \sum_{r=1}^{R} \boldsymbol{U}[j_1, r] \ldots \boldsymbol{U}[j_L, r] Q[k, r], \tag{5.1}$$

where the matrix $\boldsymbol{U} \in \mathbb{R}^{(C+1)\times R}$ is the factor matrix shared among the first L dimension (i.e. the dimensions associated to the child information) and the matrix $\boldsymbol{Q} \in \mathbb{R}^{C\times R}$ is the factor matrix associated to the last dimension. Thus, the number of parameters required is $O(CR)$.

Note that with such a constraint, the CP approximation becomes permutational invariant on the first L dimensions. Thus, recursive models which leverage infinite CP approximation define a state-transition function that ignores the child nodes order.

The aforementioned property establishes an interesting connection between the infinite CP approximation and symmetric tensors (see Definition 2.9). In particular, we can state that this approximation can represent only tensors that are symmetric on the first L dimensions.

This connection it is useful to understand the expressive power of the approximation introduced. It can be shown that the CP approximation can represent any symmetric tensors if the same factor matrix is shared among all dimensions [41]. The value of the rank which makes the approximation an identity is called the *symmetric tensor rank*. In general, the symmetric tensor rank of a symmetric tensor is always greater or equal than its tensor rank [41, 150]. Thus, we argue that the infinite CP approximation can represent any symmetric tensors on the first L dimension. The value of R indicates the symmetric tensor rank of the approximation.

Probabilistic Interpretation (Infinite-CP-HRTM)

If we interpret factor matrices as probability distributions, we obtain the following state-transition distribution:

$$P(h_v \mid h_{v1}, \ldots, h_{v|\mathrm{ch}(v)|}, \underline{\boldsymbol{P}}) \approx \sum_{r_v} P(h_v \mid r_v, \boldsymbol{Q}) \prod_{l=1}^{|\mathrm{ch}(v)|} P(r_v \mid h_{vl}, \boldsymbol{U}), \tag{5.2}$$

where both $\boldsymbol{Q}$ and $\boldsymbol{U}$ should satisfy probability constraints.

Neural Interpretation (Infinite-CP-RecNTN)

The same approximation can be used to define the multi-affine map of a neural tensor state-transition function:

$$\underline{\boldsymbol{T}}(\bar{\boldsymbol{h}}_{v1}, \ldots, \bar{\boldsymbol{h}}_{v|\mathrm{ch}(v)|}) \approx \boldsymbol{Q} \left(\bigodot_{l=1}^{|\mathrm{ch}(v)|} \boldsymbol{U}^{\mathsf{T}} \bar{\boldsymbol{h}}_{vl} \right). \tag{5.3}$$

5.2.3 Infinite Tensor-Train Approximation

Let $\underline{\boldsymbol{T}} \in \mathbb{R}^{(C+1)\times\cdots\times(C+1)\times C}$ be a $(L+1)$-way tensor, its infinite TT approximation is defined as:

$$\underline{\boldsymbol{T}}[j_1, \ldots, j_L, k] \approx \sum_{r_1}^{R_1} \cdots \sum_{r_{D-1}}^{R_{D-1}} \underline{\boldsymbol{G}}[\perp, j_1, r_1] \ldots \underline{\boldsymbol{G}}[r_{L-1}, j_L, r_L] \ldots \boldsymbol{Q}[r_L, k], \tag{5.4}$$

where $\underline{\boldsymbol{G}} \in \mathbb{R}^{(R+1)\times(C+1)\times(R+1)}$ is the core tensor shared among the first L dimension; we use the special value $\perp$ to include the first core matrix in the definition of $\underline{\boldsymbol{G}}$. The value of R is the rank of the approximation. Thus, the number of parameters required is $O(CR^2)$.

As we have shown in Section 4.2.4, the TT approximation aggregates child information iteratively, considering one child at a time. Each child hidden state is aggregated to the previous intermediate result by applying the corresponding core tensor, which is different for each child position. Thus, the weight sharing constraint imposed on the core tensors forces the application of the same multi-linear operator at each step. From this point of view, the infinite TT approximation defines a recursive process which aggregates child information. Clearly, even if the core tensor is shared across all the first L dimensions, the output of this recursive process depends on the child nodes order.

The recursive process induced by the infinite TT approximation establishes a connection between TT decomposition and recursive models for sequences (see Section 2.3.2). In the probabilistic context, the infinite TT approximation is equivalent to the so-called translation invariant or uniform Matrix Product States (uMPS) with a positivity constraint imposed on the core tensor [2]. It can be shown that non-negative uMPS are equivalent to HMMs [42, 97, 66, 2]. Nevertheless, it is worth highlighting that uMPS (and also the infinite TT approximation) do not impose that the probability

distribution represented by the core tensor $\underline{\boldsymbol{G}}$ decomposes into a *state-transition* and an *output* distribution (as it happens in HMMs). In the neural context, the connection with RNNs is more delicate due to the non-linear activation function [92, 91] that is missing in the TT decomposition.

Probabilistic Interpretation (Infinite-TT-HRTM)

If we interpret core tensors as probability distributions, we obtain the following state-transition distribution:

$$P(h_v \mid h_{v1}, \dots, h_{v|\mathrm{ch}(v)|}, \underline{\boldsymbol{P}}) \approx \sum_{r_{v1}} \cdots \sum_{r_{vL}} \prod_{l=1}^{|\mathrm{ch}(v)|} P(r_{vl} \mid r_{vl-1}, h_{vl}, \underline{\boldsymbol{G}}) P(h_v \mid r_{v|\mathrm{ch}(v)|} \boldsymbol{Q}), \tag{5.5}$$

where $\underline{\boldsymbol{G}}$ and $\boldsymbol{Q}$ should satisfy the probability constraints. Moreover, we assume $r_{v0} = \perp$.

Neural Interpretation (Infinite-TT-RecNTN)

The same approximation can be used to define the multi-affine map of a neural tensor state-transition function:

$$\underline{\boldsymbol{T}}(\bar{\boldsymbol{h}}_{v1}, \dots, \bar{\boldsymbol{h}}_{v|\mathrm{ch}(v)|}) \approx \boldsymbol{Q}^{\mathsf{T}} \left(\underline{\boldsymbol{G}} \left(\dots \underline{\boldsymbol{G}} \left(\underline{\boldsymbol{G}}(\perp, \bar{\boldsymbol{h}}_{v1}), \bar{\boldsymbol{h}}_{v2} \right) \dots, \bar{\boldsymbol{h}}_{v|\mathrm{ch}(v)|} \right) \right). \tag{5.6}$$

5.3 Application to Natural Language Processing

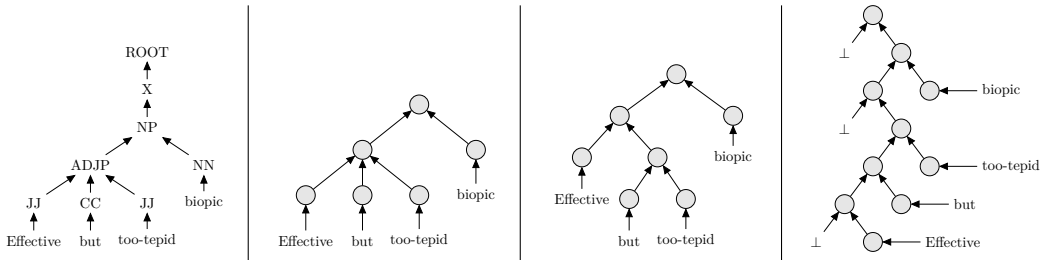
In this section, we show how the infinite models proposed in this chapter can be used in Natural Language Processing (NLP) tasks. The NLP domain is a natural choice to assess our proposal for two reasons: (1) it is an active application domain for recursive models and (2) most recursive models rely on binary-versions of the input structures.

5.3.1 Sentences as Structures

A central problem in NLP is the learning of a distributed encoding of sentences, as this is the stepping stone for many NLP tasks (e.g. sentence classification, sentiment analysis and natural language inference). A key design choice in developing such models is how the input data (i.e. sentences) are represented.

The simpler sentence representation is the bag-of-words, which depicts the sentences as words multi-sets ignoring the words order. Despite its simplicity, this representation has been used to obtain meaningful sentences encodings [89, 173, 5].

The sequence representation overcomes the limitation of bag-of-words by considering the sentences as an ordered sequences of words. Thus, the words order matters. This representation allows building models that progressively constructs the sentence encoding, processing one word at a time. Moreover, it paves the way for applying recurrent models for sequences, such as HMM, RNN and LSTM (see Section 2.3.2), in the NLP domain.



(A) Original. (B) Unbounded. (C) Binary. (D) TreeNet [32].

FIGURE 5.1: Constituency tree of the sentence "*Effective but too-tepid biopic*" taken from the Sentiment Stanford Treebank [153] test set.

A key aspect of the sentences, which is missing in the sequential processing, is the compositionality. For example, the sentence "*The sky is blue and the grass is green*" is obtained by composing the two sub-phrases "*The sky is blue*" and "*the grass is green*" with the conjunction "*and*". The intrinsic compositionality of the sentences makes them suitable for a tree representation, where the whole sentence (the root) is built in terms of sub-phrases (the internal nodes) which in turn are defined in terms of smaller constituents; the base cases are the words (the leaves) since they are the atomic piece of information. This representation takes the name of *constituency tree*. In Figure 5.1a, we show the constituency tree of the sentence "*Effective but too-tepid biopic*": the leaves are the words while internal nodes represent syntactic categories which are the constituents of the whole sentence.

The constituency trees allow using recursive models to build sentences encodings. The idea is to interpret the hidden states produced by these models as the distributed representation of the sentence. The hidden state of the root is the encoding of the whole sentence, while the hidden states of the internal nodes are the encodings of sentence sub-phrases. In general, we cannot define an upper bound on the number of child nodes in constituency trees.

A common technique to remove the unboundedness of the tree structures is the binarisation. The aim of the binarisation is to transform a generic L-ary tree into a binary tree. However, the price to pay is the loss of structural information. For example, in Fig. 5.1b and Fig. 5.1c we report the constituency and the binary constituency tree of the sentence "*Effective but too-tepid biopic*". By comparing these two representations, we can observe that the binary tree has one more node that breaks the ternary relation in the unbounded tree; in general, to break a node with L child nodes, we should add $L-2$ new nodes. All these new nodes create a chain which moves away the child nodes of the L-ary relation from their parent. Moreover, their composition is obtained by considering one child at a time, as it happens in the sequence representation. Hence, the binarisation removes the equality among child nodes, with the risk of weakening the contribution of child nodes that are moved far away from their parent and strengthening the contribution of the ones that remain close.

5.3.2 Related Works

There are many models which compute a sentence encoding starting from its constituency tree. For our purposes, we restrict the discussion to the recursive models which fit in the framework discussed in Section 2.3.

Within this framework, recursive neural models seems to be preferred over probabilistic models. For example, the Matrix-Vector Recurrent Neural Network [152] and the Recursive Neural Tensor Network [153] are specialisation of the RecNN architecture to binary constituency trees.

Also Tree-LSTMs has been proposed by Tai et al. [159] to tackle NLP tasks. In particular, they propose two different Tree-LSTM architectures: the N-ary Tree-LSTM, which is the architecture we introduced in Section 2.3.3, and the Child-Sum Tree-LSTM. The former architecture is applied to binary constituency trees. The latter is able to handle unbounded structures but has been applied only to dependency trees, which are another kind of tree representation for sentences.

In recent years, Tree-LSTMs have been used as a building block to develop more sophisticated models. For example, [88], [108], [94], [148] build new Tree-LSTM models which define dynamic state-transition functions depending on syntactic categories (i.e. Part-Of-Speech tags). Instead, [162] introduces a Bidirectional Tree-LSTM which takes advantage of both parsing directions: bottom-up and top-down. As we stated before, the constituency trees are intrinsically bottom-up; to this end, the author introduces a first bottom-up pass, called *head lexicalization*, to propagate information from the leaves to the root. All these models are applied only to binary constituency trees.

As far as we know, the only work which builds a model suitable for unbounded constituency trees is the TreeNet [33]. The idea is to consider all the child nodes in a chain: the hidden state of a node depends on the hidden state of its left sibling and its rightmost child. Even if the model itself works with unbounded trees, the state-transition function defined is binary since it always composes two elements.

Before continuing with our experimental analysis, in the next two paragraphs, we describe two models which are particularly relevant for our assessment: Child-Sum Tree-LSTM and TreeNet. In particular, we highlight (1) the connection between Child-Sum Tree-LSTM and infinite models introduced in this chapter and (2) the binarisation introduced by TreeNet state-transition function.

Child-Sum Tree-LSTM. Child-Sum Tree-LSTM [159] extends the Tree-LSTM (see Section 2.3.3) under the assumption that there is no order among child nodes. The Child-Sum Tree-LSTM state-transition is completely determined by the following

equations:

$$\boldsymbol{i}_v = \sigma\left(\boldsymbol{W}^i\boldsymbol{x}_v + \sum_{l=1}^{|\mathrm{ch}(v)|}\boldsymbol{U}^i\boldsymbol{h}_{vl} + \boldsymbol{b}^i\right), \tag{5.7a}$$

$$\boldsymbol{o}_v = \sigma\left(\boldsymbol{W}^o\boldsymbol{x}_v + \sum_{l=1}^{|\mathrm{ch}(v)|}\boldsymbol{U}^o\boldsymbol{h}_{vl} + \boldsymbol{b}^o\right), \tag{5.7b}$$

$$\boldsymbol{u}_v = \sigma\left(\boldsymbol{W}^u\boldsymbol{x}_v + \sum_{l=1}^{|\mathrm{ch}(v)|}\boldsymbol{U}^u\boldsymbol{h}_{vl} + \boldsymbol{b}^u\right), \tag{5.7c}$$

$$\boldsymbol{f}_{vl} = \sigma\left(\boldsymbol{W}^f\boldsymbol{x}_v + \boldsymbol{U}^f\boldsymbol{h}_{vl} + \boldsymbol{b}^f\right) \quad \forall l \in [1, |\mathrm{ch}(v)|], \tag{5.7d}$$

$$\boldsymbol{c}_v = \boldsymbol{i}_v \odot \boldsymbol{u}_v + \sum_{l=1}^{|\mathrm{ch}(v)|}\boldsymbol{f}_{vl} \odot \boldsymbol{c}_{vl}, \tag{5.7e}$$

$$\boldsymbol{h}_v = \boldsymbol{o}_v \odot \tanh(\boldsymbol{c}_v). \tag{5.7f}$$

Observing Eq. (5.7), we can deduce that Child-Sum Tree-LSTM can be derived from the L-ary Tree-LSTM (see Section 2.3.3) by imposing a weight sharing on the matrices which process the child nodes. Thus, this model can be interpreted as the generalisation of the first-order approximation of RecNTN (see Section 3.3.2) to unbounded input structures. In fact, as the Infinite-CP-RecNTN combines the CP approximation with a weight sharing constraint, the Child-Sum Tree-LSTM combines the first-order approximation with a weight sharing constraint. For this reason, we refer to this model as Infinite-Sum-LSTM.

TreeNet. TreeNet [33] has been introduced with the aim of learning from unbounded tree-structured data. The idea is to process child nodes sequentially from left to right; only the rightmost child node is linked to the parent node. Hence, each node composes the information coming from two sources: its left sibling and its rightmost child. Formally, the composition is defined by the following equations:

$$\begin{aligned} &\boldsymbol{o}_v = \sigma\left(\boldsymbol{U}^o_s\boldsymbol{h}_{vs} + \boldsymbol{U}^o_c\boldsymbol{h}_{vc} + \boldsymbol{b}^o\right), && \boldsymbol{c}_v = \boldsymbol{f}_{vs} \odot \boldsymbol{c}_{vs} + \boldsymbol{f}_{vc} \odot \boldsymbol{c}_{vc}, \\ &\boldsymbol{f}_{vk} = \sigma\left(\boldsymbol{U}^f_{ks}\boldsymbol{h}_{vs} + \boldsymbol{U}^f_{kc}\boldsymbol{h}_{vc} + \boldsymbol{b}^f_k\right), \quad k \in \{s, c\}, && \boldsymbol{h}_v = \boldsymbol{o}_v \odot \tanh(\boldsymbol{c}_v), \end{aligned} \tag{5.8}$$

where $\{\boldsymbol{h}_{vs}, \boldsymbol{c}_{vs}\} \in \mathbb{R}^C$ are the hidden state and the memory cell of left sibling of v, while $\{\boldsymbol{h}_{vc}, \boldsymbol{c}_{vc}\} \in \mathbb{R}^C$ are the hidden state and the memory cell of the rightmost child node of v.

If the node v is a leaf, its hidden state depends solely on the input label x_v:

$$\begin{aligned} &\boldsymbol{i}_v = \sigma\left(\boldsymbol{W}^i\boldsymbol{x}_v + \boldsymbol{b}^i\right), && \boldsymbol{o}_v = \sigma\left(\boldsymbol{W}^o\boldsymbol{x}_v + \boldsymbol{b}^o\right), && \boldsymbol{u}_v = \sigma\left(\boldsymbol{W}^u\boldsymbol{x}_v + \boldsymbol{b}^u\right), \\ &\boldsymbol{c}_v = \boldsymbol{i}_v \odot \boldsymbol{u}_v, && \boldsymbol{h}_v = \boldsymbol{o}_v \odot \tanh(\boldsymbol{c}_v). \end{aligned} \tag{5.9}$$

Observing Eq. (5.8), we can argue that TreeNet cell is a binary Tree-LSTM cell

	Infinite Models		Binary Models
Approx.	Probabilistic	LSTM-based	LSTM-based
CP	Infinite-CP-HRTM	Infinite-CP-LSTM	-
TT	Infinite-TT-HRTM	Infinite-TT-LSTM	-
Existing	Infinite-SP-HRTM	Infinite-Sum-LSTM [159]	Binary Sum-LSTM [159] TreeNet [33]

TABLE 5.1: List of all the models assessed on NLP tasks.

(see Section 2.3.3) without the input gate and the update value. In fact, in both models, all the gates are computed by composing two constituents. The TreeNet defines the constituents of a node as its left sibling and its rightmost child, while the binary Tree-LSTM used directly its left and right child node. Hence, the difference between these two models lies on how the tree is binarised, rather than on how the tree is processed. In Figure 5.1d, we show an example on how a constituency tree is binarised according to the TreeNet; the constituent *ADJP* of the original constituency tree (see Figure 5.1a) is composed of three words: "*Effective*", "*but*", "*too-tepid*". The TreeNet breaks this ternary relation processing one words at a time; as we can see in Figure 5.1d, the node which has the first word "*Effective*" is combined with a bottom node since it does not have a left sibling. The result is then fused with the word "*but*". Finally, also the word "*too-tepid*" is combined with the result of the previous composition, obtaining the encoding of the constituent *ADJP*. Hence, the original ternary relation is broken into a sequence of three binary relations (one for each word) each of them combines the composition of the previous words with the new word.

5.3.3 Experimental Analysis

In this section, we experimentally assess the effectiveness of the infinite approximations proposed in the NLP tasks. In particular, we evaluate two probabilistic and two neural models for unbounded structures. The probabilistic models are the Infinite-CP-HRTM and the Infinite-TT-HRTM. As neural models, we assess the Infinite-CP-LSTM and the Infinite-TT-LSTM. These two models are obtained by using the infinite neural multi-affine maps to compute LSTM gates (see Section 4.3). We do not evaluate Infinite-CP-RecNTN and Infinite-TT-RecNTN since the LSTM cells usually perform better in the NLP domain.

We also evaluate three other baseline models: Infinite-Sum-LSTM, Binary Sum-LSTM and TreeNet. The former is useful to assess the difference between tensor approximations and the first-order approximation on unbounded structures; Binary Sum-LSTM and TreeNet are useful to assess the difference between unbounded and binary constituency trees. In Table 5.1, we report all the models evaluated in our analysis.

For the sake of completeness, we also evaluate the probabilistic model which is obtained by combining SP approximation (see Section 3.3.1) with weight sharing constraints. We briefly introduce this model in the next paragraph.

Infinite SP-HRTM. Infinite-SP-HRTM is the generalisation of SP-HRTM (see Section 3.3.1) to unbounded input structures. Such a generalisation is obtained by imposing a stationarity assumption on the mixture components of the state-transition distribution. Recalling the SP state-transition distribution definition:

$$P(h_v \mid h_{v1}, \dots, h_{vL}) = \sum_{S_v=l}^{L} P(S_v = l) P(h_v \mid S_v = l, h_{vl}), \tag{5.10}$$

we impose that the parametrisation of the mixture components does not depend on child position l. Moreover, we assume that $P(S_v = l)$ is a uniform distribution and it is not learned. In this way, we are able to define the mixture distribution even if the maximum out-degree L is unknown. Thus, we obtain the following infinite state-transition distribution:

$$P(h_v \mid x_v, h_{v1}, \dots, h_{vL}) = \frac{1}{|\mathrm{ch}(v)|} \sum_{S_v=l}^{|\mathrm{ch}(v)|} P(h_v \mid S_v = l, x_v, h_{vl}, \underline{\boldsymbol{U}}), \tag{5.11}$$

where $\boldsymbol{U} \in \mathbb{R}_{\geq 0}^{(C+1)\times C}$ is the parameter shared across all the mixture components. The Infinite-SP-HRTM requires $O(C^2)$ parameters.

The major drawback of this definition is that we assign the same weight to each child contribution. Thus, we lose the ability of the SP approximation to learn from data the child nodes positions that are more relevant for the parent hidden state.

Tasks

We evaluate the performances of the proposed infinite tensor models on two NLP tasks: the *sentence classification* and the *semantic textual similarity* task. In the following, we describe the dataset used in our experiments for each task.

Sentence Classification. In the sentence classification task, the goal is to predict the class of a given input sentence. The class is usually a discrete label. This task can be tackled by recursive models as super-source transductions. To this end, recursive models process the input sentence to produce a succinct representation. Then, the succinct representation is used to predict the class of the whole input sentence.

Sentiment Stanford Treebank (SST) [153]. The SST dataset consists of 11855 constituency trees divided as follows: 8544 trees in the training set, 1101 trees in the validation set and 2210 in the test set. Sentences have been extracted from film reviews dataset and then parsed using the Stanford parser [153]. We refer to [153] for a detailed description of the dataset generation procedure.

SST attaches a sentiment label for all internal nodes in the dataset. The label y_v of an internal node v indicates the sentiment of the sub-phrase represented by v. The label of a root node indicates the sentiment of the whole sentence. Each sentiment label y_v can take five values: very negative, negative, neutral, positive, and

very positive. This sentiment classification is usually denoted as *fine-grained*. We refer to the dataset with the fine-grained labels as SST-5.

The same dataset is also used for a *binary* sentiment classification task, in which there are only two sentiment labels: negative and positive. In this setting, all neutral sentences are excluded and all negative (positive) sentences are collapsed in one cluster. Hence, the binary dataset contains less input trees. In particular, it contains 6920/872/1821 trees in the training, validation and test set, respectively. We refer to the dataset with the binary labels as SST-2.

The SST input constituency trees are already binarised. Thus, they cannot be used to assess models for unbounded structures. To remove such a binarisation, we re-parse all sentences.

By removing the binarisation, it is no longer possible to attach a sentiment label to all internal nodes. In fact, unbounded constituency trees define new sub-phrases that may not exist in the binary dataset. Thus, they cannot be labelled. As a reference, note that SST-5 binary constituency trees data contains 119.413 labels, while the unbounded constituency trees data contains only 91.536 labels.

TREC [107]. The TREC dataset consists of 5952 questions divided as follows: 4952 questions in the training set, 500 questions in the validation set and 500 questions in the test set. The questions have been retrieved from different sources and they have been manually labelled according to the semantics of their answers. We refer to [107] for a detailed description of the dataset generation procedure.

There are 6 possible question class: ABBREVIATION, ENTITY, DESCRIPTION, HUMAN, LOCATION and NUMERIC VALUE. Each class is divided in sub-classes, for a total of 50 fine-grained classes. In our work, we use only the coarse classification.

Semantic Textual Similarity. In semantic textual similarity task, the goal is to predict the similarity between two sentences. The similarity is usually measured as a score or label. This task cannot be tackled as a structural transduction since we have two input structures. Nevertheless, we can still use recursive models to produce the encodings of the input sentences. Such encodings are then combined to compute the similarity between input sentences.

Sentences Involving Compositional Knowledge (SICK) [113]. The SICK dataset consists of 9927 sentence pairs divided as follows: 4500 pairs in the training set, 500 pairs in the validation set and 4927 pairs in the test set. Sentences are derived from image and video description datasets, including only the sentences which contains relation among general concepts (e.g. the relation between a bride and a groom) [113]. We refer to [113] for a detailed description of the dataset generation procedure.

For each input pairs, two output values are provided: a *relatedness score* $\hat{y}_{\text{rel}}$ and an *entailment label* $\hat{y}_{\text{ent}}$. The relatedness score is a real value ranging from 1 to 5 which measures the degree of the semantic similarity between the input sentences. On

the other hand, the entailment label indicates the entailment relation holding between the input sentences. Let (A, B) be an input sentence pair, the possible attached entailments are:

Entailment: the sentence B cannot be false when A is true.

Contradiction: the sentence B is false when A is true.

Neutral: the truth of B could not be determined on the basis of A.

In the remainder of the section, we denote as SICK-R and SICK-E the SICK dataset with the relatedness scores and the entailment labels, respectively.

Experimental Settings

In this section, we describe the experimental settings used to conduct our analysis. Besides the experimental setup for probabilistic and neural models, we also detail the preprocessing procedure used to generate the input constituency trees. The implementation code of the infinite models and the sentence preprocessing pipeline can be found here.[1]. See Section 4.4.1 for more details on the model implementations.

Data Preprocessing. Constituency trees are built using the PCFG constituency parser of the Stanford Core NLP [112]. Also, we binarise them by computing the Chomsky Normal Form available in the Natural Language Tool Kit [21]. To facilitate the learning process, we collapse all unary relations. In all the experiments, we remove internal input labels which represent the syntactic categories since this information is not exploited by the assessed models.

We represent the words attached to the leaf nodes by 300-dimensional vectors initialised using Glove word embeddings [132]. Note that we prefer to use Glove embeddings rather than state-of-the-art embeddings (such as BERT [47] or ELMo [134]) because we would make a fair comparison between the proposed models and the models used as baselines (e.g. Tree-LSTM and Tree-Net). The models used as baselines have been already tested using the Glove embeddings. Also, it is worth highlighting that the purpose of the proposed experiments is to show that tensor-based models can learn interesting properties of NLP tasks. From this point of view, we believe that the usage of more powerful embeddings is negligible.

Probabilistic Models Configurations. Probabilistic models tackle sentence classification tasks as transductions. In particular, the SST task is addressed as a structural transduction since an output label is associated for each input node. On the contrary, the TREC task is addressed as super-source transduction since the output label is attached only to the root nodes. In both cases, the output labels are categorical. Thus, the generation of the output labels is handled through a categorical emission distribution. We do not assess the probabilistic models on the semantic textual

[1] https://github.com/danielecastellana22/tensor-tree-nn

	Hyper-parameters Values	
	C	R
Infinite-SP-HRTM	$\{20, 50, 100\}$	-
Infinite-CP-HRTM	$\{20, 50, 100\}$	$\{20, 50, 100\}$
Infinite-TT-HRTM	$\{20, 50, 100\}$	$\{10, 20\}$

TABLE 5.2: Hyper-parameters values validated for each HRTM on NLP tasks.

similarity task due to the poor performances obtained on the sentence classification task.

Input labels are attached only to the leaf nodes. On the internal nodes, we attach a special label $\perp$. Thus, the same state-transition parametrisation is used on all the internal nodes. The input labels of the leaf nodes are 300-dimensional vectors and they are modelled as continuous random variables. Therefore, we cannot define the prior distribution $P(h_v \mid x_v)$ since the hidden variable H_v is discrete while the input variable X_v is continuous. As we have already shown in Section 3.2.1, we handle the continuous input variable by defining the distribution $P(x_v \mid h_v)$. Such a distribution is continuous and its parametrisation depends on the state h_v. In our case, we define $P(x_v \mid h_v)$ as 300-dimensional multivariate Gaussian distribution with a diagonal covariance matrix:

$$x_v \sim \mathcal{N}(\mu_{h_v}, \text{diag}(\sigma^2_{h_v})), \tag{5.12}$$

where $\mu_{h_v} \in \mathbb{R}^{300}$ and $\sigma_{h_v} \in \mathbb{R}^{300}$ are the parameters of the distribution. Note that each parameter depends on the hidden state h_v. The distribution $P(x_v \mid h_v)$ requires $C \times 300$ parameters for all the mean values and $C \times 300$ parameters for all the variance values. This formulation allows inferring the leaf hidden state h_v given the word embeddings x_v since $P(h_v \mid x_v) \propto P(x_v \mid h_v)P(h_v) \propto P(x_v \mid h_v)$. The last equality holds because we do specify a flat prior.

Since input structures are singly connected, all probabilistic models are trained through tailored versions of the EM algorithm detailed in Appendix D. We execute the EM algorithm for a maximum of 200 iterations. Also, we use early-stopping criteria on the validation root accuracy to stop the training if the root accuracy does not increase for 50 consecutive iterations.

We perform a model selection on the validation set to find the best hyper-parameters configuration. We validate only two hyper-parameters: the number of hidden states C and the rank R (if it is used). We choose to validate only these two hyper-parameters because they are strictly related to the model expressiveness and the number of parameters required. In Table 5.2, we report the hyper-parameters values validated.

Neural Models Configurations. As in probabilistic models, we assume a special label $\perp$ attached to each internal node. Thus, the state-transition function parametrisation is the same for all the internal nodes. The hidden states of leaf nodes are computed by applying a one-layer NN: $h_v = \sigma(\boldsymbol{W}^x \bar{\boldsymbol{x}}_v)$, where v is a leaf node,

$x_v \in \mathbb{R}^{300}$ is the word embedding attached to v, σ is the sigmoid activation function and $\boldsymbol{W} \in C \times 301$ is the augmented matrix which parametrises the neural layer.

The output layer depends on the task. In the sentence classification tasks, the output layer is a one-layer MLP defined as:

$$\boldsymbol{s}_v = \mathrm{ReLU}\left(\boldsymbol{W}'\boldsymbol{h}_v + \boldsymbol{b}'\right), \quad \boldsymbol{p}_v = \mathrm{softmax}\left(\boldsymbol{W}''\boldsymbol{s}_v + \boldsymbol{b}''\right), \tag{5.13}$$

where $\boldsymbol{s}_v \in \mathbb{R}^S$ is the hidden representation of the classifier and $\boldsymbol{W}' \in \mathbb{R}^{S\times C}$, $\boldsymbol{b} \in \mathbb{R}^S, \boldsymbol{W}'' \in \mathbb{R}^{S\times K}, \boldsymbol{b}'' \in \mathbb{R}^K$ are the classifiers parameters. The output of the softmax layer $\boldsymbol{p}_v \in \mathbb{R}^K_{\geq 0}$ define a probability distribution over the output label values. The predicted label y_v is the most probable value according to $\boldsymbol{p}_v$, i.e. $y_v = \arg\max_k \boldsymbol{p}_v[k]$. Following [159], we use dropout [155] with rate 0.5 on both $\boldsymbol{h}_v$ and $\boldsymbol{s}_v$. In SST task, the output layer is applied to all the nodes in the input structure. On the contrary, in TREC task, the output layer is applied only on the root nodes.

The output layer defined for semantic textual similarity is more complicated since it should provide a mechanism to merge the encodings of the two input sentences. Let $(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}})$ the input constituency trees pair, we produce their encodings $\boldsymbol{h}_a$ and $\boldsymbol{h}_b$ by applying a recursive neural model on both constituency trees and taking the hidden state of the roots. Then, we compute the relatedness score $y_{\mathrm{rel}} \in [1, K]$ as in [159]:

$$\begin{aligned} &\boldsymbol{h}_\times = \boldsymbol{h}_a \odot \boldsymbol{h}_b, && \boldsymbol{h}_+ = |\boldsymbol{h}_a - \boldsymbol{h}_b|, && \boldsymbol{s} = \sigma\left(\boldsymbol{W}^+\boldsymbol{h}_+ + \boldsymbol{W}^\times\boldsymbol{h}_\times + \boldsymbol{b}\right), \\ &\boldsymbol{p}_{\mathrm{rel}} = \mathrm{softmax}\left(\boldsymbol{W}'\boldsymbol{s} + \boldsymbol{b}'\right), && y_{\mathrm{rel}} = \boldsymbol{k}^\mathsf{T}\boldsymbol{p}_{\mathrm{rel}}, \end{aligned} \tag{5.14}$$

where $\boldsymbol{s} \in \mathbb{R}^S$ is the hidden representation of the classifier, $\{\boldsymbol{W}^+, \boldsymbol{W}^\times\} \in \mathbb{R}^{C\times S}$, $\boldsymbol{b} \in \mathbb{R}^S, \boldsymbol{W}' \in \mathbb{R}^{S\times K}, \boldsymbol{b}' \in \mathbb{R}^K$ are the classifiers parameters and $\boldsymbol{k}^\mathsf{T} = [1, 2, \ldots, K]$.

Similarly, the entailment label y_{ent} is computed:

$$y_{\mathrm{ent}} = \arg\max_k \boldsymbol{p}_{\mathrm{ent}}[k], \quad \boldsymbol{p}_{\mathrm{ent}} = \mathrm{softmax}\left(\boldsymbol{W}^e\boldsymbol{s} + \boldsymbol{b}^e\right), \tag{5.15}$$

where the vector $\boldsymbol{s}$ is computed as in Eq. (5.14).

All neural models are trained minimising a loss function through gradient descent. In SST, TREC and SICK-E, the loss function is the negative log-likelihood of the observed labels. Following the experimental setting defined in [159], the loss function used in SICK-R is the KL-divergence between the vector $\boldsymbol{p}_{\mathrm{rel}}$ in Eq. (5.14) and a target sparse distribution $\hat{\boldsymbol{p}}_{\mathrm{rel}} \in \mathbb{R}^K$. Such a target distribution is computed starting from the true relatedness score $\hat{y}_{\mathrm{rel}}$:

$$\hat{\boldsymbol{p}}_{\mathrm{rel}}[i] = \begin{cases} \hat{y}_{\mathrm{rel}} - \lfloor \hat{y}_{\mathrm{rel}} \rfloor, & \text{if } i = \lfloor \hat{y}_{\mathrm{rel}} \rfloor + 1, \\ \lfloor \hat{y}_{\mathrm{rel}} \rfloor - \hat{y}_{\mathrm{rel}} + 1, & \text{if } i = \lfloor \hat{y}_{\mathrm{rel}} \rfloor, \\ 0 & \text{otherwise.} \end{cases} \tag{5.16}$$

Note that the following equality holds: $\hat{y}_{\mathrm{rel}} = \boldsymbol{k}^\mathsf{T}\hat{\boldsymbol{p}}_{\mathrm{rel}}$.

		Hyper-parameters Values			
		bs or lr	C	R	S
SST	Binary Sum-LSTM	$\{5, 10, 25\}$	$\{100, 200, 300\}$	-	$\{0, 500, 1000\}$
	Infinite-Sum-LSTM	$\{5, 10, 25\}$	$\{100, 200, 300\}$	-	$\{0, 500, 1000\}$
	Infinite-CP-LSTM	$\{5, 10, 25\}$	$\{100, 200, 300\}$	$\{50, 100, 150\}$	$\{0, 500, 1000\}$
	Infinite-TT-LSTM	$\{10, 25\}$	$\{150, 200, 300\}$	$\{10, 30\}$	$\{0, 500, 1000\}$
	TreeNet	$\{5, 10, 25\}$	$\{100, 200, 300\}$	-	$\{0, 500, 1000\}$
SICK	Binary Sum-LSTM	$\{10, 25, 40\}$	$\{150, 200, 300\}$	-	$\{50, 100, 200\}$
	Infinite-Sum-LSTM	$\{10, 25, 40\}$	$\{150, 200, 300\}$	-	$\{50, 100, 200\}$
	Infinite-CP-LSTM	$\{10, 25, 40\}$	$\{150, 200, 300\}$	$\{30, 50, 100\}$	$\{50, 100, 200\}$
	Infinite-TT-LSTM	$\{25, 40\}$	$\{150, 200, 300\}$	$\{10, 30\}$	$\{50, 100, 200\}$
	TreeNet*	$\{0.001, 0.005, 0.008\}$	$\{150, 200, 300\}$	-	$\{50, 100, 200\}$
TREC	Binary Sum-LSTM	$\{10, 25, 40\}$	$\{150, 200, 300\}$	-	$\{0, 50, 100\}$
	Infinite-Sum-LSTM	$\{10, 25, 40\}$	$\{150, 200, 300\}$	-	$\{0, 50, 100\}$
	Infinite-CP-LSTM	$\{10, 25, 40\}$	$\{150, 200, 300\}$	$\{30, 50, 100\}$	$\{0, 50, 100\}$
	Infinite-TT-LSTM	$\{25, 40\}$	$\{150, 200, 300\}$	$\{10, 30\}$	$\{0, 50, 100\}$
	TreeNet*	$\{0.001, 0.005, 0.008\}$	$\{150, 200, 300\}$	-	$\{0, 50, 100\}$

TABLE 5.3: Hyper-parameters values validated for each neural model on NLP tasks. The models with * validate the learning rate lr rather than the batch size bs.

In all the tasks, we update model parameters using AdaDelta [179] algorithm. The only exception is TreeNet on SICK and TREC datasets, where we use Adam [95] algorithm; we choose this algorithm to be consistent with the TreeNet experimental setting defined in [33]. In the sentiment analysis task, we update also word embedding during the training. Training is performed for at most 100 epochs. Also, we use early-stopping criteria on the validation performance to stop the training if the root accuracy does not increase for 5 consecutive iterations.

For each neural model, we perform a grid-search on the validation set to find the best hyper-parameters configuration. We validate the following model hyper-parameters: the number of hidden-states C, the rank R (if it is used) and the number of hidden units in the output layer S. These hyper-parameters are strictly related to the model expressiveness and the number of parameters required. Moreover, we also validate the hyper-parameters of the learning algorithm that could affect the training. When we use the AdaDelta algorithm, we validate the batch-size bs; the learning rate is determined automatically by the algorithm. When we use the Adam algorithm, we validate the learning rate lr and we fix the batch size to 25. In Table 5.3, we report the hyper-parameters values validated for each neural model.

Performance Measures In sentence classification tasks, we measure model performances using the root accuracy (see Section 4.4).

In semantic textual similarity tasks, we use two different performances measures: the accuracy for entailment labels and the Pearson correlation similarity for relatedness scores (as suggested in [113]). The Pearson correlation similarity ρ between sample

		Tasks				
		SST-5	SST-2	SICK-E	SICK-R	TREC
Unbounded	Infinite-SP-HRTM	17.3 (0.0)	49.9 (0.2)	-	-	31.7 (6.3)
	Infinite-CP-HRTM	17.3 (0.0)	49.7 (0.0)	-	-	18.1 (1.0)
	Infinite-TT-HRTM	17.3 (0.0)	49.7 (0.0)	-	-	18.8 (0.0)
	Infinite-Sum-LSTM	**49.4** (0.6)	85.5 (0.8)	82.6 (0.4)	84.9 (0.2)	**91.9** (1.0)
	Infinite-CP-LSTM	48.3 (0.8)	85.3 (0.3)	**84.2** (0.4)	**86.4** (0.1)	90.0 (0.7)
	Infinite-TT-LSTM	48.2 (0.5)	**86.8** (0.1)	83.9 (0.1)	85.6 (0.2)	90.7 (0.6)
Bin.	Binary Sum-LSTM	**51.5** (0.7)	**87.9** (0.2)	**82.3** (0.5)	84.3 (0.7)	**92.3** (0.8)
	TreeNet	48.4 (1.5)	87.0 (0.5)	81.2 (0.3)	84.5 (0.3)	91.3 (1.1)

TABLE 5.4: Results obtained by infinite models on different NLP tasks. All the values are accuracy except SICK-R, whose score is Pearson's correlation multiplied by 100. Values reported are averaged over three runs (standard deviation in brackets).

points $\{a_1, \dots, a_N\}$ and $\{b_1, \dots, b_N\}$ is defined as [48]:

$$\rho(a, b) = \frac{\sum_{i=1}^{N}(a_i - \tilde{a})(b_i - \tilde{b})}{\sqrt{\sum_{i=1}^{n}(a_i - \tilde{a})^2}\sqrt{\sum_{i=1}^{n}(b_i - \tilde{b})^2}}, \tag{5.17}$$

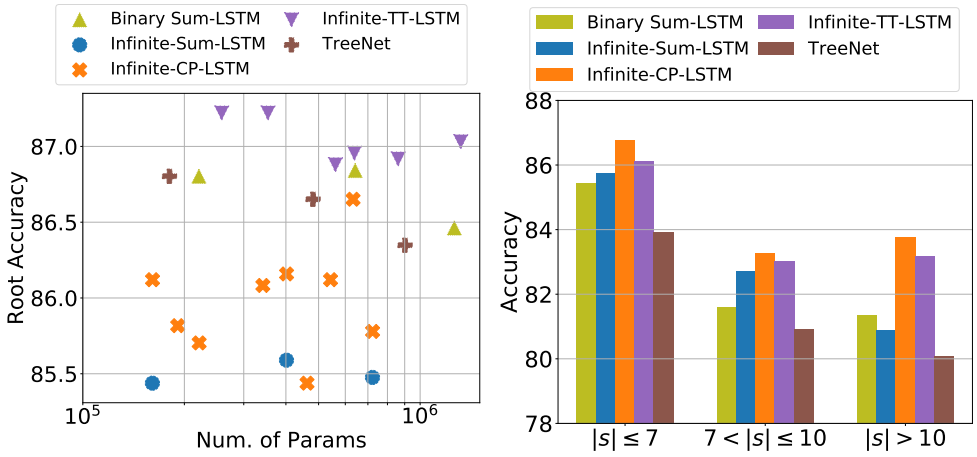
where $\tilde{a} = \frac{\sum_{i=1}^{N} a_i}{N}$ and $\tilde{b} = \frac{\sum_{i=1}^{N} a_i}{N}$ are the sample means. The value of ρ is always between 0 and 1. In the SICK-R task, the Pearson correlation is computed between the true $\{\hat{y}^1_{\text{rel}}, \dots, \hat{y}^N_{\text{rel}}\}$ and the predicted $\{y^1_{\text{rel}}, \dots, y^N_{\text{rel}}\}$ relatedness scores.

Results

In Table 5.4, we report the results obtained by all the evaluated models on all the datasets. All values are averaged over three runs to account for randomisation effect due to model parameter initialisation. The value of the Pearson correlation is multiplied by 100.

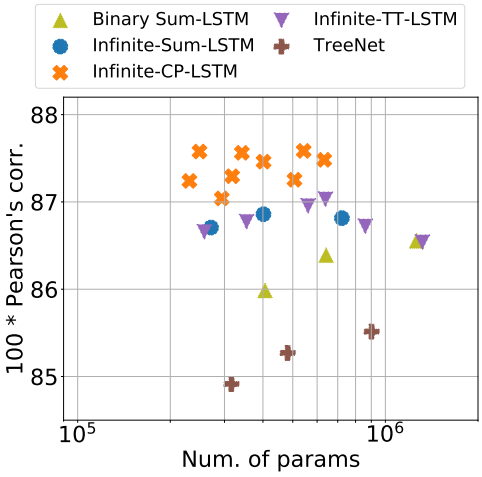
Probabilistic models fail on the sentence classification datasets. We argue that these poor performances are mainly attributable to the training algorithm. The EM algorithm (see Section 2.2.1) maximises the likelihood of the observed data. Nevertheless, the solution which maximises such a likelihood may not correspond to a solution where the root hidden state encodes information of the whole input structure. Due to the inability of probabilistic models to produce meaningful sentence encodings, we do not assess them on the semantic textual similarity task.

The aforementioned behaviour of the EM algorithm is more evident on the NLP tasks due to the presence of word embeddings on leaf nodes. Due to the continuity of the input label, we have defined the multi-variate Gaussian distribution $P(x_v \mid h_v)$ to infer the most probable leaf hidden state h_v given the word embedding x_v (see Section 5.3.3). In this setting, the EM algorithm tries to maximise also the likelihood of the input labels learning the distribution $P(x_v \mid h_v)$. Considering that the number of leaf nodes in a tree with V node is $O(V)$, it is reasonable that the EM algorithm prefer



(A) Validation root accuracy w.r.t. the number of parameters on SST-2.

(B) Test accuracy of neural infinite models on SICK-E w.r.t. the sentence length.



(C) Validation Pearson's correlation w.r.t. the number of parameters on SICK-R.

FIGURE 5.2: Test and validation results obtained by neural models on different NLP tasks.

solutions in which the hidden states are used to maximise the likelihood on leaf nodes rather than to encode sub-phrases meaning.

If we turn our attention on neural models, we notice that results obtained on the SST dataset (both with fine-grained and binary labels) do not show any improvements using unbounded constituency trees. However, the comparison is unfair since the original dataset provides labels on the internal nodes of binary constituency trees (see Section 5.3.3). If we restrict our attention on infinite models, Infinite-TT-LSTM outperforms the other two infinite models on SST-2 reaching an accuracy comparable with TreeNet. Moreover, it requires fewer parameters than other models (see Figure 5.2a).

The results obtained on the SICK dataset show the advantage of combining a rich representation (such as unbounded trees) and a tensor-based state-transition function. In fact, both Infinite-CP-LSTM and Infinite-TT-LSTM outperform all the other models in both the entailment (SICK-E) and relatedness (SICK-R) task. It is

worth to point out that the infinite neural models which leverage tensor decompositions are the only models which benefit from the unbounded representation. In Figure 5.2b, we report the test accuracy of each model on SICK-E with respect to the input length (we consider the maximum between the length of each sentence in the input pairs). Observing the plot, it is clear that most of the models struggle with long sentences. Infinite-CP-LSTM and Infinite-TT-LSTM reach an accuracy of approximately 84%, while the other models stop around 81%. Thus, the unbounded representation of sentences is beneficial especially when the length of input sentences increase. We argue that this behaviour is related to the nodes introduced by the binarisation procedure (see Section 5.3.1), which increase the depth of the constituency trees.

In both SICK-E and SICK-R datasets, the best results are obtained by Infinite-CP-LSTM. This is more evident observing Figure 5.2c, where we show the validation results on SICK-R obtained by all the neural models against the number of parameters they require. We suspect that such performances are due to the inductive bias of CP approximation, which is extremely useful on these datasets. We discuss this hypothesis more in detail in Section 5.3.4.

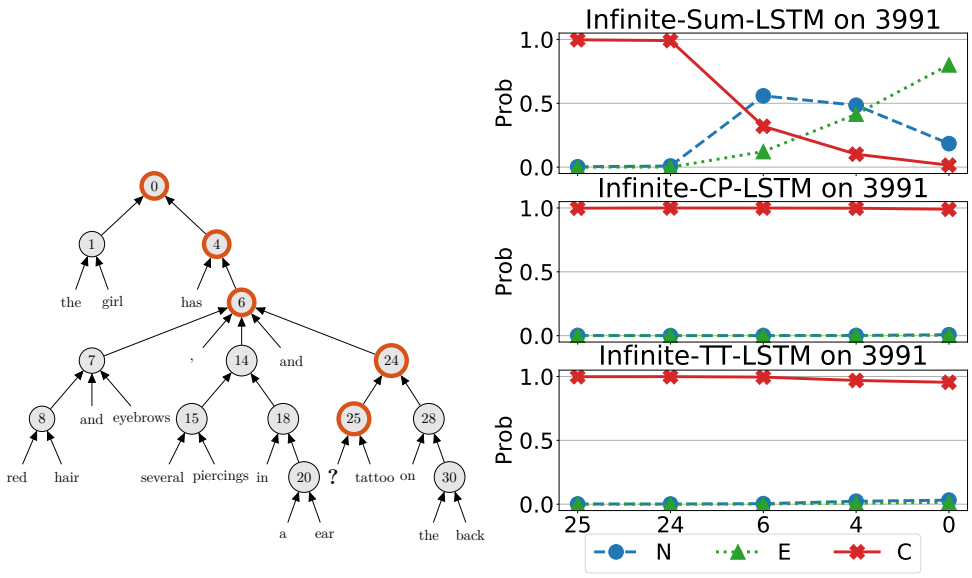
On the TREC dataset, the sum-based state-transition functions seems to be advantageous over the tensor decompositions counterparts. In fact, the sum-based models outperform all the other neural models both on binary and unbounded trees. Also, TreeNet (which is based on summation) reaches results comparable with the ones obtained by Infinite-Sum-LSTM. We argue that the summation is preferable for the question classification task, probably due to the intrinsic characteristics of question sentences.

5.3.4 Qualitative Analysis of Sentences Semantic Entailment

In this section, we analyse in details the prediction of the infinite neural models on the SICK-E dataset. In particular, we report the prediction on the example #3991 of the test set. The input pair is composed of the following sentences:

> A: *The girl has red hair and eyebrows, several piercings in a ear and* ***a tattoo*** *on the back.*
> B: *The girl has red hair and eyebrows, several piercings in a ear and* ***no tattoo*** *on the back.*

The two sentences are exactly the same, unless the sub-phrase *a tattoo* in the sentence A which becomes *no tattoo* in the sentence B. Hence, the expected output is "contradiction". In Figure 5.3a, we plot the constituency tree of the input sentences, indicating with **?** the position where the two sentences differ. Also, we highlight all the nodes that are in the path between the **?** and the root. These are the only nodes which have a different sub-tree in the two sentences. To analyse how the models predict the final label, we study how the prediction changes going up through the structure. In Figure 5.3b, we report the output of the classifier fed with the hidden states pair $(\boldsymbol{h}_v^a, \boldsymbol{h}_v^b)$, where $\boldsymbol{h}_v^a$ ($\boldsymbol{h}_v^b$) is the hidden state of the node v computed by the



(A) Input constituency trees. (B) Model prediction on different nodes.

FIGURE 5.3: Comparison between infinite neural models predictions on the input #3991 taken from the SICK [113] test set.

recursive model on the sentence A (B). On node 25, all the models predict correctly the contradiction. However, going up through the structure, the Infinite-Sum-LSTM changes the prediction to entailment, which will be the final output on node 0. The change of the output label starts at node 6, which is the node where most of the information is aggregated; it seems that the Infinite-Sum-LSTM performs a sort of average which softens the contribution of node 24 (the only one that instead should be taken into account). On the contrary, Infinite-CP-LSTM and Infinite-TT-LSTM propagate correctly the information through the structure. Even if there are sub-phrases which are identical, their contributions do not influence the output. In fact, observing Figure 5.3b, we can notice that the class predicted by Infinite-CP-LSTM and Infinite-TT-LSTM is always "contradiction" in all nodes in the path between **?** and the root.

This example show that neural models benefit of state-transition functions which are able to ignore child contributions, if it is necessary. The CP approximation provides a straightforward mechanism to implements such behaviour thanks to the negation property of the multiplicative operation (see Section 4.4.5).

5.4 Conclusion

In this chapter, we have proposed probabilistic and neural recursive models which are able to learn form unbounded structured data. Such models rely on the combination of tensor decompositions and weight sharing constraints.

The infinite models which leverage the infinite CP approximation define state-transition functions which ignore the child nodes order. This inductive bias can be

deduced by the relation between symmetric tensors and the infinite CP approximation. Moreover, this relation ensure us that every symmetric tensor can be represented by the infinite CP approximation.

On the other hand, the infinite models which leverage the infinite TT approximation are able to define state-transition functions which exploit children order. This inductive bias is intrinsic in the definition of TT decomposition since it define a sequential processing of the child nodes. Moreover, thanks to the weight sharing constraint imposed on the core tensors, such a sequential processing is transformed into a recursive one. As far as we know, in this case there is no connection with any tensor property. Indeed, the recursive processing resembles the definition of recursive models for sequences.

Finally, we have assessed the performances of our infinite models in the NLP domain. While the probabilistic models do not achieve satisfactory results, the infinite neural models based on tensor decompositions outperform the models based on summation on most of the tasks attempted. In particular, we have shown that inductive bias of the CP approximation is useful on semantic textual similarity tasks.

Chapter 6

Unbounded Models for Structured Data

6.1 Introduction

Finding the best hyper-parameters values for a ML model is a difficult challenge. This aspect is extremely relevant in the context of recursive models which leverage tensor decompositions. As we have extensively shown in Section 4, their ability to reduce the model complexity depends on the decomposition rank value, which is a model hyper-parameter.

A common technique to select hyper-parameter values is the *cross-validation* [22], where different values are validated on a sub-set of the training data (i.e. the so-called *validation* set). Then, the best performing value is selected. The drawback of this process is the significant amount of resources required to train and validate multiple configurations.

In this chapter, we study a different approach to select hyper-parameters values. In particular, we aim to introduce two *unbounded* models for structured data which adapt their complexity directly to the input data. The term unbounded refers to their complexity which is not a priori determined by fixing some hype-parameters. Thus, it is (theoretically) unbounded.

In Section 6.2, we propose a Bayesian Non-Parametric (BNP) [128] mixture model to tackle unsupervised tasks on structured data. The term Non-Parametric indicates that the model can rely on a theoretically infinite-dimensional parameter space. Nevertheless, in practice, only a subset of such an infinite space is used. The size of this subset is adapted the complexity of the input data. The term Bayesian refers to the approach used to adapt the model complexity. The Bayesian framework allows estimating this quantity by applying posterior inference. We experimentally assess the proposed model on two clustering tasks on structured domains, showing that mixture models are able to capture global structural properties. Moreover, we assess the computational advantages introduced by using the BNP approach.

In Section 6.3, we propose a Bayesian extension of HOSVD-HRTM which is able to learn the decomposition rank as well as its parameters from data. The underlying idea is to define a prior distribution on the rank values to guide a random search.

This searching strategy is integrated into the learning algorithm, obtaining a model which adapts its complexity on the input data. We validate the effectiveness of the Bayesian approach to learn the decomposition ranks on two different tasks.

6.2 Bayesian Mixture Model for Structured Data Clustering

In the previous chapters, we mainly focus on recursive models in a supervised learning setting where the objective is to produce a (possibly structured) output given the input structure. While the recursive models can also be applied to unsupervised learning tasks, the literature produced in this direction is limited. A notable exception is the seminal paper on a general framework for the unsupervised processing of structured data [80]. Within this class of models, the most relevant contributions are related to the extension of topographic mapping models to handle structured data. This is the case, for instance, of the SOM-SD model [76], extending Kohonen's self-organising maps to structured acyclic data. Extensions of generative topographic mapping to structured data have instead been proposed by [64] and [13], based on top-down and bottom-up approaches, respectively. Despite their unsupervised nature, such models have seldom been used for structured data clustering. The point is that these models are fundamentally solving a different problem than clustering, that is finding a topographic mapping that preserves structural similarities. When applied to clustering, e.g. in [75], the partitioning of samples into clusters is obtained only as a post-processing step, through expert inspection of the projects obtained on the topographic map.

In this section, we introduce a BNP mixture model designed to address the structure clustering problem. Mixture models are generative approaches widely applied in clustering applications for vector data, e.g. the Gaussian mixture model and its evolutions. Here, we propose a mixture model built on the top of probabilistic recursive models. In particular, we choose SP-HRTM as mixture component. Thanks to a Bayesian approach based on Dirichlet processes, we empower the mixture model to handle a theoretically infinite number of components [124]. As a part of its learning procedure, it adapts the number of effectively used components to the complexity of the data.

6.2.1 SP-HRTM for Unsupervised Learning

We have already detailed SP-HRTM in Section 3.3.1 as the first practical approximation of HRTM introduced in the literature [11]. Nevertheless, it is worth to briefly summarise its definition showing how it can be applied in the context of unsupervised learning tasks.

Probabilistic models for structured transduction such as SP-HRTM define a generative process for the output structure $\boldsymbol{\mathcal{Y}}$ which is conditioned on the input

structure $\boldsymbol{\mathcal{X}}$, i.e. $P(\boldsymbol{\mathcal{Y}} \mid \boldsymbol{\mathcal{X}})$. However, in unsupervised tasks, the aim is to learn an unconditioned distribution $P(\boldsymbol{\mathcal{Y}})$. Thus, the probabilistic models can be extended to tackle unsupervised tasks by simply ignoring the input labels. More formally, we should assume that all the input labels are always in a bottom state $\perp$; thus, the model defines the distribution $P(\boldsymbol{\mathcal{Y}} \mid \perp)$, where $\perp$ is a structure with the same skeleton of $\boldsymbol{\mathcal{Y}}$ which has all the node labels equal to $\perp$. Since the input label does not carry any information, we can ignore it.

A formalisation of SP-HRTM without input labels has been already introduced in [12]. Let $\boldsymbol{\mathcal{Y}}$ be an observed structure, SP-HRTM defines the following generative process:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\mathcal{Y}} \mid \theta) = \sum_{\boldsymbol{\mathcal{H}}} P(\boldsymbol{\mathcal{Y}}, \boldsymbol{\mathcal{H}} \mid \theta) = \prod_{\substack{v \in \mathrm{vert}(\boldsymbol{\mathcal{Y}}) \\ |\mathrm{ch}(v)|=0}} \sum_{h_v} P^l(h_v \mid \boldsymbol{a}_l) P(y_v \mid h_v, \boldsymbol{K}) \\ \times \prod_{\substack{v \in \mathrm{vert}(\boldsymbol{\mathcal{Y}}) \\ |\mathrm{ch}(v)>0|}} \sum_{h_v} \sum_{l=1}^{L} P(S_v \mid \boldsymbol{s}) P(h_v \mid h_{lv}, \boldsymbol{U}_l) P(y_v \mid h_v, \boldsymbol{K})\end{aligned} \tag{6.1}$$

where $\theta = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_L, \boldsymbol{K}, \boldsymbol{s}, \boldsymbol{U}_1, \ldots, \boldsymbol{U}_L\}$ are the model parameters and $\boldsymbol{\mathcal{H}}$ represents the set of all the hidden variables. In the following, we briefly summarise each parameter.

The vector $\boldsymbol{a}_l \in \mathbb{R}^C_{\geq 0}$ parametrises the leaf distribution $P^l(h_v)$, where v is a leaf node, l is its position and C is the number of hidden states.

The matrix $\boldsymbol{K} \in \mathbb{R}^{C \times K}_{\geq 0}$ parametrises the output distribution $P(y_v \mid h_v)$ which generates the visible labels. We assume there are K possible output labels.

The parameters $\boldsymbol{s}$ and $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_L$ are related to the SP approximation of the state-transition distribution. In particular, $\boldsymbol{s} \in \mathbb{R}^L_{\geq 0}$ parametrises the switching parent variable S_v; $\boldsymbol{U}_l \in \mathbb{R}^{C \times C}_{\geq 0}$ parametrise the distribution $P(h_v \mid h_{vl})$ which regulates the relation between the hidden state of v and the hidden state of its l-th child node. As usual, L indicates the maximum structure out-degree.

6.2.2 Mixture of SP-HRTMs

A finite mixture model is able to approximate complex distributions through an appropriate choice of its components to represent the local area of the truth distribution [114]. Thus, we introduce a finite mixture model whose components are SP-HRTMs to better represent complex distributions over labelled structures.

Model Definition

A finite mixture model is obtained by combining multiple generative models called *mixture components*. The combination is obtained through a hidden random variable, called *mixture variable*.

Since we are introducing a finite mixture model, the number of components is fixed and it is represented by the hyper-parameter T. In our model, all mixture

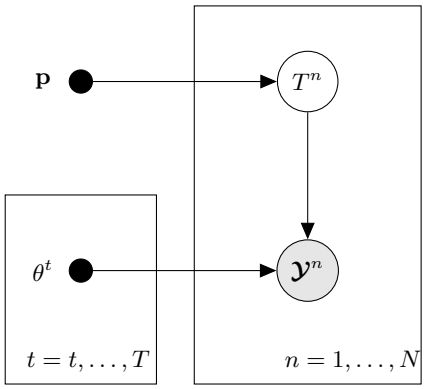


FIGURE 6.1: Graphical model of Mix-SP-HRTM, where black-point nodes identify model parameters.

components are SP-HRTM without input labels (see Section 6.2.1), each of them with different parameters $\boldsymbol{\theta} = \{\theta^1, \ldots, \theta^T\}$. To better understand how the mixture of SP-HRTM (Mix-SP-HRTM) represents the data, it is useful to summarise the underlying generative process for the n-th structure $\boldsymbol{\mathcal{Y}}^n$:

$$\begin{aligned} \boldsymbol{\mathcal{Y}}^n \mid t^n, \boldsymbol{\theta} &\sim P(\boldsymbol{\mathcal{Y}}^n \mid \theta^{t_n}) \\ t^n \mid \mathbf{p} &\sim \text{Discrete}(p_1, \ldots, p_T). \end{aligned} \tag{6.2}$$

The term t^n indicates the latent class associated to the observed structure $\boldsymbol{\mathcal{Y}}^n$, i.e. the index of the mixture component used to generate the structure. Hence, θ^{t^n} represents the model parameters of the t^n-th mixture component. The value $P(\boldsymbol{\mathcal{Y}}^n \mid \theta^{t^n})$ is the likelihood of $\boldsymbol{\mathcal{Y}}^n$ according to the t^n-th component and it is modelled using a SP-HRTM (see Eq. (6.1)). The latent class is drawn from a categorical distribution, which is the distribution of the mixture variable. In Figure 6.1, we represent the graphical model which describes this process: for the sake of clarity, the whole structure $\boldsymbol{\mathcal{Y}}^n$ is denoted as a single variable.

Learning procedure

Learning Mix-SP-HRTM parameters has two objectives: the first one is to learn the parameters of all the mixture components $\boldsymbol{\theta}$; the second one is to learn the mixing distribution $\mathbf{p}$. Both objectives can be reached through the EM algorithm (see Section 2.2.1).

In the E-step, we aim to compute the posterior distributions of all the hidden variables given the visible ones. First, we should observe that two mixture components are completely independent given the latent class: the only way to exchange information among components is through the latent class. Hence, each conditional independence assumption exploited to derive the E-step of in a single mixture component still holds in the mixture model. Thus, we can rely on the SP-HRTM upward-downward algorithm (see Appendix D) to compute the posteriors of the t^n-th mixture component

$P(\mathcal{H}^n \mid \mathcal{Y}^n, t^n, \theta^{t^n})$, where the conditioning over the latent class t^n is explicitly introduced. This value still depends on a hidden variable, i.e. the latent class t^n. Nevertheless, by applying the chain rule, we write the posterior distribution as:

$$P(\mathcal{H}^n, t^n, \mid \boldsymbol{\mathcal{Y}}^n) = P(\mathcal{H}^n \mid t^n, \boldsymbol{\mathcal{Y}}^n) P(t^n \mid \boldsymbol{\mathcal{Y}}^n), \tag{6.3}$$

where we omit the parameter θ^{t^n} since it is implicit in the latent class.

The term $P(t^n \mid \boldsymbol{\mathcal{Y}}^n)$ represents the posterior of the latent class, which can be easily obtained as:

$$P(t^n \mid \boldsymbol{\mathcal{Y}}^n) = \frac{P(\boldsymbol{\mathcal{Y}}^n \mid t^n) P(t^n)}{P(\boldsymbol{\mathcal{Y}}^n)}, \tag{6.4}$$

which completes the E-step definition, summarised in Algorithm 1. Note that the value $P(\boldsymbol{\mathcal{Y}}^n \mid t^n)$, i.e. the likelihood of the observed data according to the t^n mixture components, is computed during the upward pass of the upward-downward algorithm.

The M-step updates the component parameters $\boldsymbol{\theta}$: it is derived by the straightforward application of the formula used for a single SP-HRTM (see Appendix D) using the posterior computed in Eq. (6.3). An additional rule to update the latent class distribution $\mathbf{p}$ is also needed:

$$\boldsymbol{p}[t] = \frac{\sum_{n=1}^{N} P(T^n = t \mid \boldsymbol{\mathcal{Y}}^n)}{N}. \tag{6.5}$$

From the computational complexity point of view, the introduction of the mixture increases the computational complexity in time to $O(T \times \mathfrak{T}_{\text{up-down}})$, where $\mathfrak{T}_{\text{up-down}}$ is the time complexity of the upward-downward algorithm. The computational complexity in space has the same behaviour: it becomes $O(T \times \mathfrak{S}_{\text{SP-HRTM}} + T)$, where $\mathfrak{S}_{\text{SP-HRTM}}$ is the space required to store the parameters of a single SP-HRTM. The last term T is the space required to store the mixing distribution, which can be neglected.

Algorithm 1 E-step for Mix-SP-HRTM

Require: A labelled structure $\boldsymbol{\mathcal{Y}}^n$, T different SP-HRTM with parameters $\theta^1 \dots \theta^T$ and a mixture distribution $\boldsymbol{p}$.
  for t=1 **to** T **do**
    $postH[t] = \text{UP-DOWN}(\boldsymbol{\mathcal{Y}}^n, \theta^t)$
    $lk[t] = \text{LIKELIHOOD}(\boldsymbol{\mathcal{Y}}^n, \theta^t)$
    $postP[t] = lk[t] \times \boldsymbol{p}[t]$
  end for
  $postP = NORMALISE(postP)$
  for t=1 **to** T **do**
    $postH[t] = postH[t] \times postP[t]$
  end for
  return $(postH, postP)$

6.2.3 Bayesian Non-parametric Mixture of SP-HRTM

Setting the correct number of components in a finite mixture model is not obvious and a variety of techniques have been developed [114]. Our goal is to build a Bayesian

Non-Parametric mixture of SP-HRTM (BNP-SP-HRTM), which allows an infinite number of mixture components: in our case, each component is, again, a SP-HRTM with different parameters. In the followings, we formally define BNP-SP-HRTM as well as its learning algorithm.

Model Definition

The BNP extension of a finite mixture model relies on Dirichlet Processes (DPs) [56]. The generation of an observed structure $\boldsymbol{\mathcal{Y}}^n$ can be described as follows [124]:

$$\begin{aligned} \boldsymbol{\mathcal{Y}}^n \mid \zeta^n &\sim F(\theta^n) \\ \theta^n \mid G &\sim G \\ G &\sim DP(G_0, \gamma). \end{aligned} \tag{6.6}$$

The distribution $F(\theta^n)$ represents the mixture component (i.e. a SP-HRTM) with parameters θ^n drawn from G, which is itself distributed according to a DP with concentration parameter γ and base measure G_0. The value G_0 is the expected value of the DP and it represents the prior distribution for the mixture component parameters. For the sake of simplicity, we have ignored the dependency between the function F and the mixture component parameters and the hyper-parameters for the prior G_0. These will be stated more in detail in the remainder of the section.

For our purposes, it is convenient to derive the unbounded model taking the limit as T goes to infinity of a Mix-SP-HRTM with T components [124]. Before taking the limit, we define explicitly the prior distributions of the Mix-SP-HRTM parameters (i.e. the function G_0). Since all the model parameters follow a categorical distribution, we use the Dirichlet distribution as prior distributions since it is the conjugate prior of the categorical distribution. Thus, we obtain:

$$\begin{aligned} \boldsymbol{\mathcal{Y}}^n \mid t^n, \boldsymbol{\theta} &\sim P(\boldsymbol{\mathcal{Y}}^n \mid \theta^{t^n}) \\ t^n \mid \mathbf{p} &\sim \text{Discrete}(p_1, \dots, p_T) \\ \boldsymbol{a}_l[:] &\sim \text{Dirichlet}(\alpha_a, \dots, \alpha_a), \\ \boldsymbol{U}_l[c,:] &\sim \text{Dirichlet}(\alpha_u, \dots, \alpha_u), \\ \boldsymbol{K}[c,:] &\sim \text{Dirichlet}(\alpha_k, \dots, \alpha_k), \\ \boldsymbol{s} &\sim \text{Dirichlet}(\alpha_s, \dots, \alpha_s) \\ \boldsymbol{p} &\sim \text{Dirichlet}(\gamma/T, \dots, \gamma/T). \end{aligned} \tag{6.7}$$

Since we are using a flat Dirichlet distribution, we have a single hyper-parameter for each prior distribution. Hence, the model hyper-parameters are $\{\alpha_a, \alpha_u, \alpha_k, \alpha_s, \gamma\}$: the α terms are related to the SP-HRTM priors (i.e. are parameters of G_0) while the γ term is the concentration parameter of the DP.

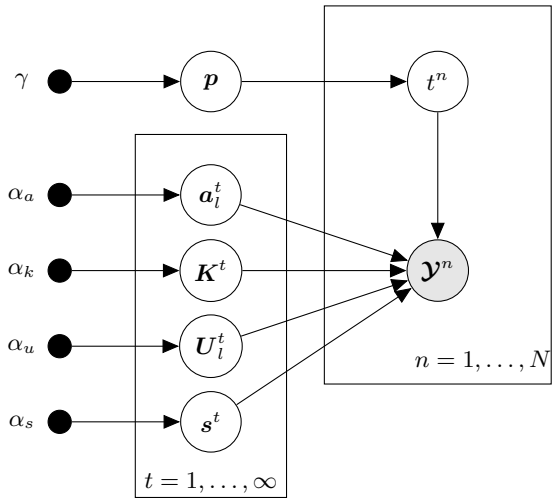


FIGURE 6.2: Graphical model of BNP-SP-HRTM.

Learning parameters

Computing the exact posterior expectation becomes unfeasible when the model is extended with a DP prior, i.e. when $T \to \infty$. However, such expectation can be estimated using Monte Carlo methods [124]. In particular, a Gibbs sampling algorithm can be applied to the model described in Eq. (6.7), integrating out the mixing proportions $\mathbf{p}$. The idea is to iteratively sample the latent class t^n for each data point and update the parameters θ for each mixture component, taking into account only those data points assigned to each mixture. Even if there is an infinite number of components, we are able to execute this algorithm since we deal only with the mixture components that are currently associated with some observations. By definition, there is only a finite number of observation and thus also the number of mixture components must be finite. In the next sections, we denote by *active components* the mixture components that are currently associated with at least one observation.

In the first step, the Gibbs sampler assigns a latent class to each structure $\boldsymbol{\mathcal{Y}}^n$. The latent class of $\boldsymbol{\mathcal{Y}}^n$ is sampled from a conditional distribution which considers (i) the latent class associated to all the other structures and (ii) the likelihood of $\boldsymbol{\mathcal{Y}}^n$ according to the mixture components [124]. Thus, we obtain:

$$P(t \mid \boldsymbol{t}^{-n}, \boldsymbol{\mathcal{Y}}^n, \boldsymbol{\theta}) = \begin{cases} \dfrac{N_{-n,t}}{Z} P(\boldsymbol{\mathcal{Y}}^n \mid \theta^t) & \text{if } N_{-n,t} > 0 \\ \dfrac{\gamma}{Z} \displaystyle\int P(\boldsymbol{\mathcal{Y}}^n \mid \theta) dG_0(\theta) & \text{otherwise} \end{cases} \tag{6.8}$$

where $N_{-n,t}$ is the number of structures (except $\boldsymbol{\mathcal{Y}}^n$) which are already assigned to the t-th class. The value $\boldsymbol{t}^{-n}$ indicates the latent class of all structures in the dataset except $\boldsymbol{\mathcal{Y}}^n$, while Z is a normalising constant to ensure that the above probability sum to one.

Equation (6.8) states that the probability of assigning a class t to a structure is

proportional to the number of structures that are already assigned to it (i.e. $N_{-n,t}$). Nevertheless, there is a non-zero probability of assigning the i-th structure to a new component: unfortunately, we cannot consider explicitly all the other components since there is an infinite number of them. The solution is to integrate over all the possible mixture component parameters (i.e. all the possible mixture components). The integral is taken over the function $G_0(\theta)$ since it represents the prior for SP-HRTM parameters. The integral can be solved analytically due to the conjugacy between parameter distributions and their prior: the result is the likelihood of $\boldsymbol{\mathcal{Y}}^n$ according to a SP-HRTM whose parameters are uniform distributions since we use flat Dirichlet distributions as priors. When a new class is sampled, we must create a new mixture component. The new parameters are sampled from the prior distribution $G_0(\theta)$. During the inference procedure, it can also happen that a latent class is no longer assigned to any structures. From Equation (6.8), it follows there is a zero probability to assign such class again. Hence, we can remove the corresponding latent class.

The second step of the inference procedure requires estimating new parameters θ for all the existing mixture components. Each component updates its parameters considering only the structures that are assigned to it during the first step. The updates can be performed by applying the EM procedure for SP-HRTM on the subset of the dataset assigned to each component. The only modification required is in the M-step, which is extended to consider also the prior. Assuming that all the SP-HRTM distribution are categorical distribution, we can consider the Dirichlet priors by simply add $\alpha - 1$ to each counting table. The whole Gibbs sampling method is summarised in Algorithm 2.

Algorithm 2 Gibbs sampling method for BNP-SP-HRTM

Require: A dataset of labelled structure $\hat{D} = \{\boldsymbol{\mathcal{Y}}^1, \dots, \boldsymbol{\mathcal{Y}}^N\}$, a set of SP-HRTM parameters $\boldsymbol{\theta} = \{\theta^1 \dots \theta^T\}$, a random assignment $\mathbf{t} = \{t^1, \dots, t^N\}$
$S^j = \{n \mid t^n = j\} \quad \forall j \in \{1, T\}$
repeat
  for $n = 1$ **to** N **do** ▷ Sample step
    $S^{t^n} = S^{t^n} \setminus \{n\}$
    if $S^{t^n} = \emptyset$ **then** ▷ Remove t^n
      $\boldsymbol{\theta} = \boldsymbol{\theta} \setminus \{\theta^{t^n}\}$
      $T = T - 1$
    end if
    $t^n = \text{SAMPLING}(\boldsymbol{t}^{-n}, \boldsymbol{\mathcal{Y}}^n, \boldsymbol{\theta})$ ▷ Eq. (6.8)
    if t^n is new **then** ▷ Create t^n
      $\theta^{t^n} \sim G_0$
      $\boldsymbol{\theta} = \boldsymbol{\theta} \cup \{\theta^{t^n}\}$
      $T = T + 1$
      $S^{t^n} = \emptyset$
    end if
    $S^{t^n} = S^{t^n} \cup \{n\}$
  end for
  for t=1 **to** T **do** ▷ Update step
    $\theta^t = \text{EM-SP-HRTM}(\theta^t, S^t, G_0)$
  end for
until stopping criteria

Again the computational complexity (both in time and space) increases linearly with respect to the number of active components T, when compared to the simple SP-HRTM model. It is worth highlighting that, in this case, the value of T is dynamically updated during the learning procedure.

6.2.4 Experimental results

To validate the proposed approaches, we empirically assess their ability to recognise clusters in tree-structured data. Evaluating the clustering quality is not trivial and multiple indexes have been defined [141]. In the following experiments, we use the Silhouette index [143] to assess the clustering quality. The Silhouette index is an *internal* measure and therefore it can be computed without any additional knowledge on data (e.g. the true clustering). However, its computation requires the definition of a suitable distance metric among data points. Here we compute the distance between two trees using the Ruzicka distance [48] on their representative matrices. Let $\boldsymbol{Z}^n$ be the representative matrix for a tree $\boldsymbol{\mathcal{Y}}^n$, its element $z^n[l,j]$ counts how many times the label j appears in a node in the l-th position. Thus, given the representative matrices for two trees $\boldsymbol{\mathcal{Y}}^n$ and $\boldsymbol{\mathcal{Y}}^m$ (i.e. $\boldsymbol{Z}^n$ and $\boldsymbol{Z}^m$, respectively) the Ruzicka distance is defined as:

$$d_R(\boldsymbol{\mathcal{Y}}^n, \boldsymbol{\mathcal{Y}}^m) = 1 - \frac{\sum_l \sum_j \min(z^n[l,j], z^m[l,j])}{\sum_l \sum_j \max(z^n[l,j], z^m[l,j])}. \tag{6.9}$$

Note that the Ruzicka distance defined does not take into account the topologies of the $\boldsymbol{\mathcal{Y}}^n$ and $\boldsymbol{\mathcal{Y}}^m$. Nevertheless, as it will be clearer in the next paragraphs, this measure is able to capture some differences among structures that belong to different clusters in the considered tasks.

Given the definition of a suitable distance measure, the Silhouette index of a given tree $\boldsymbol{\mathcal{Y}}^n$, is computed considering the distance between $\boldsymbol{\mathcal{Y}}^n$ and both elements that are inside and outside its cluster; its value is always between -1 (worst clustering) and 1 (best clustering). Mathematically speaking, let $\boldsymbol{\mathcal{Y}}^n$ an element in the t-th cluster, the Silhouette index is given by:

$$s(\boldsymbol{\mathcal{Y}}^n) = \frac{b(\boldsymbol{\mathcal{Y}}^n) - a(\boldsymbol{\mathcal{Y}}^n)}{\max(a(\boldsymbol{\mathcal{Y}}^n), b(\boldsymbol{\mathcal{Y}}^n))}, \tag{6.10}$$

where $a(\boldsymbol{\mathcal{Y}}^n)$ is the average distance of $\boldsymbol{\mathcal{Y}}^n$ and its cluster t, while $b(\boldsymbol{\mathcal{Y}}^n)$ is the minimum distance between $\boldsymbol{\mathcal{Y}}^n$ and a cluster $t' \neq t$. The distance between a tree $\boldsymbol{\mathcal{Y}}^n$ and a cluster t is computed as the average distance between $\boldsymbol{\mathcal{Y}}^n$ and all trees $\boldsymbol{\mathcal{Y}}^m \in t$. In formula, assuming $\boldsymbol{\mathcal{Y}}^n \in t$, we obtain:

$$a(\boldsymbol{\mathcal{Y}}^n) = \frac{1}{|C|} \sum_{\boldsymbol{\mathcal{Y}}^m \in C} d_R(\boldsymbol{\mathcal{Y}}^n, \boldsymbol{\mathcal{Y}}^m), \qquad b(\boldsymbol{\mathcal{Y}}^n) = \min_{C' \neq C} \frac{1}{|C'|} \sum_{\boldsymbol{\mathcal{Y}}^{m'} \in C'} d_R(\boldsymbol{\mathcal{Y}}^n, \boldsymbol{\mathcal{Y}}^{m'}), \tag{6.11}$$

where $d_R(\cdot,\cdot)$ is the Ruzicka distance defined in Eq. (6.9).

We evaluate our model on two clustering tasks; the first is a controlled dataset while the latter deals with real-world data. The MATLAB code implementing the models and the experiments conducted can be found on a public GitLab repository.[1]

Synthetic dataset

The goal of the first experiment is to assess whether the mixture of hidden trees (both finite and infinite) offers an advantage with respect to a single SP-HRTM in terms of cluster identification. To this end, we test all models (SP-HRTM, Mix-SP-HRTM, and BNP-SP-HRTM) on a synthetic clustering problem denoted as ASYMM. The ASYMM dataset contains ternary trees (i.e. L=3), comprising left-asymmetric, symmetric and right-asymmetric tree, hence defining three different clusters. A tree is defined as left-asymmetric (right-asymmetric) if the number of nodes in the leftmost (rightmost) position is greater than the number of nodes in the opposite position. In a symmetric tree, the number of nodes is almost equivalent for each position.

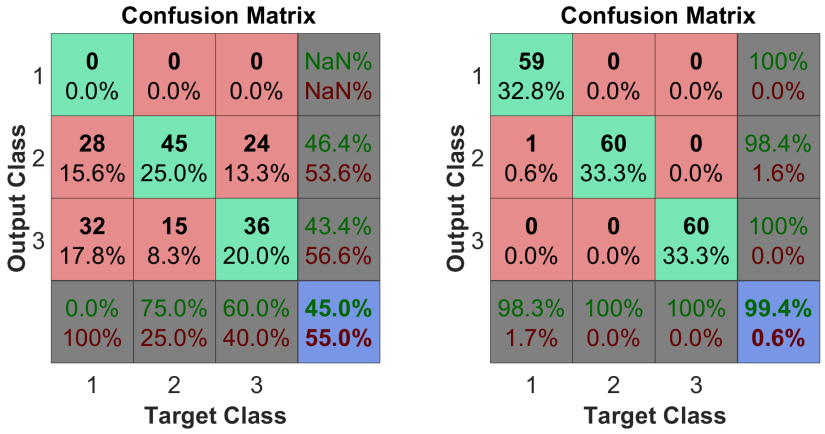
The dataset is generated through a top-down recursive procedure: starting from the root, child nodes are generated according to a distribution which indicates how likely is to generate a node in each position. If new nodes are generated, the same procedure is recursively applied until the whole tree is generated. The procedure ends when a maximum number of nodes are generated. This scheme is used to obtain all the three structure types: for each type, a proper distribution to generate child nodes is used. The label of each node encodes structural information since it represents the number of children: therefore the label goes from 0 (i.e. no child nodes) to 3 (i.e. a child node in each position). Moreover, each tree type is generated by setting a different maximum number of nodes in order to add another structural peculiarity. In particular, left-asymmetric trees are the smallest one, while the right-asymmetric are the biggest ones. Symmetric trees have a number of nodes which is between the characteristic sizes of left and right imbalanced trees. Finally, we generate 780 trees (260 for each type) and split them in training set (600 trees, 200 for each type) and test set (180 trees, 60 for each type).

All models (SP-HRTM, Mix-SP-HRTM, and BNP-SP-HRTM) are trained in an unsupervised setting, i.e. the true class is not available at training time. For each model, different configurations are tested, changing the number of hidden states (i.e. C), the number of mixtures (i.e. T), and the prior hyper-parameters. Thanks to a preliminary analysis, we have noticed that some hyper-parameters of BNP-SP-HRTM do not affect the solution too much. Therefore, to reduce the number of configurations to test, we have used the same value for all the prior hyper-parameters (i.e. α_p, α_k, α_u, α_s): we refer to this value with the letter α. Also, we have fixed the concentration parameters $\gamma = 10$. For a fair comparison, each training algorithm is executed for a maximum of 30 iterations.

[1] https://github.com/danielecastellana22/Mixture-HTMM

SP-HRTM	$C = 3$	$C = 5$	$C = 7$
Root sampling	**0.03** (0.00)	−0.02 (0.03)	−0.08 (0.02)
Mix-SP-HRTM	$T = 3$	$T = 5$	$T = 7$
$C = 2$	0.41 (0.01)	0.43 (0.03)	0.46 (0.05)
$C = 4$	0.45 (0.05)	**0.47** (0.08)	**0.47** (0.05)
$C = 6$	0.46 (0.05)	0.45 (0.05)	**0.47** (0.06)
BNP-SP-HRTM	$\alpha = 1$	$\alpha = 1.5$	$\alpha = 2$
$C = 2$	0.36 (0.23)	0.45 (0.08)	0.45 (0.07)
$C = 4$	0.43 (0.08)	**0.51** (0.00)	**0.51** (0.00)
$C = 8$	0.33 (0.00)	**0.51** (0.00)	**0.51** (0.00)

TABLE 6.1: Mean Silhouette index over 5 runs (std in brackets) on the ASYMM dataset.



(A) SP-HRTM, $C = 3$. (B) BNP-SP-HRTM, $C = 8$ and $\alpha = 2$.

FIGURE 6.3: Confusion matrices for the synthetic dataset using SP-HRTM and BNP-SP-HRTM.

At test time, SP-HRTM assigns a class to each tree by sampling the posterior of the root node while Mix-SP-HRTM samples the posterior of the mixture variable. BNP-SP-HRTM cannot directly sample from the posterior since this would be intractable; however, the Gibbs sampler introduced in Section 6.2.3 can be used to approximate the latent class assignment (skipping the parameters optimisation step). During the test phase, we limit to 10 the number of iterations of the Gibbs sampler.

In Table 6.1, we report the mean and standard deviation (in brackets) of the Silhouette index for each configuration over five runs. The advantage obtained by introducing a mixture is clear: the single SP-HRTM reaches the best performance of 0.03, which is far from the best one obtained from both Mix-SP-HRTM and BNP-SP-HRTM. Instead, the performance obtained by both mixture models is closer to the Silhouette index computed on the ground truth, that is 0.51. In Figure 6.3, we report two confusion matrices, obtained using BNP-SP-HRTM and SP-HRTM to show the benefits of mixture models.

The poor performance of the single SP-HRTM highlights that the posterior of the root node does not contain much information on the characteristics of the whole

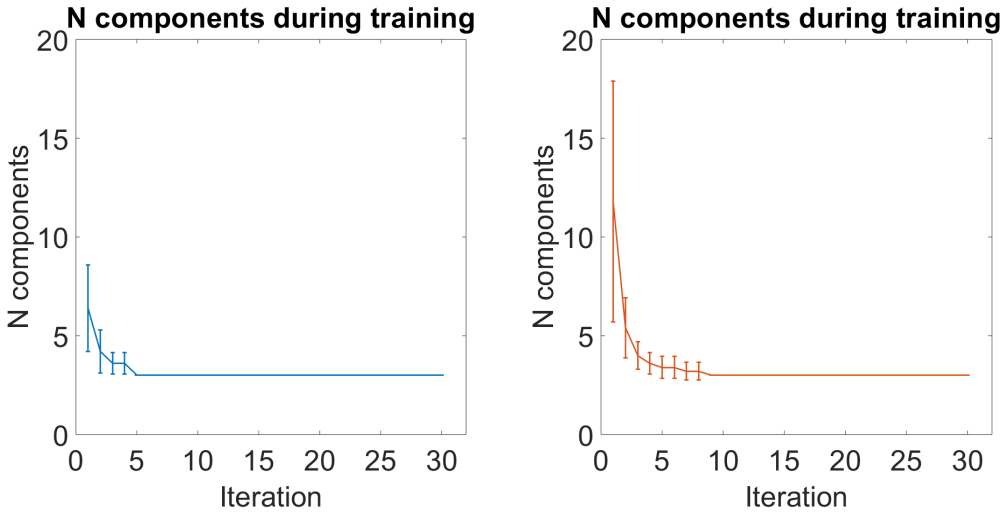
structure. As we have already pointed out in Section 5.3.3, this behaviour is attributable to the EM algorithm that is not constrained to store global information about the input structure into the root hidden state. On the contrary, it merely maximises the likelihood of the observed data. Nevertheless, introducing a mixture model and sampling the mixture variable rather than the root hidden variable, we obtain far better results. The mixture variable contains by definition information about the whole input structure since its state encodes the structure class. Thanks to the generative process defined by the mixture model (see Section 6.2.2), the EM algorithm benefits (in terms of higher likelihood) learning solutions which are able to store global information in the mixture variable state. In other words, the EM algorithm prefer solutions which cluster the input data.

Even if the best results obtained by Mix-SP-HRTM and BNP-SP-HRTM are similar, there are some key differences. First, we should notice that the infinite model reaches the best performance with zero standard deviation, i.e. the model performed the same in each run. Also, Mix-SP-HRTM performs better when there are more components than the real number of clusters: most of them are not used by the model. On the other hand, BNP-SP-HRTM is able to find the true number of clusters. In Figure 6.4, we plot the mean (and standard deviation) number of components during the training for two different configurations of BNP-SP-HRTM. In the first iterations, the model explores the solution space creating a high number of components (with different parameters); then, the model starts adapting the best components to the data, throwing away unused ones. After a few iterations, it reaches a total of 3 component, which is the actual number of clusters. The plot also shows a different behaviour between the two configurations: this aspect is examined in depth in Section 6.2.4.

This behaviour also affects the time required for both training and testing since the time complexity for both mixture models depends linearly on the number of components: hence, unused mixture components slow down the inference procedure. In Figure 6.5, we show the training time required by both mixture models on the synthetic dataset. For fairness, both training algorithms are executed without model-specific code optimisation. The plot confirms our initial intuition: during the first iterations, BNP-SP-HRTM is slower than Mix-SP-HRTM due to the high number of active components. However, after few iterations, BNP-SP-HRTM becomes faster as the Mix-SP-HRTM with three components. According to the results in Table 6.1, Mix-SP-HRTM and BNP-SP-HRTM perform better when the respective hyper-parameters are set to $T = 7$ and $\alpha = 2$: comparing their training time in Figure 6.5, the benefit of the Bayesian non-parametric extension is clear.

Real-world dataset

Previous experiments show the ability of both Mix-SP-HRTM and BNP-SP-HRTM to cluster labelled trees in a completely unsupervised fashion. The goal of this experiment is to assess the clustering performance of Mix-SP-HRTM and BNP-SP-HRTM on a



(A) $C = 8$ and $\alpha = 2$. (B) $C = 4$ and $\alpha = 1.5$.

FIGURE 6.4: Number of active components during the training (averaged over 5 runs) for two different configurations of BNP-SP-HRTM on the synthetic dataset.

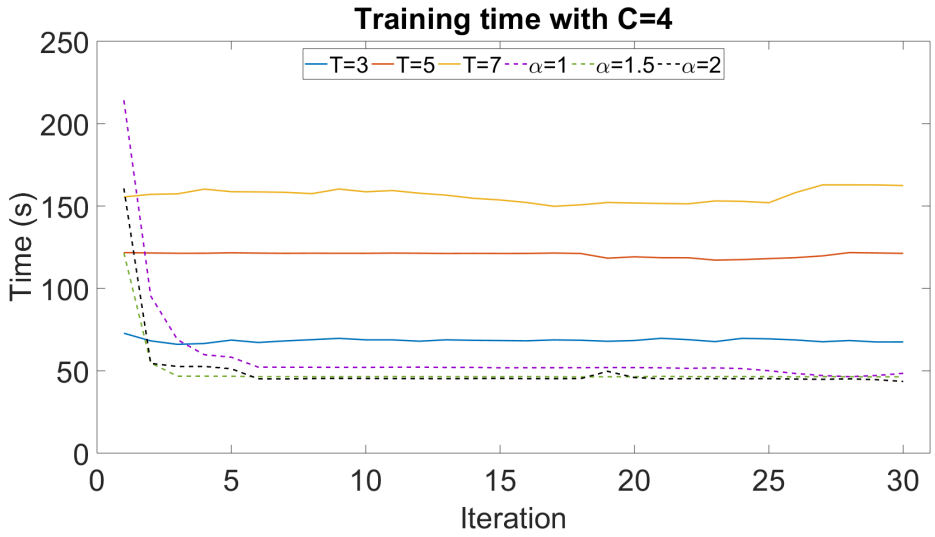


FIGURE 6.5: Time spent for a single training iteration by Mix-SP-HRTM and the BNP-SP-HRTM. Solid lines denote Mix-SP-HRTM results for different T choices. Dashed lines refer to BNP-SP-HRTM under different choices for the α hyper-parameter. For the sake of clarity, the number of hidden states is fixed to 4 in both models.

real-world dataset. Due to the poor performances obtained in the previous experiment, we do not evaluate SP-HRTM.

We choose a dataset taken from the INEX 2005 competition [46]. It is based on the (m-db-s-0) corpus, comprising 9631 XML-formatted documents represented as trees with maximum output degree $L = 32$ and labelled by 11 thematic categories, which represents the different clusters. Node labels represent XML tags: there are 366 possible labels. The dataset is split in training set (4820 trees) and test set (4811 trees) [46].

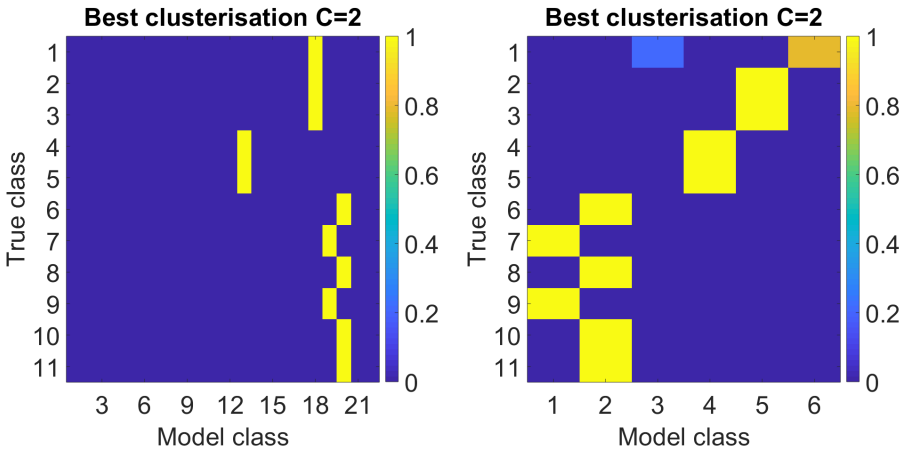
Again, we test multiple configurations for each model. In particular, in Mix-SP-HRTM, we vary the number of hidden states $C \in [2, 4, 8]$ and the number of mixture component $T \in [6, 11, 22]$. In BNP-SP-HRTM, we vary the number of hidden states $C \in [2, 4, 8]$ and the hyper-parameter of the SP-HRTM prior $\alpha \in [1, 1.2, 1.5, 2]$. As in the previous experiment, we fix the value of the concentration parameter, i.e. $\gamma = 10$. Each configuration is trained for a maximum of 30 iterations, while the BNP-SP-HRTM test procedure is executed for a maximum of 10 iterations.

In Table 6.2, we report the mean and standard deviation (in brackets) of the Silhouette index computed, for each configuration, over five training-test runs. The advantage of the infinite model is not clear, even if it reaches the best performance on the INEX2005 dataset. Rather than comparing only the performance results, it is interesting to compare the resulting clusters produced by each model to obtain such a performance. In Figure 6.6, we report clusters obtained using the best configuration of both models. The plot shows how trees in each true class (on the y-axis) are distributed with respect to the model-predicted clusters (on the x-axis). The clustering obtained using Mix-SP-HRTM (see Figure 6.6a) is made up of only 4 active clusters (even if there are 22 components): the first cluster contains all trees with ground-truth class labels $\{1, 2, 3\}$, the second cluster contains all trees with labels $\{4, 5\}$, the third cluster contains all trees with labels $\{6, 8, 10, 11\}$ and the last one contains all trees with labels $\{7, 9\}$. The clustering obtained using the BNP-SP-HRTM (see Figure 6.6b) are almost the same, but there are two main differences. The first one is the number of clusters used, that is only 6 since the components with no data are thrown away during the training, thus reducing their impact on computational complexity. The second difference is that the model creates two new clusters to contain trees with ground-truth label 1: even if the model creates a spurious cluster, it is able to learn the difference between trees from category 1 and trees from all other categories.

The clustering produced by both models exploit the structural and label information contained in INEX2005 trees. In Figure 6.7, we report a similarity measure between categories in the INEX2005 training set. The similarity between two categories t' and t'' is computed taking the mean of the Ruzicka similarity (see Eq. (6.9)) between all the t' trees and all the t'' trees. The plot shows clearly that categories with high similarity are the ones that are clustered together by our models.

Mix-SP-HRTM	$T = 6$	$T = 11$	$T = 22$	
$C = 2$	0.12 (0.01)	0.13 (0.07)	**0.20** (0.04)	
$C = 4$	0.13 (0.09)	0.17 (0.02)	0.15 (0.02)	
$C = 8$	0.08 (0.00)	0.11 (0.05)	0.17 (0.06)	
BNP-SP-HRTM	$\alpha = 1$	$\alpha = 1.2$	$\alpha = 1.5$	$\alpha = 2$
$C = 2$	0.15 (0.02)	0.15 (0.05)	0.19 (0.02)	**0.21** (0.04)
$C = 4$	0.07 (0.04)	0.20 (0.04)	0.16 (0.07)	0.18 (0.03)
$C = 8$	0.05 (0.10)	0.15 (0.05)	0.13 (0.02)	0.15 (0.06)

TABLE 6.2: Mean Silhouette index over 5 runs (std in brackets) on the INEX05 dataset. In bold the best result for each model.



(A) Mix-SP-HRTM, $C = 2$ and $T = 22$. (B) BNP-SP-HRTM, $C = 2$ and $\alpha = 2$.

FIGURE 6.6: Clustering obtained by Mix-SP-HRTM and BNP-SP-HRTM using the best (model selected) configuration on the INEX05 dataset.

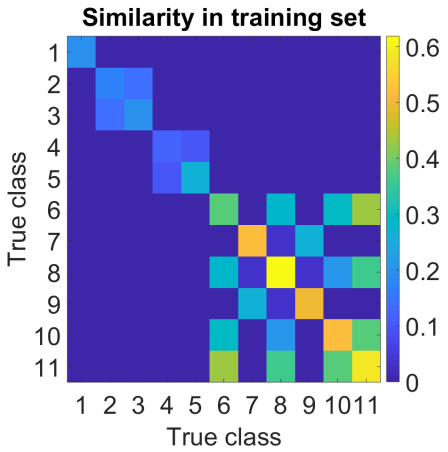


FIGURE 6.7: Ruzicka similarity between categories on the INEX2005 training set. Blue colours denote low similarity while yellow indicates a high similarity.

Mix-SP-HRTM	$T = 6$	$T = 11$	$T = 22$	
$C = 2$	1.60 (0.55)	2.00 (0.71)	3.40 (0.55)	
$C = 4$	2.00 (1.00)	2.20 (0.84)	1.80 (0.45)	
$C = 8$	1.20 (0.45)	2.00 (0.00)	2.80 (0.45)	
BNP-SP-HRTM	$\alpha = 1$	$\alpha = 1.2$	$\alpha = 1.5$	$\alpha = 2$
$C = 2$	44.20 (14.79)	33.00 (13.69)	8.80 (3.11)	4.60 (1.95)
$C = 4$	23.80 (26.36)	11.60 (4.83)	3.20 (1.30)	2.40 (1.67)
$C = 8$	45.80 (35.81)	5.80 (3.03)	2.40 (0.55)	1.80 (0.84)

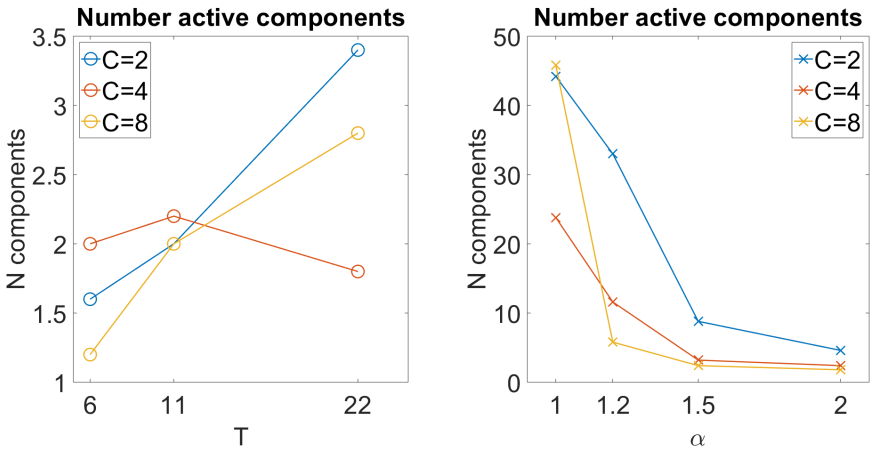
TABLE 6.3: Mean number of non-empty clusters over 5 runs (std in brackets) on INEX05 dataset.

The importance of hyper-parameters

The experiments reported so far highlight how important is choosing the right value of hyper-parameters in order to obtain satisfactory results using both mixture models. In this section, we analyse the results obtained on INEX05 to emphasise the effects of each hyper-parameter. In particular, we study the effect of the hyper-parameters on the number of clusters discovered by the models. In Table 6.3, we report the mean and standard deviation of the number of clusters for each Mix-SP-HRTM and BNP-SP-HRTM configuration over 5 runs.

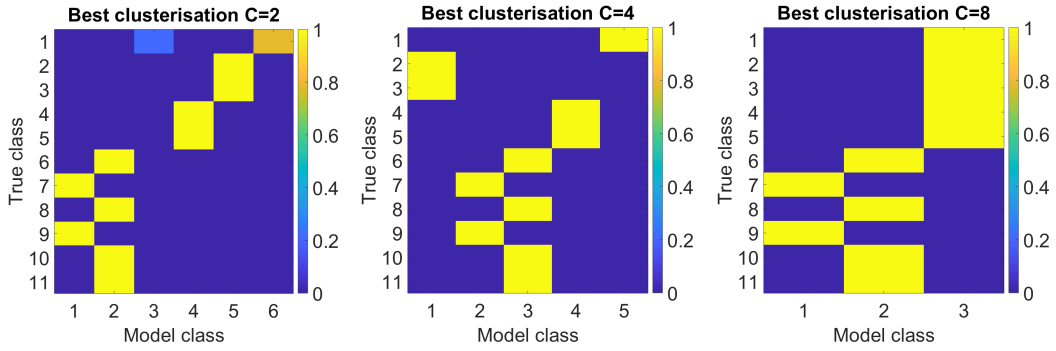
Mix-SP-HRTM is characterised by two hyper-parameters: the number of hidden states C and the number of mixture components T. By increasing the number of hidden states, we obtain more expressive SP-HRTMs as mixture components. Therefore, with higher values of C, the model tends to use fewer components since each component can be expressive enough to represent different clusters. The number of components T indicates how many SP-HRTM components are used by the model. By taking a deeper look at the results in Table 6.2, it is clear that increasing the number of components helps to obtain better performances. However, even if a high number of components is selected, the number of clusters being identified is always small (see Table 6.3). We argue that increasing the value of T allows more exploration in the solution space: each component has a random configuration that can be suitable or not to describe the data. By creating more components, it is more likely to guess a better initialisation. In Figure 6.8a, we plot the average number of clusters over 5 runs for each Mix-SP-HRTM configuration. Observing the plot, it is clear that higher values of T lead to higher numbers of active clusters. It is also visible the influence of C: the configuration with $C = 2$ has more active components than the configuration with $C = 8$.

While the complexity of BNP-SP-HRTM model is also controlled by the number of hidden states C, there is no hyper-parameter explicitly determining the number of mixture components. However, this is strictly correlated to the choice of the α values. In fact, α determines how strong is our prior belief on SP-HRTM parameters: a stronger belief means that components will not adapt to the data too much (preventing over-fitting), while weaker beliefs lead to a completely data-driven solution. Hence, higher values of α tend to create solutions with fewer clusters, while a small value



(A) Mix-SP-HRTM (B) BNP-SP-HRTM

FIGURE 6.8: Number of active components as a function of hyper-parameters for both mixture models on INEX05.



(A) $C = 2$. (B) $C = 4$. (C) $C = 8$.

FIGURE 6.9: Best clusters obtained using BNP-SP-HRTM with different values of C on INEX05 dataset.

has the opposite effects. The value of the hyper-parameters C has the same influence described before on Mix-SP-HRTM. In Figure 6.8b, we plot the average number of active components for each BNP-SP-HRTM configuration, averaged on 5 runs. The effect of the choice of α is evident: the number of active components reduces from more than 20 to around 5, independently of the value of C. The effect of the choice of C is also clear: the number of components obtained with $C = 2$ is greater than the one obtained with $C = 4$, which is greater than the one obtained with $C = 8$. Furthermore, the influence of C is evident when reporting the best clustering obtained for each C value (see Figure 6.9): selecting $C = 8$, all trees in the first five categories are merged together. On the other hand, by selecting $C = 2$, we do not have a SP-HRTM expressive enough to represent trees in the first category: hence, the model uses two components to represent them.

6.3 Bayesian HOSVD for Structured Data Labelling

The application of tensor decompositions to build new powerful recursive models has been already discussed in Chapter 4. In these models, the decomposition rank is a hyper-parameter which regulates the trade-off between the model complexity and the model expressiveness. Thus, selecting a proper rank value is a key designing choice to obtain models which achieve the desired performances. To this end, we propose a Bayesian extension of HOSVD-HRTM which is able to learn decomposition ranks during the training procedure. We refer to this model as Bayesian-HOSVD-HRTM.

Bayesian approaches have been already combined with tensor decompositions to perform multi-way data analysis, e.g. [35, 157, 86, 69]. The prior distributions are usually imposed on the decomposition factors and they can be used to prefer low-rank solution. For example, in [180], the authors impose a sparse prior distribution over CP factor matrices to control the rank of the decomposition.

In this section, we adopt a different approach. In fact, rather than imposing a specific property on the factors prior distribution, we explicitly model the decomposition rank as a random variable whose value is learned from data. This approach has been introduced in [176] to perform high-dimensional classification tasks. Also, it has been applied to approximate higher-order Markov chain [146] and higher-order HMM [145].

Finally, it is worth mentioning two BNP tensor decomposition which allows an infinite value of the rank. In [135], the authors propose a BNP model for the collaborative filtering task. The underlying idea is to apply a Dirichlet process for each class of items (i.e. mode of the tensor) to cluster its values; then, the interactions among items is modelled taking into account only the clusters they belong to. Similarly, in [175] propose a BNP tensor factorisation based on the Gaussian process. In this case, the unboundness is obtained by applying a (possibly infinite) feature map to the latent factors.

6.3.1 Bayesian HOSVD Model

While Bayesian-HOSVD-HRTM is a Bayesian extension of HOSVD-HRTM proposed in Section 4.2.3, they have a slightly different parametrisation. For the sake of simplicity, in the Bayesian formalisation, we ignore the input labels.

Thus, Bayesian-HOSVD-HRTM define the following distributions:

- the distribution $P^l(h_v \mid \boldsymbol{p}^l)$, where $l = \text{pos}(v)$ indicates the position of the node v with respect to its siblings. The prior distribution is parametrised by the vector $\boldsymbol{p}^l \in \mathbb{R}^C_{\geq 0}$, for each position $l \in [1, L]$;

- the distribution $P(r_{vl} \mid h_{vl}, \boldsymbol{U}_l)$, where $\boldsymbol{U}_l \in \mathbb{R}^{(C+1)\times \ddot{R}_l}_{\geq 0}$ is the mode matrix of the HOSVD approximation associated to the l-th dimension;

- the distribution $P(h_v \mid r_{v1}, \dots, r_{vL}, \underline{\boldsymbol{G}})$, where $\underline{\boldsymbol{G}} \in \mathbb{R}^{\ddot{R}_1 \times \cdots \times \ddot{R}_L \times C}_{\geq 0}$ is the core tensor of the HOSVD approximation. Note that in this formulation we have

collapsed the core tensor and the mode matrix associate to the last dimension in a single distribution;

- the emission distribution $P(y_v \mid h_v, \boldsymbol{K})$. Since we assume discrete output labels, it is a categorical distribution parametrised by the matrix $K \in \mathbb{R}_{\geq 0}^{C \times K}$.

The values $\boldsymbol{r} = \{\ddot{R}_1, \ldots, \ddot{R}_L\}$ represent the decomposition rank along the first L dimension and they determine the size of the core tensor $\underline{\boldsymbol{G}}$. From a probabilistic point of view, these quantities determines the number of states of the rank variables $R_{v1}, \ldots, R_{vL}$. As we have shown in Section 4.2.3, each state of the l-th rank variable can be interpreted as a cluster of the hidden states of the l-th child node. Thus, the value $\ddot{R}_l$ measures how strong is the dependence between the hidden state H_{vl} and its hidden parent state H_v. A value of $\ddot{R}_l = C$, means that it is important to know the exact child state H_{vl} in order to determine the parent state. On the contrary, if $\ddot{R}_l = 1$, the child state does not affect the parent one: no matter the true value of H_{vl}, it will collapse in the unique state available for R_{vl}.

The Bayesian nature of the model arises considering each parameter as a random variable which follows a suitable prior distribution. Since all the parameters are categorical distributions, it is natural to define the priors using their conjugate distribution: the Dirichlet distribution. Thus, the prior distribution of $\underline{\boldsymbol{G}}$ is defined as [145]:

$$\underline{\boldsymbol{G}}[r_{v1}, \ldots, r_{vL}, :] \sim \text{Dirichlet}(\alpha \boldsymbol{\lambda_0}) \tag{6.12}$$

$$\boldsymbol{\lambda_0} \sim \text{Dirichlet}(\alpha_0/C, \ldots, \alpha_0/C), \tag{6.13}$$

where the value α and α_0 are hyper-parameters which regulate the shape of the categorical distributions in $\underline{\boldsymbol{G}}$.

On the same line, we also specify a prior distribution for the other parameters:

$$\boldsymbol{p}_l \sim \text{Dirichlet}(\gamma + n_{l,1}, \ldots, \gamma + n_{l,C}) \tag{6.14}$$

$$\boldsymbol{K}[c, :] \sim \text{Dirichlet}(\beta + n_{c,1}, \ldots, \beta + n_{c,K}), \tag{6.15}$$

$$\tag{6.16}$$

where the value γ and β are hyper-parameters which regulate the shape of $P^l(h_v \mid \boldsymbol{p}^l)$ and $P(y_v \mid h_v, \boldsymbol{K})$, respectively.

The core of Bayesian-HOSVD-HRTM is its ability to estimate rank values $\boldsymbol{r}$ during the learning procedure. To this end, it also considers rank values as realisations of random variables. The distributions of such variables are very important since they force the model to focus either on compression (i.e. most of the values in $\boldsymbol{r}$ are 1) or data representation (i.e. most of the values in $\boldsymbol{r}$ are C). Moreover, we can easily insert prior knowledge selecting appropriate distributions: for example, we can build distributions to introduce our prior belief that there are children positions more informative than others.

We decide to use the following position-independent prior:

$$P(\ddot{R}_l) = e^{-\varphi \ddot{R}_l}, \tag{6.17}$$

$$\sum_{l=1}^{L} \mathbb{I}[\ddot{R}_l \neq 1] \in [L_{\min}, L_{\max}]. \tag{6.18}$$

The value φ is a hyper-parameter which regulates the distribution decay. The hyper-parameters $L_{\min}$ and $L_{\max}$ denote the minimum and the maximum number of important positions, respectively. The utility of $L_{\min}$ and $L_{\max}$ is twofold: (1) we can avoid that the parent hidden state is completely independent of all the child hidden states (by setting $L_{\min} \geq 1$), and (2) we can control the worst case space-complexity (which is $O(C^{L_{\max}+1})$).

By varying the value of $\ddot{R}_l$, we change the dimension of the core tensor $\underline{\boldsymbol{G}}$. In fact, when a rank value increases, we increase the number of states of the corresponding rank variable. Thus, we have to define new categorical distributions which regulate the relations between the new rank variable state and the parent hidden state. Thanks to the Bayesian formulation, these new categorical distributions are obtained by sampling from the $\underline{\boldsymbol{G}}$ prior distribution.

6.3.2 Parameters Learning and Rank Estimation

Our goal is to develop a learning algorithm which is able to learn the rank values in $\boldsymbol{r}$ along with the model parameters. Unfortunately, this estimation worsens the learning problem. To this end, we introduce an approximation on the mode matrices $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_L$. The approximation consists in adding the constraint $\tilde{\boldsymbol{U}}_l[j, r] = \{0, 1\}$ to each entry of the mode matrices [145]. This new constraint makes the clustering performed by mode matrices deterministic. In fact, we can associate each state to a single cluster and define each cluster as $\mathcal{C}_l^r = \{j \mid \tilde{\boldsymbol{U}}_l[j, r] = 1\}$. Thus, the approximated mode matrices $\tilde{\boldsymbol{U}}_l$ performs a *hard clustering*.

The approximation above allows us to develop a Gibbs sampling algorithm for fitting the unknown quantities from data. For the sake of simplicity, we discuss the learning algorithm assuming that the training set contains only one structure $\boldsymbol{\mathcal{Y}}$. The extension to a generic i.i.d. training set is straightforward. The procedure comprises the following steps [177]:

1. update all hidden states variables $\mathcal{H}$ and all rank variables $\mathcal{R}$;
2. update the approximation size $\boldsymbol{r}$ and the mode matrices $\tilde{\boldsymbol{U}}_1, \ldots, \tilde{\boldsymbol{U}}_L$;
3. update the model parameters $\theta = \{\boldsymbol{p}_l, \underline{\boldsymbol{G}}, \boldsymbol{\lambda_0}, \boldsymbol{K}\}$.

In the first step, we perform a Simulated Annealing [96] update of the latent variables. Given the current values of $(\mathcal{H}, \mathcal{R}, \boldsymbol{r}, \theta)$, we compute the new values of $\mathcal{H}', \mathcal{R}'$ through an ancestor sampling procedure [17] starting from the leaf nodes. In particular, we sample the new $\mathcal{H}'$ fixing the old values $\mathcal{R}$. Then, we use $\mathcal{H}'$ to sample

Algorithm 3 Sample latent states $\mathcal{H}', \mathcal{R}'$

for all $v \in \text{vert}(\boldsymbol{\mathcal{Y}})$ **do** ▷ Iterate over nodes following topological order
  if $|\text{ch}(v)| = 0$ **then** ▷ Sample leaf nodes hidden state
    $h'_v \sim \boldsymbol{p}^l$, where $l = \text{pos}(v)$
  else ▷ Sample internal nodes hidden state
    for $l = 1$ **to** L **do** ▷ Sample rank variables
      $r'_{vl} \sim \tilde{\boldsymbol{U}}_l[h'_{vl}, :]$ ▷ $h'_{vl} = \perp$ if the child does not exist
    end for
    $h'_v \sim \underline{\boldsymbol{G}}[r'_{v1}, \ldots, r'_{vL}, :]$
  end if
end for

the new values $\mathcal{R}'$. More details are given in Algorithm 3. The new values $\mathcal{H}', \mathcal{R}'$ are accepted with the following probability [145]:

$$\min\left\{\left[\frac{\prod_v \underline{\boldsymbol{G}}[r'_{v1}, \ldots, r'_{vL}, h'_v]}{\prod_v \underline{\boldsymbol{G}}[r_{v1}, \ldots, r_{vL}, h_v]} \frac{\prod_v \underline{\boldsymbol{G}}[r'_{v1}, \ldots, r'_{vL}, h_v]}{\prod_v \underline{\boldsymbol{G}}[r_{v1}, \ldots, r_{vL}, h'_v]}\right]^{1/\mathcal{T}(m)}, 1\right\}, \tag{6.19}$$

where $\mathcal{T}(0)$ and $\mathcal{T}(m) = \max\{\mathcal{T}_0^{1-m/m_0}, 1\}$ denotes the initial and the current annealing temperature, respectively; m is the current iteration number and m_0 is the iteration at which the temperature reduces to one.

In the second step, we perform a Stochastic Search for Variable Selection [63] to update rank values. The vector $\boldsymbol{r}$ is updated choosing a random position l. Then, we decide to do an increase (or decrease) move on the position l by a coin toss. The increasing move consists in adding a new cluster for the l position. Hence, we randomly select a cluster $\mathcal{C}_l^r$ and we randomly split it into two clusters. The splitting process consists in randomly select a set of element in the cluster $\mathcal{C}_l^r$ and to move them in the new cluster $\mathcal{C}_l^{r'}$. In practice, it is implemented modifying the hard clustering $\tilde{\boldsymbol{U}}_l$: for every state c which is moving from cluster r to cluster r', we set $\tilde{\boldsymbol{U}}_l[c, r] = 0$ and $\tilde{\boldsymbol{U}}_l[c, r'] = 1$. On the contrary, the decrease move merges together two randomly selected clusters $\mathcal{C}_l^r$ and $\mathcal{C}_l^{r'}$. Again, the merging operation is implemented modifying the hard clustering $\tilde{\boldsymbol{U}}_l$: for every state c in the cluster r', we set $\tilde{\boldsymbol{U}}_l[c, r'] = 0$ and $\tilde{\boldsymbol{U}}_l[c, r] = 1$.

Finally, we should guarantee that the constraint Eq. (6.18) is satisfied: Algorithm 4 provides more details on the process. The new values $\boldsymbol{r}'$ are accepted with probability [145]:

$$\min\left\{\left[\frac{\mathcal{L}(\boldsymbol{r}')}{\mathcal{L}(\boldsymbol{r})} \frac{\prod_{l=1}^L P(\ddot{R}'_l)}{\prod_{l=1}^L P(\ddot{R}_l)}\right]^{1/\mathcal{T}(m)}, 1\right\}, \tag{6.20}$$

where $\mathcal{T}(m)$ is the current annealing temperature which is computed as in the previous step. The prior $P(\ddot{R}_l)$ is defined in Eq. (6.17), while the marginal likelihood is given by [145]:

$$\mathcal{L}(\boldsymbol{r}) = \prod_{(r_1, \ldots, r_L)} \frac{\mathbf{B}(\alpha\boldsymbol{\lambda_0}[1] + n_{r_1, \ldots, r_L, 1}, \ldots, \alpha\boldsymbol{\lambda_0}[C] + n_{r_1, \ldots, r_L, C})}{\mathbf{B}(\alpha\boldsymbol{\lambda_0}[1], \ldots, \alpha\boldsymbol{\lambda_0}[C])}, \tag{6.21}$$

Algorithm 4 Sample size vector $\boldsymbol{r}$

$l \sim Uniform(L)$ ▷ Choose the position
$v \sim Uniform(2)$ ▷ Random move. 1 increase, 2 decrease
if $\ddot{R}_l = 1$ **then** ▷ Must do increase move
  $v = 1$
end if
if $\ddot{R}_l = C$ **then** ▷ Must do decrease move
  $v = 2$
end if
if $v = 1$ **then** ▷ Do increase move
  $\ddot{R}_l = \ddot{R}_l + 1$
  Randomly split a random cluster, modifying $\tilde{\boldsymbol{U}}_l$
else ▷ Do decrease move
  $\ddot{R}_l = \ddot{R}_l - 1$
  Merge two random clusters, modifying $\tilde{\boldsymbol{U}}_l$
end if
if $\sum_{p=1}^{L} \mathbb{I}[\ddot{R}_p \neq 1] > L_{\max}$ **then** ▷ Check constraints
  Randomly select a position l' s.t. $\ddot{R}_{l'} > 1$
  Do the decrease move on position l'
  if $\sum_{p=1}^{L} \mathbb{I}[\ddot{R}_p \neq 1] > L_{\max}$ **then**
    Remove the increase move on l
  end if
end if
if $\sum_{p=1}^{L} \mathbb{I}[\ddot{R}_p \neq 1] < L_{\min}$ **then** ▷ Check constraints
  Randomly select a position l' s.t. $\ddot{R}_{l'} = 1$
  Do the increase move on position l'
end if

where the function $\mathbf{B}(\cdot)$ represents the multivariate Beta function and the value $N_{r_1,\dots,r_L,c}$ counts how many times the joint configuration $(R_{v1} = r_1, \dots, R_{vL} = r_L, H_v = c)$ appears in the input structure.

In the last step, we update the model parameters sampling from their posteriors:

$$\boldsymbol{p}_l \sim \text{Dirichlet}(\gamma + N_{l,1}, \dots, \gamma + N_{l,C}) \tag{6.22}$$

$$\boldsymbol{K}[c,:] \sim \text{Dirichlet}(\beta + N_{c,1}, \dots, \beta + N_{c,K}) \tag{6.23}$$

$$\underline{\boldsymbol{G}}[c_1, \dots, c_L, :] \sim \text{Dirichlet}(\alpha\boldsymbol{\lambda_0}[1] + N_{r_1,\dots,r_L,1}, \dots, \alpha\boldsymbol{\lambda_0}[C] + N_{r_1,\dots,r_L,C}), \tag{6.24}$$

where $N_{l,c} = \sum_{\text{ch}(v)=\emptyset} \mathbb{I}[H_v = c \wedge \text{pos}(v) = l]$ and $N_{c,k} = \sum_v \mathbb{I}[H_v = c \wedge Y_v = k]$. The sampling of the base distribution $\boldsymbol{\lambda_0}$ requires a more complex procedure which is outlined in Algorithm 5.

6.3.3 Experimental Analysis

We evaluate the proposed Bayesian model on two different tasks: a classification task on XML tree-data and a labelling task on a synthetic dataset. In both tasks, we use two measures to evaluate the model performance: the *accuracy*, which assess the correctness of model's answers, and the *entropy*, which measures the amount of uncertainty in them. For the accuracy measures, higher values are better; for

Algorithm 5 Sample vector $\boldsymbol{\lambda_0}$

for all $(r_1, \dots, r_L, c)$ **do**
  for $p = 1$ **to** $n_{r_1,\dots,r_L,c}$ **do**
    $\boldsymbol{t}[p] \sim Bernoulli\left(\frac{\alpha\boldsymbol{\lambda_0}[c]}{p-1+\alpha\boldsymbol{\lambda_0}[c]}\right)$
  end for
  $\boldsymbol{m}_{r_1,\dots,r_L}[c] = \sum_p \boldsymbol{t}[p], \qquad \forall c \in \{1, C\}$
end for
$\boldsymbol{m_0}[c] = \sum_{r_1,\dots,r_L} \boldsymbol{m}_{r_1,\dots,r_L}[c], \qquad \forall c \in \{1, C\}$
$\boldsymbol{\lambda_0} \sim \mathrm{Dir}(\alpha_0/C, \cdots + \boldsymbol{m_0}[1], \alpha_0/C + \boldsymbol{m_0}[C])$

the entropy, lower values are better. All the reported results are averaged over five executions, to account for randomisation effects due to initialisation.

The MATLAB code implementing the models and the experiments conducted can be found on a public repository.[2]

Classification Task

The classification task consists in predicting the class a tree-structured sample belongs to. We test both the SP-HRTM and Bayesian-HOSVD-HRTM on two datasets taken from the INEX 2005 (INEX05) and INEX 2006 (INEX06) competition [46]. The INEX05 dataset has been already introduced in Section 6.2.4. It contains 9631 XML-formatted documents represented as trees with maximum output degree $L = 32$ and assigned to 11 different cluster. Node labels represent 366 different XML tags. The dataset is split in training set (4820 trees) and test set (4811 trees) [46]. The INEX06 is composed of 12107 XML-formatted documents representing scientific articles, each from one of 18 different IEEE journals which represent the different cluster. Again, the node labels represent different 65 XML tags and maximum output degree $L = 66$. The dataset is split in training set (6053 trees) and test set (6054 trees) [46].

On both datasets, we train a single model for each class. Each model is trained on the elements in the training set associated with the same model class; at test time, we compute the likelihood according to each model and we assign the class of the model which scores the highest sample likelihood. This setting is very similar to the unsupervised approach proposed in Section 6.2; nevertheless, in this case, the clusters are explicitly defined by the output classes. In this way, we avoid determining the class of the whole structure on the root posterior.

For all the models, we set a flat prior on model parameters. Also, we set $\varphi = 2$ (as in [145]), $L_{\min} = 1$, and $L_{\max} = 5$ for the Bayesian-HOSVD-HRTM. The values assigned to $L_{\min}$ and $L_{\max}$ allow to bound the space complexity of the model to $O(C^6)$.

To evaluate the impact of the hidden state size, we vary the number of hidden states $C \in \{2, 4, 6, 8, 10\}$. Both training algorithms are executed for 100 iterations.

[2] https://github.com/danielecastellana22/HOSVD-HTMM

		Accuracy (%)		Entropy (%)	
		SP-HRTM	Bayesian-HOSVD-HRTM	SP-HRTM	Bayesian-HOSVD-HRTM
INEX05	$C = 2$	87.55 (4.13)	**91.41** (3.62)	34.19 (5.80)	**31.99** (10.37)
	$C = 4$	90.68 (5.70)	**93.65** (1.64)	28.92 (5.16)	**25.00** (4.13)
	$C = 6$	93.81 (1.15)	**95.10** (0.27)	24.30 (2.90)	**20.91** (0.93)
	$C = 8$	93.15 (1.69)	**94.28** (2.27)	23.59 (1.29)	**23.12** (6.35)
	$C = 10$	93.30 (3.09)	**95.21** (0.17)	21.77 (1.64)	**20.60** (0.46)
INEX06	$C = 2$	21.44 (4.54)	**27.94** (2.62)	287.40 (7.24)	275.28 (2.07)
	$C = 4$	24.84 (3.15)	**29.83** (3.06)	281.23 (3.46)	279.82 (7.11)
	$C = 6$	25.57 (2.12)	**30.65** (2.36)	277.99 (2.81)	283.47 (5.34)
	$C = 8$	26.43 (2.47)	**26.48** (2.84)	278.17 (1.12)	292.36 (5.56)
	$C = 10$	22.89 (3.33)	**26.94** (2.77)	289.40 (6.28)	291.03 (7.12)

TABLE 6.4: Average accuracy and entropy over 5 runs (std in brackets) on INEX05 and INEX06 dataset.

In Table 6.4, we report the accuracy and the entropy obtained on the test set. The results show that the Bayesian-HOSVD-HRTM always outperforms the SP-HRTM, both in accuracy and in entropy: the difference is higher when the number of hidden states is small. These results are quite surprising if we consider that the Bayesian-HOSVD-HRTM considers at most 5 child nodes among 32 (we set $L_{\max} = 5$); on the contrary, the SP-HRTM always considers the contribution of all child nodes. Hence, we deduce that not all the child nodes contain useful information for the tree-classification and therefore can be ignored.

The results obtained on the INEX06 dataset (see Table 6.4) are similar to the ones obtained on the INEX05 dataset: the Bayesian-HOSVD-HRTM always reaches a higher accuracy than the SP-HRTM. On the contrary, the entropy values are high for both models due to the intrinsic difficulty of the INEX06 dataset [12].

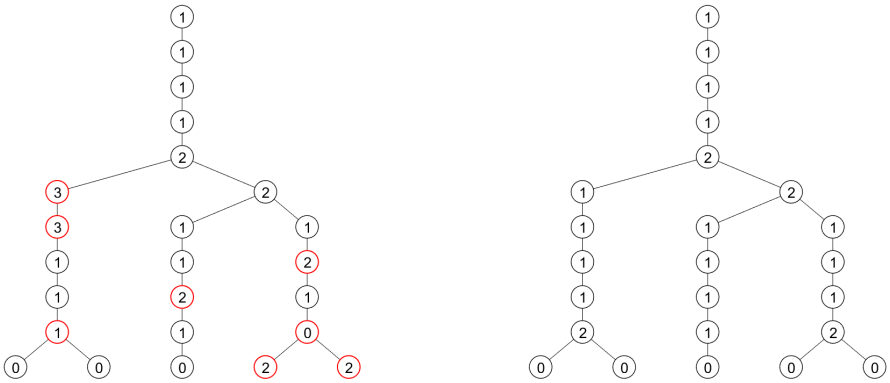
Labelling Task

The labelling task consists in predicting the visible labels associated with the nodes of a given tree structure. We test both the SP-HRTM and the Bayesian-HOSVD-HRTM on the controlled dataset ASYMM which we have already introduced in Section 6.2.4. The ASYMM dataset contains ternary trees (i.e. L=3), comprising left-asymmetric, symmetric and right-asymmetric tree, where the label of each node encodes structural information since it represents the number of children: therefore the label goes from 0 (i.e. no child nodes) to 3 (i.e. a child node in each position).

We train a SP-HRTM and a Bayesian-HOSVD-HRTM with $C = 10$ hidden states. In these experiments, we do not test multiple hyper-parameters configurations: for both models, we use flat priors to generate the initial probability distribution. Also, we set $\varphi = 2$ (as in [145]), $L_{\min} = 1$, and $L_{\max} = 3$ for the Bayesian-HOSVD-HRTM. The values assigned to $L_{\min}$ and $L_{\max}$ allow to consider all the child positions. Both training algorithms are executed for 100 iteration. At test time, both models generate the output labels given an input structure without labels. In Table 6.5, we report the accuracy and the entropy obtained by both models for each output label.

	Accuracy (%)		Entropy (%)	
	SP-HRTM	Bayesian-HOSVD-HRTM	SP-HRTM	Bayesian-HOSVD-HRTM
0	55.47 (12.90)	**99.67** (0.11)	35.49 (8.25)	**1.34** (2.03)
1	60.15 (2.22)	**79.59** (22.35)	151.51 (13.14)	**65.61** (61.25)
2	45.84 (4.43)	**64.71** (27.17)	180.88 (5.40)	**93.69** (51.19)
3	17.04 (2.05)	**29.68** (38.22)	184.08 (4.93)	**97.71** (44.38)
All	53.08 (4.83)	**80.68** (15.77)	140.38 (10.39)	**51.42** (34.91)

TABLE 6.5: Average label accuracy over 5 runs (std in brackets) on the synthetic dataset.



(A) Labels generated by the SP-HRTM.

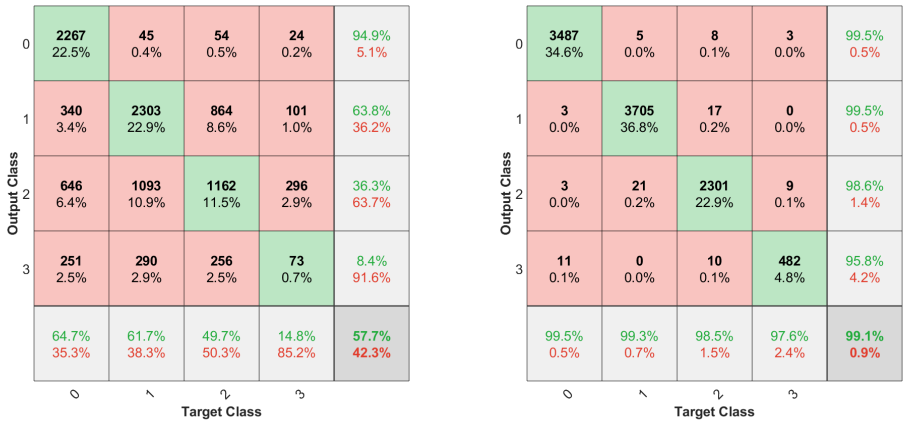
(B) Labels generated by the Bayesian-HOSVD-HRTM.

FIGURE 6.10: An example of tree label generation on the synthetic dataset by the SP-HRTM and the best Bayesian-HOSVD-HRTM execution. Red nodes have wrong label.

The overall accuracy reached by SP-HRTM and Bayesian-HOSVD-HRTM is 53.08% and 80.68% respectively, showing the effectiveness of the approximation introduced. Moreover, the results show that Bayesian-HOSVD-HRTM is able to learn a suitable value of the rank along each dimension to solve the task.

If we observe the accuracy results obtained on each label, the greatest improvement is obtained on label 0. Even if the prediction of the label 0 should be the easiest one (it appears only on leaf nodes), the accuracy obtained by SP-HRTM is only around 55% while the accuracy of Bayesian-HOSVD-HRTM is around 99%. The high entropy value suggests that SP-HRTM attaches the 0 label also to internal nodes (see Figure 6.10a).

In Figure 6.11, we report the confusion matrix obtained by the best SP-HRTM and the best Bayesian-HOSVD-HRTM. The best Bayesian-HOSVD-HRTM reaches an accuracy of 99.1%, while the best SP-HRTM reaches only an accuracy of 57.5%. This huge difference is due to the SP approximation that mixes together the contributions from child nodes. On the contrary, the Bayesian-HOSVD-HRTM is able to distinguish the contribution of each child due to the core tensor $\underline{\boldsymbol{G}}$ which models all the possible



(A) SP-HRTM confusion matrix.

(B) Bayesian-HOSVD-HRTM confusion matrix.

FIGURE 6.11: Confusion matrices obtained by the SP-HRTM and the best Bayesian-HOSVD-HRTM execution.

joint configurations of the rank variables on child nodes. This is in line with the results obtained in Section 4.4.

Nevertheless, there is a consistent difference between the accuracy obtained by the best Bayesian-HOSVD-HRTM model and the results obtained averaging Bayesian-HOSVD-HRTM over 5 runs. This is confirmed by the high standard deviation reported in Table 6.5. We believe that the high variance in the results is due to the strong dependence between the first two steps of the learning algorithm.

6.4 Conclusion

In this chapter, we have introduced two unbounded models for structured data that adapt their complexity directly to data.

The first unbounded model introduced is a Bayesian Non-Parametric (BNP) mixture of SP-HRTMs for structured data clustering. Such a model addresses the problem of setting the number of mixture components a priori by allowing an infinite number of them. Nevertheless, only a finite set of components is actually used during training, while the learning procedure can create (or remove) components on the fly. This worsens the learning procedure which relies on a Gibbs sampling method to approximate the intractable posterior. The experiments have shown the benefit of mixture models in an unsupervised setting. Even on controlled data, a single SP-HRTM have not performed an effective clustering due to its inability to learn root node posteriors which contains global structure information. On the other hand, BNP-SP-HRTM achieves satisfactory results. Moreover, thanks to its ability to learn the number of clusters directly from data, it is computationally more efficient than the finite mixture model.

The second unbounded model introduced is a Bayesian extension of HOSVD-HRTM, which can learn the decomposition ranks directly from data, simplifying the model selection step. The core of the proposed approach is the definition of a prior distribution on the rank values. Then, a stochastic search procedure is implemented to explore rank values during the learning phase. The Bayesian fashion of the model is essential to generate new parameters on the fly when the rank value increases. The experimental analysis conducted shows the ability of Bayesian-HOSVD-HRTM to learn suitable rank values to outperform SP-HRTM.

Chapter 7

Conclusion

In this thesis, we have built a connection between structured data processing and tensors theory. Such a connection has paved the way to the definition of new recursive models by leveraging tensor decompositions. This new class of models have been extensively studied, showing their capacity to limit model complexity as well as the inductive bias they introduce.

The first contribution of this work has been the definition of a framework that relates recursive models for structured data and tensors. This connection arises observing that recursive models with a tensor parametrisation can be used as a mould to define different recursive models by imposing specific constraints on the tensor parameter. Thus, we have argued that full-tensorial models have a low inductive bias since they do not add such constraints. We have instantiated the proposed framework by defining two full-tensorial models: HRTM and RecNTN. The former is a probabilistic model whose state-transition distribution models all possible joint configurations of the input label and the child hidden states. The latter is a neural model whose state-transition function aggregates the input label and the child hidden states of a node through a multi-affine map. Moreover, we have shown how existing approximations in the literature can be framed in the proposed framework, highlighting their major drawbacks.

The connection built between tensors and recursive models has paved the way to applying tensor decompositions to define new state-transition functions. The main advantage of these new tensor-based state-transition functions is that they provided a principled way to limit their complexity. By varying the decomposition rank, we can control the trade-off between model complexity and model expressiveness. Thus, we have introduced nine different recursive models applying three tensor decompositions (i.e. CP, HOSVD, TT) on three model classes (i.e. probabilistic, neural and LSTM). A key point of our work has been the study of the inductive bias introduced by each tensor decomposition. To experimentally assess the advantages of tensor-based models, we have introduced an ad-hoc task on boolean expressions. Despite the simplicity of the task, the results demonstrate that the sum-based and the full-tensorial models achieve performances comparable with a dummy model which simply outputs the most probable answer. In the sum-based models, we have concluded that these performances are due to the independence assumption imposed among the child

hidden states. In the full-tensorial models, we argue that the poor performances are due to their strict relation between model complexity and hidden state size. On the other hand, the models based on the tensor decompositions always achieve 100% accuracy, independently of the model class, using a small number of parameters. A second set of experiments have been run on a benchmark from literature, including both a performance and a computational cost analysis. The results obtained confirm the behaviour highlighted in the previous task. Regarding the computational cost of the proposed models, the models which leverage the tensor decompositions are surprisingly slower than full-tensorial models (for the same number of parameters). We argue that this behaviour is attributable to the optimisations in the computational backend used for the implementation (i.e. PyTorch), which favour the execution of fewer operations on larger operands rather than more operations on small operands.

Despite the effectiveness of the proposed tensor models, they cannot be applied in application domains where the structure out-degree is not known. Thus, we have extended the recursive tensor models by imposing weight sharing constraints on decomposition factors. The weight sharing constraints allow removing the relationship between the number of model parameters and the structure out-degree. Unfortunately, this constraint can be applied only to CP and TT decompositions. The HOSVD cannot be extended in this fashion since its core tensor order depends on the structure out-degree. Also in this case, the use of different tensor decompositions leads to the definition of models with different inductive bias. In particular, models which leverage the CP approximation define permutational invariant state-transition functions (i.e. they ignore the child order). This inductive bias can be deduced by the relation between the symmetric tensors and the CP approximation with shared factor matrices. On the other hand, models which leverage TT approximation define state-transition functions which exploit the child order. We have experimentally assessed the proposed models on different natural language tasks. While probabilistic models do not achieve satisfactory results, neural models based on the infinite tensor decompositions seem appropriate for semantic textual similarity tasks. In this respect, a qualitative analysis has highlighted the importance of tensor-based inductive bias in this task.

All the models which leverage tensor decompositions control the trade-off between model complexity and model expressiveness by regulating a model hyper-parameter (i.e. the decomposition rank). Nevertheless, finding the best hyper-parameters values is a difficult challenge in ML that usually requires a costly model-selection procedure. To this end, we have explored the Bayesian approach to develop unbounded models for structured data, i.e. models which adapt their complexity directly to the input data. In particular, we have introduced two different Bayesian models. The former is a Bayesian Non-Parametric mixture model to tackle unsupervised tasks on structured data. The results obtained on two clustering tasks show the mixture model effectiveness in capturing global structure properties as well as its ability to determine the correct number of mixture components. The latter model is a Bayesian extension of HOSVD-HRTM which learns the decomposition ranks during the learning procedure.

The experimental analysis conducted shows the ability of the Bayesian model to learn suitable rank values outperforming the SP-HRTM.

The connection introduced in this thesis between tensors and ML models for structured data opens the way for future research.

From an applicative perspective, the promising results of this work encourage us to further apply the proposed framework on more complex tasks. A very interesting task could be the source code analysis. We believe that the inductive bias of tensor decompositions are suitable to model the semantics of different commands. For example, the TT decomposition seems to be appropriate to model the execution of a list of commands.

From a more theoretic perspective, it would be interesting to formalise the connection between the approximation of the state-transition function and the model inductive bias. For example, we would like to define a measure to quantify how much the inductive bias introduced by the approximation of the state-transition function limits the model expressiveness. Such a measure should take into account the size of the hidden state. Nevertheless, when tensor decompositions are applied, it should also consider the value of the decomposition rank. In this regard, it would be also interesting to compare how the rank value affects this measure in different tensor decompositions. Similarly, we could also compare the approximations introduced by the tensor decompositions with the sum-based approximation. A good starting point for this study could be a theoretic analysis of the two tasks considered in Chapter 4, showing that all the operators considered can be represented by a tensor. Then, we could compare the minimal rank value required to approximate such tensors by each tensor decompositions. In the same fashion, we could compute the minimal value of the hidden state size required to represent the operators as a summation. On the same line, it would be also interesting to investigate the effect of the training data size in the experiments. In fact, when more data are available, the inductive bias should be less needed.

The framework proposed in this thesis can also be a starting point for developing new models for structured data. In probabilistic models, we believe that the most exciting research direction entails defining new Bayesian models. In this regard, the first step would be applying the approach used in Bayesian-HOSVD-HRTM to the other tensor decompositions. On the same line, it would be interesting to combine the Bayesian approach to determine decomposition ranks automatically with a Bayesian Non-Parametric approach to determine the hidden state size (e.g. [161, 57]). In this way, it would be possible to define recursive models that adapt their hidden state size on the complexity of the input structure and their decomposition rank on the complexity of the interactions among constituents. On sequence domains, an example of such models has been introduced in [145].

Regarding neural models, it would be interesting to apply our tensor framework to other common approaches used in the literature to process structures that have not

been discussed in our thesis, e.g. convolutional models and reservoir computing. In both cases, the contextual information are aggregated through sum-based functions (e.g. in Tree-based Convolutional Neural Network [120] and in Tree Echo State Network [59]). In the case of reservoir computing, further studies should also determine how the tensor that parametrises the encoding function should be initialised to ensure their typical contractive property. Another interesting venue for the application of the proposed framework are the transformers [167] (and more in general the attention mechanism [14]). In this context, tensor decompositions can be used to model more complex interactions among query, key and value matrices while limiting the number of parameters. For example, in [111] the authors use a combination of the CP and HOSVD decomposition to this purpose.

Another exciting research line would be the extension of our tensor framework to deal with cyclic graphs. In this context, most of the existing ML models aggregate contextual information by means of summations [10]. Nevertheless, further investigations are required to incorporate specific properties of aggregation functions for graphs, such as the permutation invariance. A good starting point is given by the infinite canonical approximation introduced in Chapter 5, which naturally implements permutation invariant aggregation functions.

Despite the theoretic appealing of tensors, there are some limitations that should be considered in their practical applications. The first limitation is the numerical instability of tensor operations since they commonly involve the multiplication of L factors, where L is the tensor order. As we have shown in Section 3.2.2, the instability also affects the gradient computation. Note that the instability arises also when tensor decompositions are used since they are also based on multiplications. This aspect is extremely relevant in applications such as learning on social graphs, where the number of neighbours to aggregate can be in the order of thousands. Another limitation that should be considered regards the deep-learning frameworks (e.g. PyTorch) used to implement tensor models. As we have highlighted in Section 4.4.6, they are not optimised to perform tensor operations. Thus, using these frameworks can limit the computational advantages of using tensor decompositions. In this respect, it is worth to be mentioning the recent python library TensorLy [101].

Appendix A

List of Publications

The following list includes all the publications produced during the Ph.D. programme:

1. D. Bacciu and D. Castellana. "Mixture of Hidden Markov Models as tree encoder". In: *Proceedings of the 26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN'18)*. 2018

2. D. Bacciu and D. Castellana. "Bayesian mixtures of Hidden Tree Markov Models for structured data clustering". In: *Neurocomputing* 342 (2019). DOI: 10.1016/j.neucom.2018.11.091

3. D. Bacciu and D. Castellana. "Learning Tree Distributions by Hidden Markov Models". In: *Workshop on Learning and Automata (LearnAut'18)*. 2018

4. D. Castellana and D. Bacciu. "Bayesian Tensor Factorisation for Bottom-up Hidden Tree Markov Models". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. vol. 2019-July. IEEE, July 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851851

5. D. Castellana and D. Bacciu. "Generalising Recursive Neural Models by Tensor Decomposition". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9206597

6. D. Castellana and D. Bacciu. "Tensor Decompositions in Recursive Neural-Networks for Tree-Structured Data". In: *Proceedings of the the 28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN20)*. 2020

7. D. Castellana and D. Bacciu. "Learning from Non-Binary Constituency Trees via Tensor Decomposition". In: *28th International Conference on Computational Linguistic*. 2020

8. D. Castellana and D. Bacciu. "A Tensor Framework for Learning in Structured Domains". In: *Neurocomputing* (2021). Submitted

Appendix B

Contributed Code

The following list includes all the code repositories made available to reproduce all the experiments conducted in this thesis:

- https://github.com/danielecastellana22/tensor-tree-nn provides a framework to build recursive (either probabilistic or neural) models for tree-structured by leveraging tensor decomposition. This repository contains the code for the experiments conducted in Chapter 4 and Chapter 5;
- https://github.com/danielecastellana22/Mixture-HTMM provides the implementation of mixture models for tree data clustering based on SP-HRTM. This repository contains the code for the experiments conducted in Section 6.2;
- https://github.com/danielecastellana22/HOSVD-HTMM provides the implementation of the Bayesian extension of HOSVD-HRTM. This repository contains the code for the experiments conducted in Section 6.3.

Appendix C

Proofs

C.1 Proof of Theorem 1

Theorem (Augmented tensors)**.** *Let* $\psi : \mathbb{R}^{I_1} \times \cdots \times \mathbb{R}^{I_D} \to \mathbb{R}^K$ *be a multi-affine function. There exists an* augmented tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{(I_1+1)\times\cdots\times(I_D+1)\times K}$ *such that, for every* $\boldsymbol{a}_d \in \mathbb{R}^{I_d}$*, it holds:*

$$\psi\left(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_d\right) = \underline{\boldsymbol{T}}\left(\bar{\boldsymbol{a}}_1, \ldots, \bar{\boldsymbol{a}}_d\right),$$

where $\bar{\boldsymbol{a}}_d = [\boldsymbol{a}_d; 1]$ *are the homogeneous coordinate of d-th input vector* $\boldsymbol{a}_d$*.*

Proof. By the Lemma 4.1.3 in [61], we can write:

$$\psi(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_D) = \psi(\boldsymbol{0}, \ldots, \boldsymbol{0}) + \sum_{\substack{S\subseteq\{1,\ldots,d\} \\ S=\{j_1,\ldots,j_k\},\, M\geq 1}} f_S(\boldsymbol{a}_{j_1}, \ldots, \boldsymbol{a}_{j_L}), \tag{C.1}$$

where $\boldsymbol{0} \in \mathbb{R}^{I_1}$ are the vectors with all the entries equal to 0 and each f_S is a multi-linear function. Hence, for each f_S, it exists a tensor $\underline{\boldsymbol{F}}_S$ such that:

$$f_S(\boldsymbol{a}_{j_1}, \ldots, \boldsymbol{a}_{j_k}) = \underline{\boldsymbol{F}}_S(\boldsymbol{a}_{j_1}, \ldots, \boldsymbol{a}_{j_k}) = \underline{\boldsymbol{F}}_S(A_S), \tag{C.2}$$

where $A_S = \{\boldsymbol{a}_j | j \in S\}$ is the subset of input vectors indexed by the elements in S.

First of all, we show how the multi-linear functions $\underline{\boldsymbol{F}}_S$ can be stacked together into the augmented tensor $\underline{\boldsymbol{T}} \in \mathbb{R}^{(I_1+1)\times\cdots\times(I_D+1)\times K}$. To this end, we define $\underline{\boldsymbol{T}}$ as:

$$\underline{\boldsymbol{T}}[i_1, \ldots, i_D, :] = \begin{cases} \psi(\boldsymbol{0}, \ldots, \boldsymbol{0}) \text{ if } \forall d \in [1, D].\, i_d = I_d + 1 \\ \underline{\boldsymbol{F}}_S[i_{j_1}, \ldots, i_{j_L}, :] \text{ if } \forall d \in [1, D].\, i_d = \begin{cases} I_d + 1 \text{ if } d \notin S \\ i_{j_l} \text{ if } d \in S \wedge j_l = d \end{cases} \end{cases}. \tag{C.3}$$

In order to conclude the proof, we must show that the multi-linear function induced by the augmented tensor $\underline{\boldsymbol{T}}$ is exactly $\psi(\cdot)$. Thus, we prove that:

$$\underline{\boldsymbol{T}}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_D) = \sum_{S\subseteq\{1,\ldots,D\}} \underline{\boldsymbol{F}}_S(A_S), \tag{C.4}$$

where the constant vector $\psi(\boldsymbol{0}, \ldots, \boldsymbol{0})$ is obtained when $S = \emptyset$.

We prove it by induction over the number of dimensions D.

Base case $D = 1$. By the definition of multi-linear function induced by a tensor, it holds:

$$\begin{aligned}\underline{\boldsymbol{T}}(\bar{\boldsymbol{a}}_1) &= \sum_{i_1}^{I_1+1} \underline{\boldsymbol{T}}[i_1, :]\bar{\boldsymbol{a}}_1[i_1] = \\ &= \sum_{i_1}^{I_1} \underline{\boldsymbol{T}}[i_1, :]\boldsymbol{a}[i_1] + \underline{\boldsymbol{T}}[I+1, :] = \sum_{i_1}^{I_1} \underline{\boldsymbol{F}}_{\{1\}}[i_1, :]\boldsymbol{a}[i_1] + \psi(\mathbf{0}),\end{aligned} \tag{C.5}$$

where the last equality holds by the definition of $\underline{\boldsymbol{T}}$ in Eq. (C.3). Thus, we can conclude that:

$$\underline{\boldsymbol{T}}(\bar{\boldsymbol{a}}_1) = \sum_{S \subseteq \{1\}} \underline{\boldsymbol{F}}_S(A_S), \tag{C.6}$$

Inductive case. By the definition of multi-linear function induced by a tensor, it holds:

$$\begin{aligned}\underline{\boldsymbol{T}}(\bar{\boldsymbol{a}}_1, \dots, \bar{\boldsymbol{a}}_D) &= \sum_{i_1=1}^{I_1+1} \cdots \sum_{i_D=1}^{I_D+1} \underline{\boldsymbol{T}}[i_1, \dots, i_D, :]\bar{\boldsymbol{a}}_1[i_1] \dots \bar{\boldsymbol{a}}_D[i_D] = \\ &= \sum_{i_1=1}^{I_1+1} \cdots \sum_{i_D=1}^{I_D} \underline{\boldsymbol{T}}[i_1, \dots, i_D, :]\bar{\boldsymbol{a}}_1[i_1] \dots \boldsymbol{a}_D[i_D] + \\ &\quad + \sum_{i_1=1}^{I_1+1} \cdots \sum_{i_D=1}^{I_D} \underline{\boldsymbol{T}}[i_1, \dots, I_D+1, :]\bar{\boldsymbol{a}}_1[i_1] \dots \boldsymbol{a}_{D-1}[i_{D-1}].\end{aligned} \tag{C.7}$$

The sub-tensor $\underline{\boldsymbol{B}} = \underline{\boldsymbol{T}}[:, \dots, I_D + 1, :]$ is a $(D-1)$-way tensor. The function induced by $\underline{\boldsymbol{B}}$ is applied on vectors represented in homogeneous coordinates; thus, it is an $D-1$-way augmented tensor. By applying the inductive hypothesis, we obtain:

$$\underline{\boldsymbol{B}}(\bar{\boldsymbol{a}}_1, \dots, \bar{\boldsymbol{a}}_{D-1}) = \sum_{S \subseteq \{1, \dots, D-1\}} \underline{\boldsymbol{F}}_S(A_S). \tag{C.8}$$

where $\underline{\boldsymbol{F}}_S$ are sub-arrays of $\underline{\boldsymbol{T}}$.

On the other hand, also the sub-tensor $\underline{\boldsymbol{C}} = \sum_{i_D=1}^{I_D} \underline{\boldsymbol{T}}[:, \dots, i_D, :]$ is a $(D-1)$-way augmented tensor. Nevertheless, it is obtained by contracting the tensor $\underline{\boldsymbol{T}}$ with the last input vector $\boldsymbol{a}_D$. By applying the inductive hypothesis on $\underline{\boldsymbol{C}}$, we obtain:

$$\underline{\boldsymbol{C}}(\bar{\boldsymbol{a}}_1, \dots, \bar{\boldsymbol{a}}_{D-1}) = \sum_{S \subseteq \{1, \dots, D-1\}} \underline{\boldsymbol{F}}'_S(A_S), \tag{C.9}$$

where $\underline{\boldsymbol{F}}'_S$ do not correspond to any sub-arrays of $\underline{\boldsymbol{T}}$. However, we can observe that each $\underline{\boldsymbol{F}}'_S$ is equivalent to $\sum_{i_D} \underline{\boldsymbol{F}}_{S \cup \{D\}}[:, \dots, i_D, :]\boldsymbol{a}_D[i_d]$, i.e. they can be obtained by

contracting the sub-arrays $\underline{\boldsymbol{F}}_{S\cup\{D\}}$ of $\underline{\boldsymbol{T}}$ with the input vector $\boldsymbol{a}_D$. Thus, we can write:

$$\underline{\boldsymbol{C}}(\bar{\boldsymbol{a}}_1,\dots,\bar{\boldsymbol{a}}_{D-1}) = \sum_{\substack{S'\subseteq\{1,\dots,D-1\}\\ S=S'\cup\{D\}}} \underline{\boldsymbol{F}}_S(A_S). \tag{C.10}$$

Finally, by plugging Eq. (C.8) and Eq. (C.10) into Eq. (C.7), we obtain:

$$\begin{aligned}\underline{\boldsymbol{T}}(\bar{\boldsymbol{a}}_1,\dots,\bar{\boldsymbol{a}}_D) &= \\ &= \sum_{S\subseteq\{1,\dots,D-1\}} \underline{\boldsymbol{F}}_S(A_S) + \sum_{\substack{S'\subseteq\{1,\dots,D-1\}\\ S=S'\cup\{D\}}} \underline{\boldsymbol{F}}_S(A_S) = \\ &= \sum_{S\subset\{1,\dots,D\}} \underline{\boldsymbol{F}}_S(A_S).\end{aligned} \tag{C.11}$$

Thus, the inductive hypothesis is verified.

We can conclude that for every multi-affine map $\psi : \mathbb{R}^{I_1}\times\cdots\times\mathbb{R}^{I_D}\to\mathbb{R}^K$, there exists a $\underline{\boldsymbol{T}}$ of size $(I_1+1)\times\cdots\times(I_D+1)\times K$ such that:

$$\psi(\boldsymbol{a}_1,\dots,\boldsymbol{a}_D) = \underline{\boldsymbol{T}}(\bar{\boldsymbol{a}}_1,\dots,\bar{\boldsymbol{a}}_D) \tag{C.12}$$

□

Appendix D

EM Procedures

In this appendix, we report the specialisations of the EM procedure that can be used to learn the state-transition distribution of SP-HRTM, CP-HRTM, HOSVD-HRTM and TT-HRTM. Each specialisation is based on a slight modification of the learning procedure introduced for Full-HRTM in Section 3.2.1. We recall that the upward-downward procedure used to compute the posteriors requires two recursive pass of the input structure. An upward pass which goes from the sinks to the super-source node, and a downward pass which goes from the super-source node to the sinks.

For the sake of simplicity, we present the E-step considering only one training example $(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}})$, where $\boldsymbol{\mathcal{X}}$ is the input structure and $\boldsymbol{\mathcal{Y}}$ is the output structure. Moreover, we assume that the input labels are categorical and we define a different state-transition parametrisation for each input label. Thus, all model parameters have a new dimension that can be indexed by the input label.

D.1 SP-HRTM Derivations

The parameters of the SP-HRTM state-transition distribution are the tensors $\underline{\boldsymbol{U}}_1, \ldots, \underline{\boldsymbol{U}}_L$ (which parametrise the mixture component) and the matrix $\boldsymbol{S}$ (which parametrises the mixture distribution). Moreover, we consider the matrix $\boldsymbol{A}$ to parametrise the prior distribution $P(h_v \mid x_v, \boldsymbol{A})$ on sinks.

E-step

The posteriors required are:

$$\epsilon_{s,vl,v} = P(S_v = l, h_{vl}, h_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \quad \forall v \in \text{vert}(\boldsymbol{\mathcal{X}}); \tag{D.1}$$

$$\epsilon_v = P(h_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \quad \forall v \in \text{vert}(\boldsymbol{\mathcal{X}}). \tag{D.2}$$

Upward pass. This procedure computes the following values:

$$\beta_v = P(h_v \mid \boldsymbol{\mathcal{X}}_v, \boldsymbol{\mathcal{Y}}_v). \tag{D.3}$$

Let v be a sink node, the quantity β_v is computed as:

$$\beta_v[h_v] = \frac{P(y_v \mid h_v)\boldsymbol{A}[x_v, h_v]}{Z}, \tag{D.4}$$

where $Z = P(y_v \mid x_v)$ is a normalisation constant.

Let v be an internal node, its β value can be computed as:

$$\beta_v[h_v] \approx \frac{P(y_v \mid h_v) \sum_{l=1}^{L} \sum_{h_{vl}} \boldsymbol{S}[x_v, l] \underline{\boldsymbol{U}}_l[x_v, h_{vl}, h_v] \beta_{vl}[h_{vl}]}{Z}, \tag{D.5}$$

where Z is a normalisation constant. Moreover, we use the approximation symbol to emphasise that this equality does not hold, as we stated in Eq. (3.26).

Downward pass. In this procedure, we first compute the posterior $\epsilon_{s,vl,v}$ given the parent posterior ϵ_v:

$$\epsilon_{s,vl,v}[l, h_{vl}, h_v] = \frac{\epsilon_v[h_v] \beta_{vl}[h_{vl}] \boldsymbol{S}[x_v, l] \underline{\boldsymbol{U}}_l[x_v, h_{vl}, h_v]}{\sum_{l'=1}^{L} \sum_{h_{vl'}} \epsilon_v[h_v] \beta_{vl'}[h_{vl'}] \boldsymbol{S}[x_v, l'] \underline{\boldsymbol{U}}_{l'}[x_v, h_{vl'}, h_v]}. \tag{D.6}$$

Then, we can obtain the posterior of each hidden child variable as:

$$\epsilon_{vl}[h_{vl}] = \sum_{h_v} \epsilon_{s,vl,v}[l, h_{vl}, h_v] \tag{D.7}$$

M-step

The parameter updates are:

$$\boldsymbol{A}[i, j] \propto \sum_{n=1}^{N} \sum_{\substack{v \in \mathrm{vert}(\boldsymbol{\mathcal{X}}^n) \\ \mathrm{ch}(v) = \emptyset}} \epsilon_v[h_v] \times \mathbb{I}[x_v = i], \tag{D.8a}$$

$$\boldsymbol{S}[i, l] \propto \sum_{n=1}^{N} \sum_{v \in \mathrm{vert}(\boldsymbol{\mathcal{X}}^n)} \sum_{h_{vl}} \sum_{h_v} \epsilon_{s,vl,v}[l, hvl, h_v] \times \mathbb{I}[x_v = i], \tag{D.8b}$$

$$\underline{\boldsymbol{U}}_l[i, j, k] \propto \sum_{n=1}^{N} \sum_{v \in \mathrm{vert}(\boldsymbol{\mathcal{X}}^n)} \epsilon_{s,vl,v}[l, j, k] \times \mathbb{I}[x_v = i], \tag{D.8c}$$

D.2 CP-HRTM Derivations

The parameters of the CP-HRTM state-transition distribution are the factor matrices $\underline{\boldsymbol{U}}_1, \ldots, \underline{\boldsymbol{U}}_L, \underline{\boldsymbol{Q}}$. Moreover, we consider the matrix $\boldsymbol{A}$ to parametrise the prior distribution $P(h_v \mid x_v, \boldsymbol{A})$ on sinks.

E-step

The posteriors required by the EM algorithm are:

$$\rho_{vl,v} = P(h_{vl}, r_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \tag{D.9}$$

$$\rho_v = P(r_v, h_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \tag{D.10}$$

$$\epsilon_v = P(h_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}). \tag{D.11}$$

Upward pass. The aim is the computation of the values:

$$\beta_v = P(h_v \mid \boldsymbol{\mathcal{X}}_v, \boldsymbol{\mathcal{Y}}_v), \tag{D.12}$$

$$\gamma_v = P(r_v \mid \boldsymbol{\mathcal{X}}_v, \boldsymbol{\mathcal{Y}}_v). \tag{D.13}$$

Let v be a sink node, the quantity β_v is computed as:

$$\beta_v[h_v] = \frac{P(y_v \mid h_v)\boldsymbol{A}[x_v, h_v]}{Z}, \tag{D.14}$$

where $Z = P(y_v \mid x_v)$ is a normalising constant.

If v is an internal node, we first compute γ_v as:

$$\gamma_v[r_v] = \prod_{l=1}^{L} \sum_{h_{vl}} \underline{\boldsymbol{U}}_l[x_v, h_{vl}, r_v]\beta_{vl}[h_{vl}], \tag{D.15}$$

then we compute β_v as:

$$\beta_v[h_v] = \frac{P(y_v \mid h_v) \sum_{r_v} \underline{\boldsymbol{Q}}[x_v, r_v, h_v]\gamma_v[r_v]}{Z}, \tag{D.16}$$

where $Z = \frac{P(\boldsymbol{\mathcal{Y}}_v|\boldsymbol{\mathcal{X}}_v)}{\prod_{l=1}^{L} P(\boldsymbol{\mathcal{Y}}_{vl}|\boldsymbol{\mathcal{X}}_{vl})}$ is a normalising constant.

Downward pass. The aim is the computations of the posterior ρ_v given ϵ_v as:

$$\rho_v[r_v, h_v] = \frac{\underline{\boldsymbol{Q}}[x_v, r_v, h_v]\gamma_v[r_v]\epsilon_v[h_v]}{\sum_{r_v} \underline{\boldsymbol{Q}}[x_v, r_v, h_v]\gamma_v[r_v]}. \tag{D.17}$$

The posterior $\rho_{vl,v}$ is computed as:

$$\rho_{vl,v}[h_{vl}, r_v] = \frac{\tilde{\rho}_v[r_v]\underline{\boldsymbol{U}}_l[x_v, h_{vl}, r_v]\beta_{vl}[h_{vl}]}{\sum_{h_{vl}} \underline{\boldsymbol{U}}_l[x_v, h_{vl}, r_v]\beta_{vl}[h_{vl}]}, \tag{D.18}$$

where $\tilde{\rho}_v[r_v] = \sum_{h_v} \rho_v[r_v, h_v]$.

Finally, the posterior ϵ_{vl} is computed by marginalisation:

$$\epsilon_{vl}[h_{vl}] = \sum_{r_v} \rho_{vl,v}[h_{vl}, r_v]. \tag{D.19}$$

M-step

The parameter updates are:

$$\underline{\boldsymbol{U}}_l[i, j_l, k] \propto \sum_{n=1}^{N} \sum_{v \in \text{vert}(\boldsymbol{\mathcal{X}}^n)} \rho_{vl,v}[j_l, \ldots, k] \times \mathbb{I}[x_v = i], \tag{D.20a}$$

$$\underline{\boldsymbol{Q}}[i, j, k] \propto \sum_{n=1}^{N} \sum_{v \in \text{vert}(\boldsymbol{\mathcal{X}}^n)} \rho_v[j, \ldots, k] \times \mathbb{I}[x_v = i], \tag{D.20b}$$

$$\boldsymbol{S}_l[i, k] \propto \sum_{n=1}^{N} \sum_{\substack{v \in \text{vert}(\boldsymbol{\mathcal{X}}^n) \\ \text{ch}(v)=\emptyset}} \epsilon_v[k] \times \mathbb{I}[x_v = i]. \tag{D.20c}$$

D.3 HOSVD-HRTM Derivations

The parameters of the HOSVD-HRTM state-transition distribution are the factor matrices $\underline{\boldsymbol{U}}_1, \ldots, \underline{\boldsymbol{U}}_L, \underline{\boldsymbol{Q}}$ and the core tensor $\underline{\boldsymbol{G}}$. Moreover, we consider the matrix $\boldsymbol{A}$ to parametrise the prior distribution $P(h_v \mid x_v, \boldsymbol{A})$ on sinks.

E-step

The posteriors required are:

$$\rho_{vl} = P(h_{vl}, r_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \tag{D.21}$$

$$\rho_{v1,\ldots,vL,v} = P(r_{v1}, \ldots, r_{vL}, r_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \tag{D.22}$$

$$\rho_v = P(r_v, h_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \tag{D.23}$$

$$\epsilon_v = P(h_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}). \tag{D.24}$$

Upward pass. This procedure aims to compute the values:

$$\beta_v = P(h_v \mid \boldsymbol{\mathcal{X}}_v, \boldsymbol{\mathcal{Y}}_v), \tag{D.25}$$

$$\gamma_{vl} = P(r_{vl} \mid \boldsymbol{\mathcal{X}}_v, \boldsymbol{\mathcal{Y}}_v), \tag{D.26}$$

$$\gamma_v = P(r_v \mid \boldsymbol{\mathcal{X}}_v, \boldsymbol{\mathcal{Y}}_v). \tag{D.27}$$

Let v be a sink node, the quantity β_v is computed as:

$$\beta_v[h_v] = \frac{P(y_v \mid h_v)\boldsymbol{A}[x_v, h_v]}{Z}, \tag{D.28}$$

where $Z = P(y_v \mid x_v)$ is a normalising constant.

Let v be an internal node, we first compute γ_{vl} of its child nodes as:

$$\gamma_{vl}[r_{vl}] = \sum_{h_{vl}} \underline{\boldsymbol{U}}_l[x_v, h_{vl}, r_{vl}]\beta_{vl}[h_{vl}]. \tag{D.29}$$

Then, we compute γ_v as:

$$\gamma_v[r_v] = \sum_{r_{v1}} \cdots \sum_{r_{vL}} \underline{\boldsymbol{G}}[x_v, h_{v1}, \ldots, h_{vL}] \prod_{l=1}^{L} \gamma_{vl}[r_{vl}]. \tag{D.30}$$

Finally, we can compute β_v as:

$$\beta_v[h_v] = \frac{P(y_v \mid h_v) \sum_{r_v} \underline{\boldsymbol{Q}}[x_v, r_v, h_v] \gamma_v[r_v]}{Z}, \tag{D.31}$$

where $Z = \frac{P(\boldsymbol{\mathcal{Y}}_v|\boldsymbol{\mathcal{X}}_v)}{\prod_{l=1}^{L} P(\boldsymbol{\mathcal{Y}}_{vl}|\boldsymbol{\mathcal{X}}_{vl})}$ is a normalising constant.

Downward pass. The aim is the computation of the posterior ρ_v given ϵ_v as:

$$\rho_v[r_v, h_v] = \frac{\underline{\boldsymbol{Q}}[x_v, r_v, h_v] \gamma_v[r_v] \epsilon_v[h_v]}{\sum_{r_v} \underline{\boldsymbol{Q}}[x_v, r_v, h_v] \gamma_v[r_v]}. \tag{D.32}$$

The posterior $\rho_{v1,\ldots,vL,v}$ is computed as:

$$\rho_{v1,\ldots,vL,v}[r_{v1}, \ldots, r_{vL}, r_v] = \frac{\tilde{\rho}_v[r_v] \underline{\boldsymbol{G}}[x_v, r_{v1}, \ldots, r_{vL}, r_v] \prod_{l=1}^{L} \gamma_{vl}[r_{vl}]}{\sum_{r_{v1}} \cdots \sum_{r_{vL}} \underline{\boldsymbol{G}}[x_v, r_{v1}, \ldots, r_{vL}, r_v] \prod_{l=1}^{L} \gamma_{vl}[r_{vl}]}, \tag{D.33}$$

where $\tilde{\rho}_v[r_v] = \sum_{h_v} \rho_v[r_v, h_v]$.

Then, we compute the posterior ρ_{vl} as:

$$\rho_{vl}[h_{vl}, r_{vl}] = \frac{\tilde{\rho}_{vl}[r_{vl}] \underline{\boldsymbol{U}}_l[x_v, h_{vl}, r_{vl}] \beta_{vl}}{\sum_{r_{vl}} \underline{\boldsymbol{U}}_l[x_v, h_{vl}, r_{vl}]} \tag{D.34}$$

where $\tilde{\rho}_{vl}[r_{vl}] = \sum_{r_v} \sum_{r_{vl'} \neq r_{vl}} \rho_{v1,\ldots,vL,v}[r_{v1}, \ldots, r_{vL}, r_v]$.

Finally, the posterior ϵ_{vl} is computed by marginalisation:

$$\epsilon_{vl}[h_{vl}] = \sum_{r_{vl}} \rho_{vl}[h_{vl}, r_{vl}]. \tag{D.35}$$

M-step

The parameter updates are:

$$\underline{\boldsymbol{U}}_l[i, j_l, k] \propto \sum_{n=1}^{N} \sum_{v \in \mathrm{vert}(\boldsymbol{\mathcal{X}}^n)} \rho_{vl}[j_l, k] \times \mathbb{I}[x_v = i], \tag{D.36a}$$

$$\underline{\boldsymbol{G}}[i, j_1, \ldots, j_L, k] \propto \sum_{n=1}^{N} \sum_{v \in \mathrm{vert}(\boldsymbol{\mathcal{X}}^n)} \rho_{v1,\ldots,vL,v}[j_1, \ldots, j_L, k] \times \mathbb{I}[x_v = i], \tag{D.36b}$$

$$\underline{\boldsymbol{Q}}[i, j, k] \propto \sum_{n=1}^{N} \sum_{v \in \mathrm{vert}(\boldsymbol{\mathcal{X}}^n)} \rho_v[j, k] \times \mathbb{I}[x_v = i], \tag{D.36c}$$

$$\boldsymbol{S}_l[i, k] \propto \sum_{n=1}^{N} \sum_{\substack{v \in \mathrm{vert}(\boldsymbol{\mathcal{X}}^n) \\ \mathrm{ch}(v) = \emptyset}} \epsilon_v[k] \times \mathbb{I}[x_v = i]. \tag{D.36d}$$

D.4 TT-HRTM Derivations

The parameters of the TT-HRTM state-transition distribution are the core tensors $\underline{\boldsymbol{G}}_1, \ldots, \underline{\boldsymbol{G}}_L, \underline{\boldsymbol{Q}}$. Moreover, we consider the matrix $\boldsymbol{A}$ to parametrise the prior distribution $P(h_v \mid x_v, \boldsymbol{A})$ on sinks.

E-step

The posteriors required are:

$$\rho_{vl} = P(r_{vl-1}, h_{vl}, r_{vL} \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}), \tag{D.37}$$

$$\rho_{vL,v} = P(r_{vL}, h_v \mid \boldsymbol{\mathcal{X}}, \tag{D.38}$$

$$\epsilon_v = P(h_v \mid \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}). \tag{D.39}$$

Upward pass. This procedure computes the values:

$$\beta_v = P(h_v \mid \boldsymbol{\mathcal{X}}_v, \boldsymbol{\mathcal{Y}}_v), \tag{D.40}$$

$$\gamma_{vl} = P(r_{vl} \mid \boldsymbol{\mathcal{X}}_{v1}, \ldots, \boldsymbol{\mathcal{X}}_{vl-1}, \boldsymbol{\mathcal{Y}}_v). \tag{D.41}$$

Let v be a sink node, the quantity β_v is computed as:

$$\beta_v[h_v] = \frac{P(y_v \mid h_v)\boldsymbol{A}[x_v, h_v]}{Z}, \tag{D.42}$$

where $Z = P(y_v \mid x_v)$ is a normalising constant.

Let v be an internal node, we first compute γ_{v1} associated to its first child node as:

$$\gamma_{v1}[r_{v1}] = \sum_{h_{v1}} \underline{\boldsymbol{G}}_1[x_v, h_{v1}, r_{v1}]\beta_{v1}[h_{v1}]. \tag{D.43}$$

Then, we recursively computes γ_{vl} for all $l \in [1, L]$ as:

$$\gamma_{vl}[r_{vl}] = \sum_{h_{vl}} \sum_{r_{vl-1}} \underline{\boldsymbol{G}}_l[x_v, r_{vl-1}h_{vl}, r_{vl}]\beta_{vl}[h_{vl}]\gamma_{vl-1}[r_{vl-1}]. \tag{D.44}$$

Finally, we can compute β_v as:

$$\beta_v[h_v] = \frac{P(y_v \mid h_v)\sum_{r_{vL}} \underline{\boldsymbol{Q}}[x_v, r_{vL}, h_v]\gamma_{vL}[r_vL]}{Z}, \tag{D.45}$$

where $Z = \frac{P(\boldsymbol{\mathcal{Y}}_v|\boldsymbol{\mathcal{X}}_v)}{\prod_{l=1}^{L} P(\boldsymbol{\mathcal{Y}}_{vl}|\boldsymbol{\mathcal{X}}_{vl})}$ is a normalising constant.

Downward pass. This procedure first computes the posterior $\rho_{vL,v}$ given ϵ_v as:

$$\rho_{vL,v}[r_{vL}, h_v] = \frac{\epsilon_v[h_v]\underline{\boldsymbol{Q}}[x_v, r_{vL}, h_v]\gamma_{vL}[r_{vL}]}{\sum_{r_{vL}} \underline{\boldsymbol{Q}}[x_v, r_{vL}, h_v]\gamma_{vL}[r_{vL}]}. \tag{D.46}$$

Then, the posterior ρ_{vl} is computed as:

$$\rho_{vl}[r_{vl-1}, h_{vl}, r_{vl}] = \frac{\tilde{\rho}_{vl}[r_{vl}]\underline{\boldsymbol{G}}_l[x_v, r_{vl-1}, h_{vl}, r_{vl}]\gamma_{vl}[r_{vl}]\beta_{vl}[h_{vl}]}{\sum_{r_{vl-1}}\sum_{h_{vl}}\underline{\boldsymbol{G}}_l[x_v, r_{vl-1}, h_{vl}, r_{vl}]\gamma_{vl}[r_{vl}]\beta_{vl}[h_{vl}]}, \tag{D.47}$$

where $\tilde{\rho}_{vl}[r_{vl}] = \sum_{r_{vl+1}}\sum_{h_{vl+1}}\rho_{vl+1}[r_{vl}, h_{vl+1}, r_{vl+1}]$.

Finally, the posterior ϵ_{vl} is computed by marginalisation:

$$\epsilon_{vl}[h_{vl}] = \sum_{r_{vl-1}}\sum_{r_{vl}}\rho_{vl}[r_{vl-1}, h_{vl}, r_{vl}]. \tag{D.48}$$

M-step

The parameter updates are:

$$\underline{\boldsymbol{G}}_l[i, j_{l-1}, j_l, k] \propto \sum_{n=1}^{N}\sum_{v\in\mathrm{vert}(\boldsymbol{\mathcal{X}}^n)}\rho_{vl}[j_{l-1}, j_l, k] \times \mathbb{I}[x_v = i], \tag{D.49a}$$

$$\underline{\boldsymbol{Q}}[i, j, k] \propto \sum_{n=1}^{N}\sum_{v\in\mathrm{vert}(\boldsymbol{\mathcal{X}}^n)}\rho_{vL,v}[j, k] \times \mathbb{I}[x_v = i], \tag{D.49b}$$

$$\boldsymbol{S}_l[i, k] \propto \sum_{n=1}^{N}\sum_{\substack{v\in\mathrm{vert}(\boldsymbol{\mathcal{X}}^n)\\ \mathrm{ch}(v)=\emptyset}}\epsilon_v[k] \times \mathbb{I}[x_v = i]. \tag{D.49c}$$

Bibliography

[1] E. Acar and B. Yener. "Unsupervised multiway data analysis: A literature survey". In: *IEEE Transactions on Knowledge and Data Engineering* 21.1 (Jan. 2009), pp. 6–20. DOI: 10.1109/TKDE.2008.112.

[2] S. Adhikary, S. Srinivasan, J. Miller, G. Rabusseau, B. Boots, S. Adhikary, J. Miller, G. Rabusseau, and B. Boots. "Quantum Tensor Networks, Stochastic Processes, and Weighted Automata". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 2021, pp. 2080–2088.

[3] A. Anandkumar, R. Ge, S. M. Kakade, D. Hsu, and M. Telgarsky. "Tensor decompositions for learning latent variable models". In: *Journal of Machine Learning Research* 15 (2014), pp. 2773–2832.

[4] S. Arora, N. Cohen, W. Hu, and Y. Luo. "Implicit Regularization in Deep Matrix Factorization". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, d\textquotesingle Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 7413–7424.

[5] S. Arora, Y. Liang, and T. Ma. "A simple but though Baseline for Sentence Embeddings". In: *International Conference on Learning Representations (ICLR)* 15 (2017), pp. 416–424.

[6] P. Austrin, P. Kaski, and K. Kubjas. "Tensor Network Complexity of Multilinear Maps". In: *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Ed. by A. Blum. Vol. 124. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 7:1–7:21. DOI: 10.4230/LIPIcs.ITCS.2019.7.

[7] D. Bacciu and D. Castellana. "Bayesian mixtures of Hidden Tree Markov Models for structured data clustering". In: *Neurocomputing* 342 (2019). DOI: 10.1016/j.neucom.2018.11.091.

[8] D. Bacciu and D. Castellana. "Learning Tree Distributions by Hidden Markov Models". In: *Workshop on Learning and Automata (LearnAut'18)*. 2018.

[9] D. Bacciu and D. Castellana. "Mixture of Hidden Markov Models as tree encoder". In: *Proceedings of the 26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN'18)*. 2018.

[10] D. Bacciu, F. Errica, A. Micheli, and M. Podda. *A gentle introduction to deep learning for graphs*. Sept. 2020. DOI: 10.1016/j.neunet.2020.06.006.

[11] D. Bacciu, A. Micheli, and A. Sperduti. "An input-output hidden Markov model for tree transductions". In: *Neurocomputing* 112 (2013), pp. 34–46. DOI: 10.1016/j.neucom.2012.12.044.

[12] D. Bacciu, A. Micheli, and A. Sperduti. "Compositional Generative Mapping for Tree-Structured Data - Part I: Bottom-Up Probabilistic Modeling of Trees". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.12 (2012), pp. 1987–2002. DOI: 10.1109/TNNLS.2012.2222044.

[13] D. Bacciu, A. Micheli, and A. Sperduti. "Compositional Generative Mapping for Tree-Structured Data - Part II: Topographic Projection Model". In: *IEEE Transactions on Neural Networks and Learning Systems* (2013). DOI: 10.1109/TNNLS.2012.2228226.

[14] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *International Conference on Learning Representations (ICLR)* (2015), pp. 1–15. DOI: 10.1146/annurev.neuro.26.041002.131047.

[15] E. Ballico. "An upper bound for the real tensor rank and the real symmetric tensor rank in terms of the complex ranks". In: *Linear and Multilinear Algebra* 62.11 (Nov. 2014), pp. 1546–1552. DOI: 10.1080/03081087.2013.839671.

[16] E. Ballico. "An Upper Bound for the Tensor Rank". In: *ISRN Geometry* 2013 (2013), pp. 1–3. DOI: 10.1155/2013/241835.

[17] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2016, p. 646. DOI: 10.1017/CBO9780511804779.

[18] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains". In: *The Annals of Mathematical Statistics* (1970). DOI: 10.1214/aoms/1177697196.

[19] H. Ben-Younes, R. Cadene, M. Cord, and N. Thome. "MUTAN: Multimodal Tucker Fusion for Visual Question Answering". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017. DOI: 10.1109/ICCV.2017.285.

[20] Y. Bengio and P. Frasconi. "Input-output HMMs for sequence processing". In: *IEEE Transactions on Neural Networks* 7.5 (Sept. 1996), pp. 1231–1249. DOI: 10.1109/72.536317.

[21] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[22] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[23] G. G. Calvi, A. Moniri, M. Mahfouz, Q. Zhao, and D. P. Mandic. “Compression and Interpretability of Deep Neural Networks via Tucker Tensor Layer: From First Principles to Tensor Valued Back-Propagation”. In: *CoRR* abs/1903.0 (Mar. 2019).

[24] J. D. Carroll and J. J. Chang. “Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition”. In: *Psychometrika* 35.3 (Sept. 1970), pp. 283–319. DOI: 10.1007/BF02310791.

[25] D. Castellana and D. Bacciu. “A Tensor Framework for Learning in Structured Domains”. In: *Neurocomputing* (2021). Submitted.

[26] D. Castellana and D. Bacciu. “Bayesian Tensor Factorisation for Bottom-up Hidden Tree Markov Models”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. Vol. 2019-July. IEEE, July 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851851.

[27] D. Castellana and D. Bacciu. “Generalising Recursive Neural Models by Tensor Decomposition”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9206597.

[28] D. Castellana and D. Bacciu. “Learning from Non-Binary Constituency Trees via Tensor Decomposition”. In: *28th International Conference on Computational Linguistic*. 2020.

[29] D. Castellana and D. Bacciu. “Tensor Decompositions in Recursive Neural-Networks for Tree-Structured Data”. In: *Proceedings of the the 28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN20)*. 2020.

[30] R. B. Cattell. “"Parallel proportional profiles" and other principles for determining the choice of factors by rotation”. In: *Psychometrika* 9.4 (Dec. 1944), pp. 267–283. DOI: 10.1007/BF02288739.

[31] R. B. Cattell. “The three basic factor-analytic research designs–their interrelations and derivatives”. In: *Psychological Bulletin* 49.5 (Sept. 1952), pp. 499–520. DOI: 10.1037/h0054245.

[32] X. Chen, C. Liu, and D. Song. “Tree-to-tree Neural Networks for Program Translation”. In: *Workshop track in International Conference on Learning Representations (ICLR)* 2018-Decem (2018), pp. 2547–2557.

[33] Z. Cheng, C. Yuan, J. Li, and H. Yang. “TreeNet: Learning sentence representations with unconstrained tree structure”. In: *IJCAI International Joint Conference on Artificial Intelligence* 2018-July (2018), pp. 4005–4011. DOI: 10.24963/ijcai.2018/557.

[34] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches". In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Stroudsburg, PA, USA: Association for Computational Linguistics, June 2014, pp. 103–111. DOI: 10.3115/v1/W14-4012.

[35] W. Chu and Z. Ghahramani. "Probabilistic models for incomplete multi-dimensional arrays". In: *Journal of Machine Learning Research* 5.2006 (2009), pp. 89–96.

[36] A. Cichocki, N. Lee, I. V. Oseledets, A. H. Phan, Q. Zhao, and D. Mandic. "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions". In: *Foundations and Trends in Machine Learning* 9.4-5 (Dec. 2016), pp. 249–429. DOI: 10.1561/2200000059.

[37] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and A. H. Phan. "Tensor decompositions for signal processing applications: From two-way to multiway component analysis". In: *IEEE Signal Processing Magazine* 32.2 (2015), pp. 145–163. DOI: 10.1109/MSP.2013.2297439.

[38] N. Cohen, O. Sharir, and A. Shashua. "On the expressive power of deep learning: A tensor analysis". In: *Journal of Machine Learning Research* 49.June (2016), pp. 698–728.

[39] N. Cohen and A. Shashua. "SimNets: A Generalization of Convolutional Networks". In: *Advances in Neural Information Processing Systems 28 (NeurIPS), Deep Learning Workshop* (2014).

[40] S. B. Cohen and M. Collins. "Tensor decomposition for fast parsing with latent-variable PCFGs". In: *Advances in Neural Information Processing Systems*. 2012.

[41] P. Comon, G. Golub, L.-H. Lim, and B. Mourrain. "Symmetric Tensors and Symmetric Tensor Rank". In: *SIAM Journal on Matrix Analysis and Applications* 30.3 (2008), pp. 1254–1279. DOI: 10.1137/060661569.

[42] A. Critch and J. Morton. "Algebraic geometry of matrix product states". In: *Symmetry, Integrability and Geometry: Methods and Applications (SIGMA)* 10 (2014). DOI: 10.3842/SIGMA.2014.095.

[43] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/BF02551274.

[44] L. De Lathauwer, B. De Moor, and J. Vandewalle. "A Multilinear Singular Value Decomposition". In: *SIAM Journal on Matrix Analysis and Applications* 21.4 (Jan. 2000), pp. 1253–1278. DOI: 10.1137/S0895479896305696.

[45] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data Via the EM Algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (Sept. 1977), pp. 1–22. DOI: 10.1111/j.2517-6161.1977.tb01600.x.

[46] L. Denoyer and P. Gallinari. "Report on the XML mining track at INEX 2005 and INEX 2006". In: *ACM SIGIR Forum* 41.1 (June 2007), pp. 79–90. DOI: 10.1145/1273221.1273230.

[47] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of deep bidirectional transformers for language understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[48] M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-30958-8.

[49] M. Droste, W. Kuich, and H. Vogler. *Handbook of weighted automata*. 2009.

[50] P. Dupont, F. Denis, and Y. Esposito. "Links between probabilistic automata and hidden Markov models: Probability distributions, learning models and induction algorithms". In: *Pattern Recognition* 38.9 (Sept. 2005), pp. 1349–1371. DOI: 10.1016/j.patcog.2004.03.020.

[51] J. B. Durand, P. Gonçalvès, and Y. Guédon. "Computational methods for hidden Markov tree models-An application to wavelet trees". In: *IEEE Transactions on Signal Processing* 52.9 (Sept. 2004), pp. 2551–2560. DOI: 10.1109/TSP.2004.832006.

[52] C. A. Ellis. "Probabilistic tree automata". In: *Information and Control* 19.5 (Dec. 1971), pp. 401–416. DOI: 10.1016/S0019-9958(71)90673-5.

[53] J. L. Elman. "Finding structure in time". In: *Cognitive Science* 14.2 (1990), pp. 179–211. DOI: 10.1016/0364-0213(90)90002-E.

[54] A. Eriguchi, K. Hashimoto, and Y. Tsuruoka. "Tree-to-Sequence Attentional Neural Machine Translation". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* 1 (2016), pp. 823–833. DOI: 10.18653/v1/P16-1078.

[55] M. Fannes, B. Nachtergaele, and R. F. Werner. "Finitely correlated states on quantum spin chains". In: *Communications in Mathematical Physics* 144.3 (Mar. 1992), pp. 443–490. DOI: 10.1007/BF02099178.

[56] T. S. Ferguson. "A Bayesian Analysis of Some Nonparametric Problems". In: *The Annals of Statistics* 1.2 (1973), pp. 209–230.

[57] J. R. Finkel, T. Grenager, and C. D. Manning. "The Infinite Tree". In: *Annual Meeting-Association for Computational Linguistics*. 2006, pp. 272–279.

[58] P. Frasconi, M. Gori, and A. Sperduti. "A general framework for adaptive processing of data structures". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 768–786. DOI: 10.1109/72.712151.

[59] C. Gallicchio and A. Micheli. "Tree Echo State Networks". In: *Neurocomputing* 101 (2013), pp. 319–337. DOI: 10.1016/j.neucom.2012.08.017.

[60] C. Gallicchio, A. Micheli, and L. Pedrelli. "Deep reservoir computing: A critical experimental analysis". In: *Neurocomputing* 268 (Dec. 2017), pp. 87–99. DOI: 10.1016/j.neucom.2016.12.089.

[61] J. Gallier. *Curves and surfaces in geometric modeling: theory and algorithms*. Morgan Kaufmann Publishers, 2018.

[62] T. Gärtner. "A survey of kernels for structured data". In: *ACM SIGKDD Explorations Newsletter* 5.1 (July 2003), pp. 49–58. DOI: 10.1145/959242.959248.

[63] E. I. George and R. E. McCulloch. "Approaches for Bayesian variable selection." In: *Statistica Sinica* (1997). DOI: 10.1.1.211.4871.

[64] N. Gianniotis and P. Tiňo. "Visualization of tree-structured data through generative topographic mapping". In: *IEEE Transactions on Neural Networks* (2008). DOI: 10.1109/TNN.2008.2001000.

[65] C. L. Giles and T. Maxwell. *Learning, invariance, and generalization in high-order neural networks*. Dec. 1987. DOI: 10.1364/ao.26.004972.

[66] I. Glasser, R. Sweke, N. Pancotti, J. Eisert, J. Ignacio Cirac, and I. Cirac. "Expressive power of tensor-network factorizations for probabilistic modeling". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, d\textquotesingle Alché-Buc, E. Fox, and R. Garnett. Vol. 32. NeurIPS. Curran Associates, Inc., 2019, pp. 1–13.

[67] C. Goller and A. Kuchler. "Learning task-dependent distributed representations by backpropagation through structure". In: *Proceedings of International Conference on Neural Networks (ICNN'96)* 1 (1996), pp. 347–352. DOI: 10.1109/ICNN.1996.548916.

[68] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[69] P. Gopalan, F. J. Ruiz, R. Ranganath, and D. M. Blei. "Bayesian nonparametric poisson factorization for recommendation systems". In: *Journal of Machine Learning Research* 33 (2014), pp. 275–283.

[70] A. L. Gorodentsev. *Algebra II*. Cham: Springer International Publishing, 2017. DOI: 10.1007/978-3-319-50853-5.

[71] L. Grasedyck. "Hierarchical Singular Value Decomposition of Tensors". In: *SIAM Journal on Matrix Analysis and Applications* 31.4 (Jan. 2010), pp. 2029–2054. DOI: 10.1137/090764189.

[72] A. Graves. “Generating Sequences With Recurrent Neural Networks”. In: (2013).

[73] W. Hackbusch and ·. S. Kühn. “A New Scheme for the Tensor Representation”. In: *J Fourier Anal Appl* 15 (2009), pp. 706–722. DOI: 10.1007/s00041-009-9094-9.

[74] W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*. Vol. 42. Springer Series in Computational Mathematics 8. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–5. DOI: 10.1007/978-3-642-28027-6.

[75] M. Hagenbuchner, A. Sperduti, A. C. Tsoi, F. Trentini, F. Scarselli, and M. Gori. “Clustering XML documents using Self-Organizing Maps for structures”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2006. DOI: 10.1007/978-3-540-34963-1_37.

[76] M. Hagenbuchner, A. Sperduti, and A. C. Tsoi. “A self-organizing map for adaptive processing of structured data”. In: *IEEE Transactions on Neural Networks* (2003). DOI: 10.1109/TNN.2003.810735.

[77] B. Hammer. “Neural methods for non-standard data”. In: *Proceedings of ESANN 2004* (2004).

[78] B. Hammer. *Learning with recurrent neural networks*. Vol. 254. Lecture Notes in Control and Information Sciences. London: Springer London, 2000. DOI: 10.1007/BFb0110016.

[79] B. Hammer. *Universal Approximation of Mappings on Structured Objects using the Folding Architecture*. Tech. rep. 1996.

[80] B. Hammer, A. Micheli, A. Sperduti, and M. Strickert. “A general framework for unsupervised processing of structured data”. In: *Neurocomputing* 57.1-4 (Mar. 2004), pp. 3–35. DOI: 10.1016/j.neucom.2004.01.008.

[81] R. Harshman. “Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis”.

[82] S. E. Hihi and Y. Bengio. “Hierarchical Recurrent Neural Networks for Long-Term Dependencies”. In: *Advances in Neural Information Processing Systems* (1995).

[83] F. L. Hitchcock. “Multiple Invariants and Generalized Rank of a P-Way Matrix or Tensor”. In: *Journal of Mathematics and Physics* 7.1-4 (Apr. 1928), pp. 39–79. DOI: 10.1002/sapm19287139.

[84] F. L. Hitchcock. “The Expression of a Tensor or a Polyadic as a Sum of Products”. In: *Journal of Mathematics and Physics* 6.1-4 (Apr. 1927), pp. 164–189. DOI: 10.1002/sapm192761164.

[85] S. Hochreiter and J. Schmidhuber. “Long short term memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[86] P. D. Hoff. "Hierarchical multilinear models for multiway data". In: *Computational Statistics and Data Analysis* 55.1 (2011), pp. 530–543. DOI: 10.1016/j.csda.2010.05.020.

[87] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* (1989). DOI: 10.1016/0893-6080(89)90020-8.

[88] M. Huang, Q. Qian, and X. Zhu. "Encoding syntactic knowledge in neural networks for sentiment classification". In: *ACM Transactions on Information Systems* 35.3 (2017). DOI: 10.1145/3052770.

[89] M. Iyyer, V. Manjunatha, J. Boyd-Graber, and H. Daumé III. "Deep Unordered Composition Rivals Syntactic Methods for Text Classification". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2015, pp. 1681–1691. DOI: 10.3115/v1/P15-1162.

[90] Y. Ji, Q. Wang, X. Li, and J. Liu. "A Survey on Tensor Techniques and Applications in Machine Learning". In: *IEEE Access* 7 (2019), pp. 162950–162990. DOI: 10.1109/ACCESS.2019.2949814.

[91] V. Khrulkov, O. Hrinchuk, and I. V. Oseledets. "Generalized Tensor Models for Recurrent Neural Networks". In: *International Conference on Learning Representations*. 2019.

[92] V. Khrulkov, A. Novikov, and I. V. Oseledets. "Expressive power of recurrent neural networks". In: *International Conference on Learning Representations*. Nov. 2018, pp. 1–12.

[93] H. A. Kiers. "Towards a standardized notation and terminology in multiway analysis". In: *Journal of Chemometrics* 14.3 (May 2000), pp. 105–122. DOI: 10.1002/1099-128X(200005/06)14:3<105::AID-CEM582>3.0.CO;2-I.

[94] T. Kim, J. Choi, D. Edmiston, S. Bae, and S.-g. Lee. "Dynamic Compositionality in Recursive Neural Networks with Structure-Aware Tag Representations". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (2019), pp. 6594–6601. DOI: 10.1609/aaai.v33i01.33016594.

[95] D. P. Kingma and J. L. Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, Dec. 2015.

[96] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by simulated annealing". In: *Science* (1983). DOI: 10.1126/science.220.4598.671.

[97] M. Kliesch, D. Gross, and J. Eisert. "Matrix-product operators and states: NP-hardness and undecidability". In: *Physical Review Letters* 113.16 (2014), pp. 1–7. DOI: 10.1103/PhysRevLett.113.160503.

[98] A. Klümper, A. Schadschneider, and J. Zittartz. "Matrix Product Ground States for One-Dimensional Spin-1 Quantum Antiferromagnets". In: *Europhysics Letters (EPL)* 24.4 (Nov. 1993), pp. 293–297. DOI: 10.1209/0295-5075/24/4/010.

[99] T. G. Kolda and B. W. Bader. "Tensor Decompositions and Applications". In: *SIAM Review* 51.3 (2009), pp. 455–500. DOI: 10.1137/07070111X.

[100] J. Kossaifi, Z. C. Lipton, A. Kolbeinsson, A. Khanna, T. Furlanello, and A. Anandkumar. "Tensor regression networks". In: *Journal of Machine Learning Research* 21.123 (2020), pp. 1–21.

[101] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic. "TensorLy: Tensor learning in python". In: *Journal of Machine Learning Research* (2019).

[102] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. "Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming". In: *Artificial Intelligence in Design '96*. Springer Netherlands, 1996, pp. 151–170. DOI: 10.1007/978-94-009-0279-4_9.

[103] S. C. Kremer. "On the Computational Power of Elman-Style Recurrent Networks". In: *IEEE Transactions on Neural Networks* 6.4 (1995), pp. 1000–1004. DOI: 10.1109/72.392262.

[104] A. Kuchler. *On the Correspondence between Neural Folding Architectures and Tree Automata*. Tech. rep. Ulmer Informatik-Berichte Nr. 98-06, 1998.

[105] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. Lempitsky. "Speeding-up convolutional neural networks using fine-tuned CP-decomposition". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), pp. 1–11.

[106] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* (1989). DOI: 10.1162/neco.1989.1.4.541.

[107] X. Li and D. Roth. "Learning question classifiers". In: *Proceedings of the 19th international conference on Computational linguistics* -. Vol. 1. Morristown, NJ, USA: Association for Computational Linguistics, 2002, pp. 1–7. DOI: 10.3115/1072228.1072378.

[108] P. Liu, X. Qiu, and X. Huang. "Dynamic Compositional Neural Networks over Tree Structure". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. California: International Joint Conferences on Artificial Intelligence Organization, Aug. 2017, pp. 4054–4060. DOI: 10.24963/ijcai.2017/566.

[109] R. Livni, S. Shalev-Shwartz, and O. Shamir. "On the Computational Efficiency of Training Neural Networks". In: *Advances in Neural Information Processing Systems* 1.January (Oct. 2014), pp. 855–863.

[110] M. Lukoševičius and H. Jaeger. "Reservoir computing approaches to recurrent neural network training". In: *Computer Science Review* 3.3 (Aug. 2009), pp. 127–149. DOI: 10.1016/j.cosrev.2009.03.005.

[111] X. Ma, P. Zhang, S. Zhang, N. Duan, Y. Hou, D. Song, and M. Zhou. "A tensorized transformer for language modeling". In: *Advances in Neural Information Processing Systems*. 2019.

[112] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2014, pp. 55–60. DOI: 10.3115/v1/P14-5010.

[113] M. Marelli, L. Bentivogli, M. Baroni, R. Bernardi, S. Menini, and R. Zamparelli. "SemEval-2014 Task 1: Evaluation of Compositional Distributional Semantic Models on Full Sentences through Semantic Relatedness and Textual Entailment". In: *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. 1. Stroudsburg, PA, USA: Association for Computational Linguistics, 2014, pp. 1–8. DOI: 10.3115/v1/S14-2001.

[114] G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley Series in Probability and Statistics. Hoboken, NJ, USA: John Wiley & Sons, Inc., Sept. 2000. DOI: 10.1002/0471721182.

[115] T. Mikolov. "STATISTICAL LANGUAGE MODELS BASED ON NEURAL NETWORKS". Ph.D. thesis. Brno University of Technology, Faculty of Information Technology, 2012.

[116] C. B. Miller and C. L. Giles. "Experimental Comparison of the Effect of Order in Recurrent Neural Networks". In: *International Journal of Pattern Recognition and Artificial Intelligence* 7.4 (1993), p. 849.

[117] T. M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997.

[118] T. M. Mitchell. *The Need for Biases in Learning Generalizations*. Tech. rep.

[119] L. Mou and Z. Jin. *Tree-Based Convolutional Neural Networks*. SpringerBriefs in Computer Science. Singapore: Springer Singapore, 2018. DOI: 10.1007/978-981-13-1870-2.

[120] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin. "Convolutional Neural Networks over Tree Structures for Programming Language Processing". 2014.

[121] M. Mozer, R. Lippmann, J. Moody, and D. Touretsky. "Induction of multiscale temporal structure". In: *Advances in Neural Information Processing Systems 4* (1992).

[122] V. Murg, F. Verstraete, and J. I. Cirac. "Variational study of hard-core bosons in a two-dimensional optical lattice using projected entangled pair states". In: *Physical Review A - Atomic, Molecular, and Optical Physics* 75.3 (Mar. 2007), p. 033605. DOI: 10.1103/PhysRevA.75.033605.

[123] N. Nangia and S. R. Bowman. "ListOps: A Diagnostic Dataset for Latent Tree Learning". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2018, pp. 92–99. DOI: 10.18653/v1/N18-4013.

[124] R. M. Neal. "Markov Chain Sampling Methods for Dirichlet Process Mixture Models". In: *Journal of Computational and Graphical Statistics* (2000). DOI: 10.1080/10618600.2000.10474879.

[125] A. Novikov, D. D. Podoprikhin, A. Osokin, and D. P. Vetrov. "Tensorizing Neural Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., Sept. 2015, pp. 442–450. DOI: 10.1002/eji.200324821.

[126] A. Novikov, A. Rodomanov, A. Osokin, and D. P. Vetrov. "Putting {MRFs} on a Tensor Train". In: *Proceedings of the 31st International Conference on Machine Learning* 32 (2014), pp. 811–819.

[127] A. Novikov, M. Trofimov, and I. V. Oseledets. "Exponential Machines". In: *Bulletin of the Polish Academy of Sciences: Technical Sciences* 66.6 (May 2016), pp. 789–797. DOI: 10.24425/bpas.2018.125926.

[128] P. Orbanz and Y. W. Teh. *Encyclopedia of Machine Learning*. Ed. by C. Sammut and G. I. Webb. 1. Boston, MA: Springer US, 2010, pp. 1–14. DOI: 10.1007/978-0-387-30164-8.

[129] I. V. Oseledets. "Tensor-Train Decomposition". In: *SIAM Journal on Scientific Computing* 33.5 (Jan. 2011), pp. 2295–2317. DOI: 10.1137/090752286.

[130] R. Pascanu, T. Mikolov, and Y. Bengio. "On the difficulty of training recurrent neural networks". In: *30th International Conference on Machine Learning, ICML 2013*. 2013.

[131] A. Paszke and et al. "Automatic Differentiation in PyTorch". In: *NIPS Autodiff Workshop*. 2017.

[132] J. Pennington, R. Socher, and C. D. Manning. "GloVe: Global vectors for word representation". In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*. 2014, pp. 1532–1543. DOI: 10.3115/v1/d14-1162.

[133] D. Perez-Garcia, F. Verstraete, M. Wolf, and J. Cirac. "Matrix product state representations". In: *Quantum Information and Computation* 7.5&6 (July 2007), pp. 401–430. DOI: 10.26421/QIC7.5-6-1.

[134] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202.

[135] I. Porteous, E. Bart, and M. Welling. "Multi-HDP: A non parametric bayesian model for tensor factorization". In: *Proceedings of the National Conference on Artificial Intelligence* 3 (2008), pp. 1487–1490.

[136] G. Rabusseau, B. Balle, and S. B. Cohen. "Low-rank approximation of weighted tree automata". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016*. 2016.

[137] G. Rabusseau and H. Kadri. "Low-rank regression with tensor responses". In: *Advances in Neural Information Processing Systems*. 2016.

[138] G. Rabusseau, T. Li, and D. Precup. "Connecting Weighted Automata and Recurrent Neural Networks through Spectral Learning". In: *Proceedings of Machine Learning Research*. Ed. by K. Chaudhuri and M. Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1630–1639.

[139] N. Razin and N. Cohen. "Implicit Regularization in Deep Learning May Not Be Explainable by Norms". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Curran Associates, Inc., May 2020, pp. 21174–21187.

[140] S. Rendle. "Factorization Machines". In: *2010 IEEE International Conference on Data Mining*. IEEE, Dec. 2010, pp. 995–1000. DOI: 10.1109/ICDM.2010.127.

[141] E. Rendón, I. Abundez, A. Arizmendi, and E. M. Quiroz. "Internal versus External cluster validation indexes". In: *International Journal of Computers and Communications* (2011).

[142] E. Robeva and A. Seigal. "Duality of graphical models and tensor networks". In: *Information and Inference* 8.2 (2019), pp. 273–288. DOI: 10.1093/imaiai/iay009.

[143] P. J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20 (Nov. 1987), pp. 53–65. DOI: 10.1016/0377-0427(87)90125-7.

[144] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* (1986). DOI: 10.1038/323533a0.

[145] A. Sarkar and D. B. Dunson. "Bayesian higher order hidden markov models". In: *CoRR* abs/1805.1 (2018).

[146] A. Sarkar and D. B. Dunson. "Bayesian Nonparametric Modeling of Higher Order Markov Chains". In: *Journal of the American Statistical Association* 111.516 (2016), pp. 1791–1803. DOI: 10.1080/01621459.2015.1115763.

[147] L. K. Saul and M. I. Jordan. "Mixed memory Markov models: Decomposing complex stochastic processes as mixtures of simpler ones". In: *Machine Learning* 37.1 (1999), pp. 75–87. DOI: 10.1023/A:1007649326333.

[148] G. Shen, Z.-H. Deng, T. Huang, and X. Chen. "Learning to compose over tree structures via POS tags for sentence representation". In: *Expert Systems with Applications* 141 (Mar. 2020), p. 112917. DOI: 10.1016/j.eswa.2019.112917.

[149] Y. Shido, Y. Kobayashi, A. Yamamoto, A. Miyamoto, and T. Matsumura. "Automatic Source Code Summarization with Extended Tree-LSTM". In: *Proceedings of the International Joint Conference on Neural Networks* 2019-July (2019), pp. 1–14. DOI: 10.1109/IJCNN.2019.8851751.

[150] Y. Shitov. "A Counterexample to Comon's Conjecture". In: *SIAM Journal on Applied Algebra and Geometry* 2.3 (Sept. 2018), pp. 428–443. DOI: 10.1137/17M1131970.

[151] H. T. Siegelmann and E. D. Sontag. "On the computational power of neural nets". In: *Journal of Computer and System Sciences* 50.1 (Feb. 1995), pp. 132–150. DOI: 10.1006/jcss.1995.1013.

[152] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. "Semantic compositionality through recursive matrix-vector spaces". In: *EMNLP-CoNLL 2012 - 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Proceedings of the Conference*. 2012.

[153] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *EMNLP 2013 - 2013 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* 8.9 (Dec. 2013). Ed. by T. Preis, pp. 1631–1642.

[154] E. D. Sontag. "Neural nets as system models and controllers". In: *Proc. 7th Yale workshop on Adaptive and Learning Systems* (1992).

[155] N. Srivastava, G. E. Hinton, A. Krizhevsky, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

[156] E. M. Stoudenmire and D. J. Schwab. "Supervised learning with tensor networks". In: *Advances in Neural Information Processing Systems* Nips (2016), pp. 4806–4814.

[157] I. Sutskever, R. Salakhutdinov, and J. B. Tenenbaum. "Modelling relational data using Bayesian clustered tensor factorization". In: *Advances in Neural Information Processing Systems 22 - Proceedings of the 2009 Conference* (2009), pp. 1821–1828.

[158] I. Sutskever, O. Vinyals, Q. V. Le, I. Sutskever Google, O. Vinyals Google, and Q. V. Le Google. "Sequence to sequence learning with neural networks". In: *Advances in Neural Information Processing Systems (NIPS)* 27 (2014), pp. 3104–3112. DOI: 10.1007/s10107-014-0839-0.

[159] K. S. Tai, R. Socher, and C. D. Manning. "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (2015), pp. 1556–1566. DOI: 10.3115/v1/P15-1150.

[160] D. Tao, X. Li, X. Wu, W. Hu, and S. J. Maybank. "Supervised tensor learning". In: *Knowledge and Information Systems* 13.1 (Sept. 2007), pp. 1–42. DOI: 10.1007/s10115-006-0050-6.

[161] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. "Sharing clusters among related groups: Hierarchical dirichlet processes". In: *Advances in Neural Information Processing Systems*. 2005.

[162] Z. Teng and Y. Zhang. "Head-Lexicalized Bidirectional Tree LSTMs". In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 163–177. DOI: 10.1162/tacl_a_00053.

[163] A. Tjandra, S. Sakti, and S. Nakamura. "Tensor Decomposition for Compressing Recurrent Neural Network". In: *Proceedings of the International Joint Conference on Neural Networks*. Vol. 2018-July. Institute of Electrical and Electronics Engineers Inc., Oct. 2018. DOI: 10.1109/IJCNN.2018.8489213.

[164] L. R. Tucker. "Some mathematical notes on three-mode factor analysis". In: *Psychometrika* 31.3 (1966), pp. 279–311. DOI: 10.1007/BF02289464.

[165] V. Vapnik. "An overview of statistical learning theory". In: *IEEE Transactions on Neural Networks* 10.5 (1999), pp. 988–999. DOI: 10.1109/72.788640.

[166] V. N. Vapnik. *Statistical Learning Theory*. A Wiley-Interscience publication. Wiley, 1998.

[167] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates, Inc., 2017, pp. 6000–6010.

[168] F. Verstraete and J. I. Cirac. "Renormalization algorithms for Quantum-Many Body Systems in two and higher dimensions". In: *CoRR* cond-mat/0 (July 2004).

[169] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. "An experimental unification of reservoir computing methods". In: *Neural Networks* 20.3 (Apr. 2007), pp. 391–403. DOI: 10.1016/j.neunet.2007.04.003.

[170] M. Wang and et al. "Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[171] N. Weber, N. Balasubramanian, and N. Chambers. "Event representations with tensor-based compositions". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. 2018.

[172] P. J. Werbos. "Generalization of backpropagation with application to a recurrent gas market model". In: *Neural Networks* (1988). DOI: 10.1016/0893-6080(88)90007-X.

[173] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu. "Towards universal paraphrastic sentence embeddings". In: *4th International Conference on Learning Representations, ICLR 2016*. 2016.

[174] H. Wu, C. Ma, N. Cohen, and A. Shashua. "Convolutional Rectifier Networks As Generalized Tensor Decompositions". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. Vol. 48. ICML'16. JMLR.org, 2016, pp. 955–963. DOI: 10.1109/WISM.2010.164.

[175] Z. Xu, F. Yan, and Y. Qi. "Infinite tucker decomposition: nonparametric Bayesian models for multiway data analysis". In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012* 2.2009 (2012), pp. 1023–1030.

[176] Y. Yang and D. B. Dunson. "Bayesian Conditional Tensor Factorizations for High-Dimensional Classification". In: *Journal of the American Statistical Association* 111.514 (2016), pp. 656–669. DOI: 10.1080/01621459.2015.1029129.

[177] Z. Yang, W. Chen, F. Wang, and B. Xu. "Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets". In: *16th Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2018)*. 2018.

[178] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue. "Long-term forecasting using higher-order tensor RNNs". In: *CoRR* abs/1711.0 (2017), pp. 1–24.

[179] M. D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". Dec. 2012.

[180] Q. Zhao, L. Zhang, and A. Cichocki. "Bayesian CP factorization of incomplete tensors with automatic rank determination". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (2015), pp. 1751–1763. DOI: 10.1109/TPAMI.2015.2392756.

[181] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki. "Tensor Ring Decomposition". In: *CoRR* abs/1606.0 (2016), pp. 1–14.