Weierstraß-Institut
für Angewandte Analysis und Stochastik

Leibniz-Institut im Forschungsverbund Berlin e. V.

Preprint ISSN 2198-5855

A tensor network formalism for Neuro-Symbolic AI

Alex Goessmann[1], Janina Schütte[1], Maximilian Fröhlich[1], Martin Eigel[1]

submitted: January 14, 2026

[1] Weierstrass Institute
Anton-Wilhelm-Amo-Straße 39
10117 Berlin
Germany
E-Mail: alex.goessmann@wias-berlin.de
janina.schuette@wias-berlin.de
maximilian.froehlich@wias-berlin.de
martin.eigel@wias-berlin.de

No. ??
Berlin 2025

2020 *Mathematics Subject Classification.* 65C30, 35R60, 60H35, 65N22, 65J10, 62H12, 60H35, 68T07.

Key words and phrases. Neurosymbolic AI, tensor networks.

We acknowledge funding through BMBF (QROM) and MATH+ (Project PaA-7)..

Edited by
Weierstraß-Institut für Angewandte Analysis und Stochastik (WIAS)
Leibniz-Institut im Forschungsverbund Berlin e. V.
Anton-Wilhelm-Amo-Straße 39
10117 Berlin
Germany

Fax: +49 30 20372-303
E-Mail: preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

A tensor network formalism for Neuro-Symbolic AI

Alex Goessmann, Janina Schütte, Maximilian Fröhlich, Martin Eigel

Abstract

The unification of neural and symbolic approaches to artificial intelligence remains a central open challenge. In this work, we introduce a tensor network formalism, which captures sparsity principles originating in the different paradigms in tensor decompositions. In particular, we describe a basis encoding scheme for functions and model neural decompositions by tensor decompositions. Furthermore, the proposed formalism can be applied to represent logical formulas and probability distributions as structured tensor decompositions. This unified treatment identifies tensor network contractions as a fundamental inference class and formulates efficiently scaling reasoning algorithms, originating from probability theory and propositional logic, as contraction message passing schemes. The framework enables the definition and training of hybrid logical and probabilistic models, which we call Hybrid Logic Networks. The theoretical concepts are accompanied by the python library $\mathrm{tnreason}$, which enables the implementation and practical use of the proposed architectures.

1 Introduction

- general remark about captions: proper sentences not a) this. b) that.
- tensor references in introduction: i would like to add that tensors have also been used for statistical inverse problems (ours and Dolgov/Scheichl) and (PDE constrained) control problems (ours and ?). also cite our tensor machine learning work "bridging..."(eigel-schneider-trunscke).
- reformulate sentence in Definition 7: we define the by t computable and by $\mathcal{G}$ activated family of distributions by Done.
- ensure all sections and subsections have at least one sentence intro.
- the use of capital letters for certain terms is inconsistent. Should be fine now, i went through sections, theorems, examples. Are there further inconsistencies?

Modern artificial intelligence is dominated by large-scale neural models that excel at various tasks but mostly remain black-boxes. While these models offer adaptability, the two main concerns when integrating these architectures into safety-critical processes, are reliability and explainability. To match these demands, artificial intelligence has followed symbolic paradigms, including probabilistic and logical approaches. However, these paradigms have been mostly neglected due to the success of black-box neural models. The logical tradition of artificial intelligence, historically motivated by the resemblance of human thought to formal logics [**?**], offers explicit structures and human-readable inference. However, the main problem hindering the success of this classical approach is the inability of classical first-order logic to handle uncertainty or scale to complex real-world data. Probabilistic graphical models [**?**, **?**] provide insights based on encoded variable independences and causality [**?**].

DOI 10.20347/WIAS.PREPRINT.??

While probabilistic models and Statistical Relational AI [**?**, **?**] have improved uncertainty handling, bridging these paradigms remains the central goal of *Neuro-Symbolic AI* [**?**, **?**, **?**]. The field seeks a single, mathematically coherent framework combining structural clarity with neural adaptability. Although progress has been made, for example with Markov Logic Networks [**?**], a fully unified substrate that treats logical and probabilistic inference as instances of the same operation is still missing.

In this work, we propose to fill the gap between the probabilistic, neural, and logical paradigms with *tensor networks* in a framework called $\mathrm{tnreason}$. Tensor spaces capture both the semantics of logical formulas (by boolean tensors) and probability distributions (by normalized non-negative tensors). This abstraction eliminates the traditional divide between symbolic and neural representations: logical inference, probabilistic computations, and neural inference become different instances of the same underlying operation. As naive tensors are prone to the curse of dimensionality, we turn to distributed representation schemes by tensor networks. We show that fundamental sparsity principles of neural and symbolic AI, such as conditional independence, the existence of sufficient statistics, and neural model decomposition, are equivalent to tensor network decompositions. Moreover, we identify tensor network contraction as the fundamental operation underlying inference tasks, such as computing marginal distributions and deciding entailment. While these contractions are in general computationally hard, efficient schemes to perform inference are known as message passing schemes. These algorithms have appeared in different communities under names such as belief propagation [**?**, **?**] and constraint propagation [**?**]. We review these schemes based on our tensor network formalism.

To capture both logical and probabilistic models, and exploit their neural decompositions, we introduce *Computation-Activation Networks (CompActNets)*, an expressive tensor network architecture. The architecture consists of two complementary subarchitectures that serve the purpose of computation and activation, respectively. The *computation network* prepares auxiliary hidden variables with deterministic dependence on the main variables. We interpret the network as a distributed computation scheme of functions describing this dependence, whose decomposition is related to sparsity concepts of the corresponding functions. These auxiliary variables represent in different contexts logical formulas or more generic statistics. The *activation network* then assigns numerical values to the states of those auxiliary variables, and in this way activates the variables to represent factors of a model. Logical models emerge when the activation network is a boolean tensor, probabilistic exponential families when they are elementary positive-valued tensors, and hybrid models in the most general cases.

1.1 Related works

Historically rooted in quantum many-body physics [**?**], tensor networks found its first major success with Matrix Product States (MPS), originally developed to efficiently capture the quantum dynamics and ground states of one-dimensional spin chains [**?**]. This format remains a standard tool in the field, with recent contributions refining it for tasks such as large-scale stochastic simulations and variational circuit operations [**?**, **?**]. To address the topological constraints of MPS, the landscape of architectures was subsequently expanded to include Projected Entangled Pair States (PEPS) for two-dimensional lattices and the Multi-scale Entanglement Renormalization Ansatz (MERA), which utilizes a hierarchical geometry to represent scale-invariant critical systems and has recently been adapted for simulating quantum systems [**?**, **?**]. Beyond the quantum realm, these formats have been successfully adapted to applied mathematics, particularly for solving high-dimensional parametric PDEs, sampling problems, modeling complex continuous fields and learning dynamical laws [**?**, **?**, **?**]. Furthermore, they exhibit properties helpful for handling these high-dimensional spaces, such as restricted isometry properties [**?**]. Recent advancements have demonstrated the efficacy of these methods in capturing

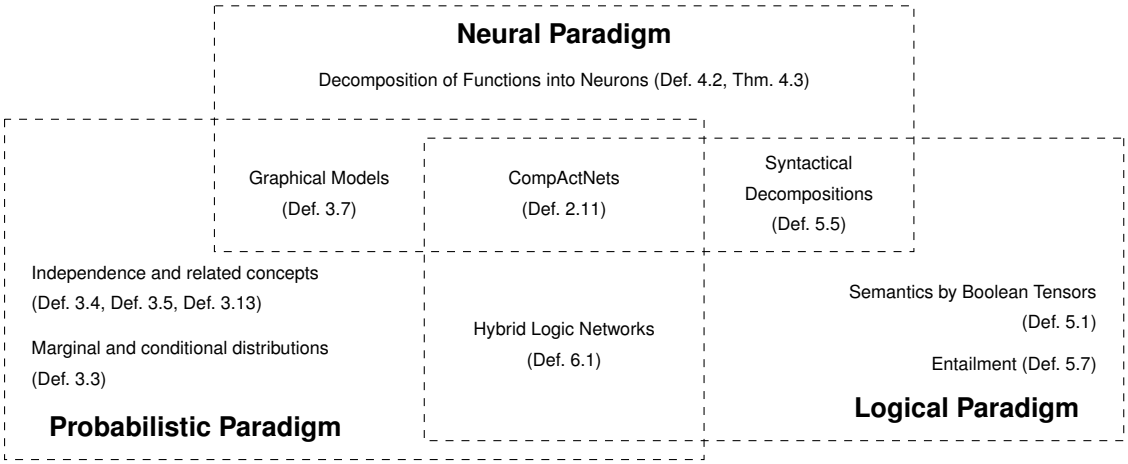


Figure 1: Sketch of the concepts in the neural, probabilistic and logical paradigms, which we define based on tensor network decompositions and contractions.

multiscale phenomena in fluid dynamics and turbulence, proving that the tensor network formalism offers a robust alternative to classical numerical schemes [**?**].

The unification of neural, symbolic, and probabilistic approaches to interpretable model architectures has been a long-standing aim of Neuro-Symbolic AI. A central goal is to achieve *intrinsic explainability*, which, unlike post-hoc interpretations analysing input influence after training [**?**, **?**], aims at explainability of the architecture itself. Early connectionist approaches [**?**, **?**] towards Neuro-Symbolic AI focus on embedding logical rules into neural connectivity. Further, fruitful relations with statistical relational learning have been identified [**?**].

Tensor networks have recently gained interest as a unifying language for AI, framed by Logical Tensor Networks [**?**] and Tensor Logic [**?**]. Furthermore, the MeLoCoToN approach [**?**] applies tensor network architectures similar to CompActNets in combinatorical optimization problems. Specifically, tensor networks have emerged as a highly efficient mathematical framework for handling data in high-dimensional spaces, effectively circumventing the "curse of dimensionality"that typically plagues grid-based methods [**?**]. By decomposing high-order tensors into networks of low-rank components, these structures reduce the storage and computational complexity from exponential to polynomial with respect to the dimension [**?**, **?**, **?**].

1.2 Structure of the paper

The paper is organized as follows. In Section 2 we introduce the basic concepts and notation for categorical variables, tensors, and tensor networks, which are applied in the following sections. Sections 3, 4, and 5 anchor the tensor network formalism in the basic paradigms of artificial intelligence (see Figure 1). The *probabilistic paradigm* is discussed in Section 3, where we in particular show that concepts of independence, graphical models, and sufficient statistics correspond to specific tensor network decompositions. Section 4 is dedicated to the *neural paradigm*, where we show that generic function decompositions have an analogous tensor networks. Section 5 turns to the *logical paradigm*, where we study tensor equivalents of propositional formulas, knowledge bases, and entailment. In Section 6 we present *Hybrid Logic Networks* as an application of the unified tensor network formalism, combining logical and probabilistic models. We briefly discuss the implementation of these concepts in our open source python package $\mathrm{tnreason}$ in Section 7, before concluding the paper in Section 8.

2 Foundations

In this section, we introduce the central hypergraph-based tensor network formalism and define the architecture of CompActNets based on this formalism.

2.1 Tensors

Tensors are multiway arrays and a generalization of vectors and matrices to higher orders. We first provide a formal definition as real maps from index sets enumerating the coordinates of vectors, matrices, and higher-order tensors.

Definition 2.1 (Tensor)**.** *For $k \in [d]$, let $m_k \in \mathbb{N}$ and let X_k be variables taking values in $\{0, \dots, m_k - 1\}$. A tensor $\tau[X_0, \dots, X_{d-1}]$ of order d and with leg dimensions $m_0, \dots, m_{d-1}$ is defined through its coordinates*

$$\tau[X_0 = x_0, \dots, X_{d-1} = x_{d-1}] \in \mathbb{R}$$

for index tuples

$$(x_0, \dots, x_{d-1}) \in \bigtimes_{k \in [d]} [m_k] .$$

Tensors $\tau[X_0, \dots, X_{d-1}]$, also denoted by $\tau\left[X_{[d]}\right]$, are elements of the tensor space

$$\bigotimes_{k \in [d]} \mathbb{R}^{m_k} ,$$

which is a linear space, enriched with the operations of coordinate-wise summation and scalar multiplication. We call a tensor $\tau\left[X_{[d]}\right]$ boolean, when all coordinates are in $\{0, 1\}$, and positive, when all coordinates are greater than 0. To ease the notation, we abbreviate sets as $[d] = \{0, \dots, d-1\}$, tuples of state indices by $x_{[d]} = (x_0, \dots, x_{d-1})$ and tuples of variables by $X_{[d]} = (X_0, \dots, X_{d-1})$.

We introduced tensors here in a non-canonical way based on categorical variables assigned to their axes. While this may look like syntactic sugar at this point, it allows us to define contractions without further specification of axes, based on comparisons of shared variables. We occasionally also allow for variables X taking values in infinite sets such as $\mathbb{R}$, in which case we denote the set of values to a variable by $\mathrm{val}(X)$.

Example 2.2 (Delta tensor)**.** *Given a tuple of variables $X_{[d]} = (X_0, \dots, X_{d-1})$ with identical dimension m, where $d \geq 1$, the delta tensor is the element*

$$\delta^{[d],m}\left[X_{[d]}\right] \in \bigotimes_{k \in [d]} \mathbb{R}^m$$

with coordinates

$$\delta^{[d],m}\left[X_{[d]} = x_{[d]}\right] = \begin{cases} 1 & \text{if} \quad x_0 = \ldots = x_{d-1} \\ 0 & \text{else} \end{cases} . \tag{2.1}$$

We depict this tensor by black dots, which sometimes appears as auxiliary elements in tensor network diagrams (see e.g. Figure 4). For $d = 1$, the delta tensor is the trivial vector, whose coordinates are constantly 1, which we denote by $\mathbb{I}[X]$.

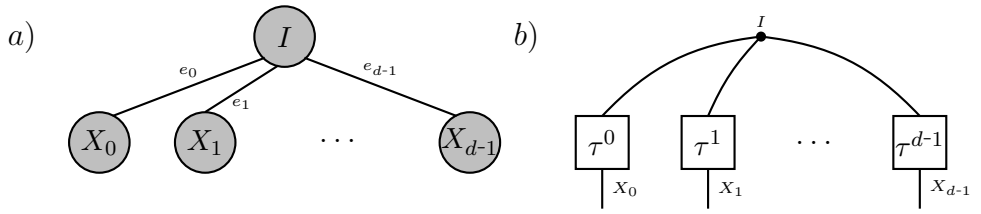
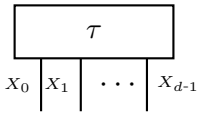


Figure 2: Hypergraph to a CP format. a) Node-centric design. b) Corresponding tensor network on the edges of the hypergraph.

2.2 Tensor networks and contractions

We use a standard visualization of tensors (dating back to [**?**]) by blocks with lines depicting the axes of the tensor. Additionally, we assign to each axis of the tensor the corresponding variable X_k:



Drawing on the association of variables with nodes and of tensors with hyperedges built from the assigned variables, we continue with the definition of tensor networks.

Definition 2.3 (Tensor network)**.** *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a hypergraph, let X_v for $v \in \mathcal{V}$ be categorical variables with dimensions $m_v \in \mathbb{N}$, and let*

$$\tau^e [X_e] \in \bigotimes_{v \in e} \mathbb{R}^{m_v},$$

be core tensors for $e \in \mathcal{E}$, where we denote by X_e the set of categorical variables X_v with $v \in e$. Then, we call the set

$$\tau^{\mathcal{G}} [X_{\mathcal{V}}] = \{\tau^e [X_e] \,:\, e \in \mathcal{E}\}$$

the tensor network of the decorated hypergraph $\mathcal{G}$. The set of tensor networks on $\mathcal{G}$ such that all tensors have non-negative coordinates is denoted by $\mathcal{T}^{\mathcal{G}}$.

As examples we now present the CP and the TT formats in our hypergraph notation.

Example 2.4 (The CP format)**.** *The Candecomp-Parafac (CP ([**?**]) tensor format corresponds in our notation to a hypergraph (see Figure 2) defined by*

- *nodes $X_{[d]}$ and a single hidden variable I, decorated by dimensions $m_{[d]}$ and the CP-rank n, respectively*
- *edges $\{e_k = \{X_k, I\} \,:\, k \in [d]\}$ each decorated by a matrix $\tau^{e_k} [X_k, I] \in \mathbb{R}^{m_k \times n}$.*

Example 2.5 (The TT format)**.** *The Tensor-Train (TT) format (see [**?**]) corresponds in our notation to a hypergraph (see Figure 3) defined by*

- *nodes $X_{[d]}$ and hidden variables $I_{[d-1]}$, each decorated by a dimension $m_{[d]}$ and $n_{[d-1]}$,*
- *edges*

  $$\{e_0 = \{X_0, I_0\}\} \cup \{e_k = \{I_{k-1}, X_k, I_k\} \,:\, k \in \{1, \ldots, d-2\}\} \cup \{e_{d-1} = \{I_{d-2}, X_{d-1}\}\}$$

  each decorated by a tensor of order 3 (respectively 2 for $k \in \{0, p-1\}$).

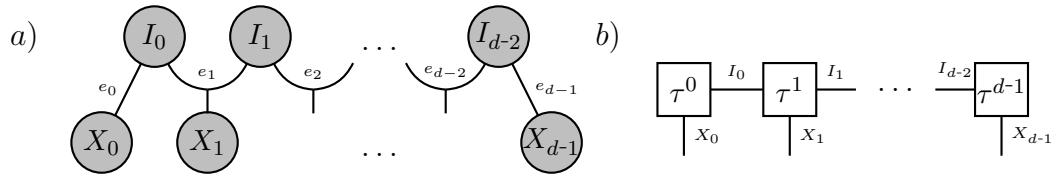


Figure 3: Hypergraph to a TT format. a) Node-centric design. b) Corresponding tensor network on the edges of the hypergraph.

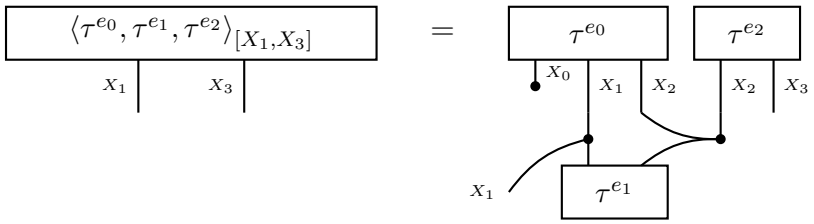


Figure 4: Graphical depiction of a tensor network contraction with the open variables X_1, X_3. Open variables are depicted by those without a dot at the end of the line.

2.3 Generic contractions

Let us now exploit our graphical approach to tensor networks in the definition of contractions.

Definition 2.6. *Let $\tau^{\mathcal{G}}$ be a tensor network on a decorated hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For any subset $\mathcal{U} \subset \mathcal{V}$ we define the contraction to be the tensor (for an example see Figure 4)*

$$\left\langle \tau^{\mathcal{G}} \right\rangle_{[X_{\mathcal{U}}]} \in \bigotimes_{v \in \mathcal{U}} \mathbb{R}^{m_v} \tag{2.2}$$

defined coordinatewise by the sum

$$\left\langle \tau^{\mathcal{G}} \right\rangle_{[X_{\mathcal{U}} = x_{\mathcal{U}}]} = \sum_{x_{\mathcal{V}/\mathcal{U}} \in \bigtimes_{v \in \mathcal{V}/\mathcal{U}} [m_v]} \left(\prod_{e \in \mathcal{E}} \tau^e \left[X_e = x_e \right] \right) . \tag{2.3}$$

We call $X_{\mathcal{U}}$ the open variables of the contraction.

When an open variable X does not appear in any tensor in a contraction, we define the contraction as a tensor product with the trivial tensor $\mathbb{I}[X]$. To ease notation, we often omit the set notation by brackets $\{\cdot\}$.

Example 2.7 (Tensor product)**.** *The simplest contraction is the tensor product, which maps a pair of two tensors with distinct variables onto a third tensor and has an interpretation by coordinate-wise products. Such a contraction corresponds with a tensor network of two tensors with disjoint variables. Let there be two tensors*

$$\tau \left[X_{[d]} \right] \in \bigotimes_{k \in [d]} \mathbb{R}^{m_k} \quad \textit{and} \quad \tilde{\tau} \left[Y_{[p]} \right] \in \bigotimes_{l \in [p]} \mathbb{R}^{m_l}$$

with different categorical variables assigned to their axes. Then their tensor product is the map

$$\left\langle \tau\left[X_{[d]}\right], \tilde{\tau}\left[Y_{[p]}\right]\right\rangle_{\left[X_{[d]}, Y_{[p]}\right]} \in \left(\bigotimes_{k\in[d]} \mathbb{R}^{m_k}\right) \otimes \left(\bigotimes_{l\in[p]} \mathbb{R}^{m_l}\right)$$

defined coordinatewise for tuples of $(x_0, \ldots, x_{d-1}) \in \bigtimes_{k\in[d]}[m_k]$ *and* $(y_0, \ldots, y_{r-1}) \in \bigtimes_{l\in[r]}[m_l]$ *as*

$$\begin{aligned}&\left\langle \tau\left[X_{[d]}\right], \tilde{\tau}\left[Y_{[p]}\right]\right\rangle_{[X_0=x_0,\ldots,X_{d-1}=x_{d-1},Y_0=y_0\ldots,Y_{p-1}=y_{p-1}]}\\ &\quad \coloneqq \tau\left[X_0 = x_0, \ldots, X_{d-1} = x_{d-1}\right] \cdot \tilde{\tau}\left[Y_0 = y_0 \ldots, Y_{p-1} = y_{p-1}\right] .\end{aligned}$$

2.4 Normalizations

Based on generic contractions, we now introduce the normalization of tensors, which is later needed in the probabilistic paradigm, e.g. to represent probability distributions in the tensor network framework.

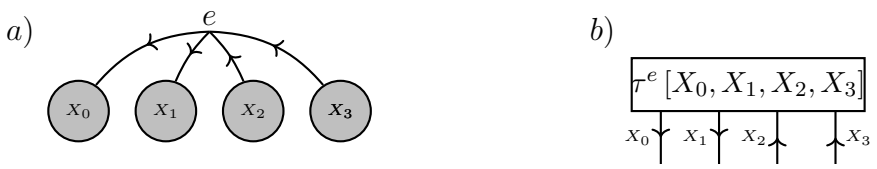
Definition 2.8. *The normalization of a tensor* $\tau\left[X_{\mathcal{V}}\right]$ *on incoming nodes* $\mathcal{V}^{\text{in}} \subset \mathcal{V}$ *and outgoing nodes* $\mathcal{V}^{\text{out}} \subset \mathcal{V}/\mathcal{V}^{\text{in}}$ *is the tensor* $\left\langle \tau\left[X_{\mathcal{V}}\right]\right\rangle_{\left[X_{\mathcal{V}^{\text{out}}}|X_{\mathcal{V}^{\text{in}}}\right]}$ *defined for* $x_{\mathcal{V}^{\text{in}}}$ *as*

$$\left\langle \tau\left[X_{\mathcal{V}}\right]\right\rangle_{\left[X_{\mathcal{V}^{\text{out}}}|X_{\mathcal{V}^{\text{in}}}=x_{\mathcal{V}^{\text{in}}}\right]} = \begin{cases} \frac{\langle\tau\rangle_{\left[X_{\mathcal{V}^{\text{out}}}, X_{\mathcal{V}^{\text{in}}}=x_{\mathcal{V}^{\text{in}}}\right]}}{\langle\tau\rangle_{\left[X_{\mathcal{V}^{\text{in}}}=x_{\mathcal{V}^{\text{in}}}\right]}} & \textit{if} \quad \langle\tau\rangle_{\left[X_{\mathcal{V}^{\text{in}}}=x_{\mathcal{V}^{\text{in}}}\right]} \neq 0 \\ \frac{1}{\prod_{v\in\mathcal{V}^{\text{out}}} m_v}\mathbb{I}\left[X_{\mathcal{V}^{\text{out}}}\right] & \textit{else} \end{cases} .$$

We say that $\tau\left[X_{\mathcal{V}}\right]$ *is normalized with incoming nodes* $\mathcal{V}^{\text{in}} \subset \mathcal{V}$*, if*

$$\tau\left[X_{\mathcal{V}}\right] = \left\langle \tau\left[X_{\mathcal{V}}\right]\right\rangle_{\left[X_{\mathcal{V}/\mathcal{V}^{\text{in}}}|X_{\mathcal{V}^{\text{in}}}\right]} .$$

In our graphical tensor notation, we depict normalized tensors by directed hyperedges (a), which are decorated by directed tensors (b), for example:



2.5 Function encoding and Computation-Activation Networks

Towards presenting the function encoding schemes, we define one-hot encodings mapping the states of variables to basis tensors.

Definition 2.9 (One-hot encoding)**.** *To any variable* X *taking values in* $[m]$*, the one-hot encoding of any state* $x \in [m]$ *is the vector with coordinates*

$$\epsilon_x\left[X = \tilde{x}\right] \coloneqq \begin{cases} 1 & \textit{if} \quad x = \tilde{x} \\ 0 & \textit{else} . \end{cases} \tag{2.4}$$

To any tuple $(X_0, \dots, X_{d-1})$ *of variables taking values in* $\bigtimes_{k\in[d]}[m_k]$*, the one-hot encoding of a state tuple* $x_{[d]} = (x_0, \dots, x_{d-1})$ *is the tensor product*

$$\epsilon_{x_{[d]}}\left[X_{[d]}\right] := \bigotimes_{k\in[d]} \epsilon_{x_k}\left[X_k\right] .$$

We now use one-hot encodings to encode functions between state sets.

Definition 2.10 (Basis encoding of maps between state sets)**.** *Let there be two sets of variables* $X_{[d]}$ *and* $Y_{[p]}$*, and let there be a map*

$$q : \bigtimes_{k\in[d]}[m_k] \to \bigtimes_{l\in[r]}[m_l]$$

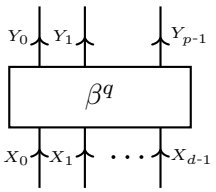
between their state sets. Then, the basis encoding of q *is a tensor*

$$\beta^q\left[Y_{[p]}, X_{[d]}\right] \in \left(\bigotimes_{l\in[p]} \mathbb{R}^{m_l}\right) \otimes \left(\bigotimes_{k\in[d]} \mathbb{R}^{m_k}\right)$$

defined by

$$\beta^q\left[Y_{[p]}, X_{[d]}\right] \quad = \sum_{(x_0,\dots,x_{d-1})\in\bigtimes_{k\in[d]}[m_k]} \epsilon_{q\left(x_{[d]}\right)}\left[Y_{[p]}\right] \otimes \epsilon_{x_{[d]}}\left[X_{[d]}\right] .$$

Basis encodings are normalized tensors and are thus depicted as decorations of directed edges in hypergraphs:



We further generalize basis encodings to arbitrary functions between finite sets by the use of image enumeration maps. Given an arbitrary set $\mathcal{U}$, we say a map

$$I : \bigtimes_{k\in[d]}[m_k] \to \mathcal{U}$$

is an enumeration map of $\mathcal{U}$ by d variables X_k, taking values in m_k. Given a function $q : \mathcal{U}^{\text{in}} \to \mathcal{U}^{\text{out}}$ between arbitrary sets and enumerating maps I_{in} and I_{out} for both sets, we define the basis encoding of q as

$$\beta^q\left[Y_{[p]}, X_{[d]}\right] = \sum_{u\in\mathcal{U}^{\text{in}}} \epsilon_{I_{\text{out}}^{-1}(q(u))}\left[Y_{[p]}\right] \otimes \epsilon_{I_{\text{in}}^{-1}(u)}\left[X_{[d]}\right] ,$$

where X, Y are variables taking values in $\left[\left|\mathcal{U}^{\text{in}}\right|\right]$ and $[|\mathcal{U}^{\text{out}}|]$. In Example 4.4 we present index enumeration maps for summations in m-adic integer representations. Based on these concepts, we define the most general tensor network architecture to be applied in the rest of this work.

Definition 2.11 (Computation-Activation Network (CompActNets)). *Let there be a function* $t : \bigtimes_{k\in[d]}[m_k] \to \mathbb{R}^p$ *and a hypergraph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *with nodes* $[p] \subset \mathcal{V}$ *containing the image coordinates of* t*. We define the by* t *computable and by* $\mathcal{G}$ *activated family of distributions by*

$$\Lambda^{t,\mathcal{G}} = \left\{ \left\langle \beta^t \left[Y_{[p]}, X_{[d]} \right], \langle \xi \rangle_{\left[Y_{[p]}\right]} \right\rangle_{\left[X_{[d]}|\varnothing\right]} \,:\, \xi\left[Y_{\mathcal{V}}\right] \in \mathcal{T}^{\mathcal{G}} \right\} .$$

We refer to any member $\mathbb{P}\left[X_{[d]}\right] \in \Lambda^{t,\mathcal{G}}$ *as a* Computation-Activation Network *(*CompActNet*). We call* $\beta^t \left[Y_{[p]}, X_{[d]}\right]$ *(and any decomposition of it) the* computation network *and* $\xi\left[Y_{\mathcal{V}}\right]$ *the* activation network*.*

The elementary activated networks are representable by an elementary activation tensor with respect to the graph

$$\mathrm{EL} = \big(\mathcal{V}, \{\{v\} \,:\, v \in \mathcal{V}\}\big)$$

and we denote such networks by $\Lambda^{t,\mathrm{EL}}$. Any CompActNet is representable with respect to the maximal hypergraph

$$\mathrm{MAX} = \big(\mathcal{V}, \{\mathcal{V}\}\big) .$$

We therefore have for any graph that $\Lambda^{t,\mathcal{G}} \subset \Lambda^{t,\mathrm{MAX}}$.

3 The probabilistic paradigm

In the following we investigate tensor network decomposition mechanisms of probability distributions. After introducing probability distributions as tensors, we derive tensor network decompositions based on conditional independencies (applying the Hammersley-Clifford theorem [**?**]) to motivate graphical models. Furthermore, we present the Computation mechanism, in which the Fisher-Neyman Factorization Theorem is used to decompose distributions in the presence of sufficient statistics.

3.1 Basic concepts

As defined next, distributions $\mathbb{P}$ over a discrete state space can be represented by tensors, where each entry corresponds to the probability of a corresponding state.

Definition 3.1 (Joint probability distribution). *Let there be for each* $k \in [d]$ *a categorical variable* X_k *taking values in* $[m_k]$*. A joint probability distribution of these categorical variables is a function*

$$\mathbb{P}\left[X_{[d]}\right] \,:\, \bigtimes_{k\in[d]} [m_k] \to \mathbb{R}$$

which is non-negative, that is for any $x_{[d]} \in \bigtimes_{k\in[d]}[m_k]$ *it holds*

$$\mathbb{P}\left[X_{[d]} = x_{[d]}\right] \geq 0 ,$$

and which is normalized, that is

$$\left\langle \mathbb{P}\left[X_{[d]}\right] \right\rangle_{[\varnothing]} = 1 .$$

Let Z *be another variable taking values in a possibly infinite set* $\mathrm{val}(Z)$*. Then, a tensor* $\mathbb{P}\big[X_{[d]}\big|Z\big]$ *is a family of joint probability distributions if, for any* $z \in \mathrm{val}(Z)$*, the slice* $\mathbb{P}\big[X_{[d]}\big|Z = z\big]$ *is a joint probability distribution.*

Example 3.2 (Family of independent coin tosses). *Consider tossing a coin with head probability $z \in [0,1]$ and repeating the experiment independently $d \in \mathbb{N}$ times. We define a variable Z taking values in $\mathrm{val}(Z) = [0,1]$ and denote by $X_{[d]}$ d boolean variables. Then, the family of coin toss distributions is the tensor $\mathbb{P}\left[X_{[d]} \middle| Z\right]$ with coordinates $x_{[d]} \in \bigtimes_{k\in[d]}[2]$ and $z \in [0,1]$ defined by*

$$\mathbb{P}\left[X_{[d]} = x_{[d]} \middle| Z = z\right] = \prod_{k\in[d]} z^{x_k}(1-z)^{1-x_k} = z^{\sum_{k\in[d]} x_k}(1-z)^{d-\sum_{k\in[d]} x_k} .$$

Note that for each slice with respect to $z \in [0,1]$ by the binomial theorem we have $\left\langle \mathbb{P}\left[X_{[d]}, Z = z\right]\right\rangle_{[\varnothing]} = 1$ and thus $\mathbb{P}\left[X_{[d]}, Z\right]$ is indeed a family of probability distributions. For $d = 2$ we have more explicitly for any $z \in [0,1]$ that

$$\mathbb{P}\left[X_{[2]} \middle| Z = z\right] = \begin{bmatrix} (1-z)^2 & z\cdot(1-z) \\ z\cdot(1-z) & z^2 \end{bmatrix} .$$

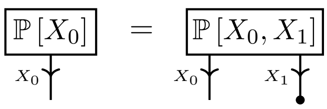
(rows indexed by $x_0 \in \{0,1\}$, columns by $x_1 \in \{0,1\}$)

A basic inference operation on probability distributions is the computation of marginal and conditional distributions.

Definition 3.3. *For any distribution $\mathbb{P}[X_0, X_1]$ the marginal distribution is given by the contraction (see Def. 2.6)*

$$\mathbb{P}[X_0] := \langle \mathbb{P}[X_0, X_1] \rangle_{[X_0]} ,$$

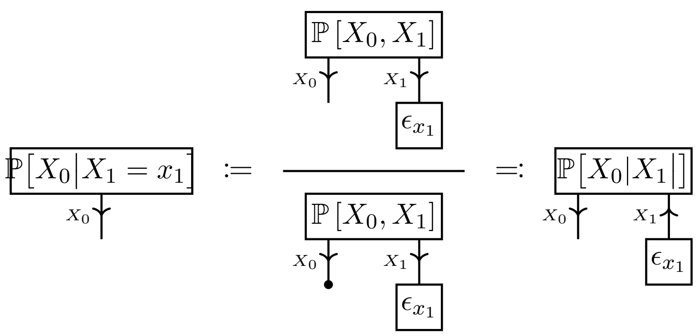
which is depicted by the diagram



The conditional distribution of X_0 on X_1 is the normalization (see Def. 2.8)

$$\mathbb{P}\left[X_0 \middle| X_1\right] := \langle \mathbb{P}[X_0, X_1] \rangle_{[X_0|X_1]} .$$

For $x_1 \in [m_1]$ with $\langle \mathbb{P}[X_0, X_1 = x_1] \rangle_{[\varnothing]}$ we depict the normalization by



3.2 The independence mechanism: Factorization into graphical models

The number of coordinates in a tensor representation of probability distributions is the product

$$\prod_{k\in[d]} m_k \,.$$

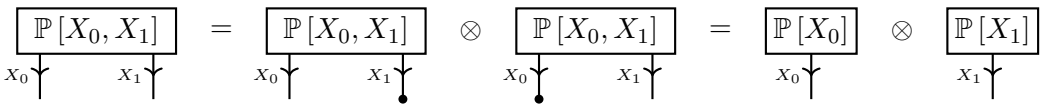
It therefore scales exponentially in the number of coordinates. To find efficient representation schemes of probability distributions by tensor networks, we need to exploit additional properties of the distribution. Independence leads to severe sparsifications of conditional probabilities and is hence the key assumption to gain sparse decompositions of probability distributions.

Definition 3.4 (Independence)**.** *We say that X_0 is independent of X_1 with respect to a distribution $\mathbb{P}[X_0, X_1]$ if the distribution is the tensor product of the marginal distributions, that is*

$$\mathbb{P}[X_0, X_1] = \mathbb{P}[X_0] \otimes \mathbb{P}[X_1] \,.$$

In this case we write $(X_0 \perp X_1)$.

Thus, independence appears directly as a tensor–product decomposition of probability distribution. Using tensor network diagrams, we depict this property by



Note that the assumption of independence reduces the degrees of freedom from $m_0 \cdot m_1 - 1$ to $(m_0 - 1) + (m_1 - 1)$. The decomposition into marginal distributions furthermore exploits this reduced freedom and provides an efficient storage. Having a joint distribution of multiple variables whose disjoint subsets are independent, we can iteratively apply the decomposition scheme. As a result, we can reduce the scaling of the degrees of freedom from exponential to linear by the assumption of independence.

As we observed, independence is a strong assumption, which is often too restrictive. Less demanding is conditional independence, which still implies efficient tensor network decomposition schemes. We introduce conditional independence as independence of variables with respect to conditional distributions.

Definition 3.5 (Conditional independence)**.** *Assume a joint distribution of variables X_0, X_1 and X_2 such that $\mathbb{P}[X_2]$ is positive. We say that X_0 is independent of X_1 conditioned on X_2 if*

$$\mathbb{P}\big[X_0, X_1 \big| X_2\big] = \big\langle \mathbb{P}\big[X_0 \big| X_2\big], \mathbb{P}\big[X_1 \big| X_2\big] \big\rangle_{[X_0, X_1, X_2]} \,.$$

In this case we write $(X_0 \perp X_1) \,|\, X_2$.

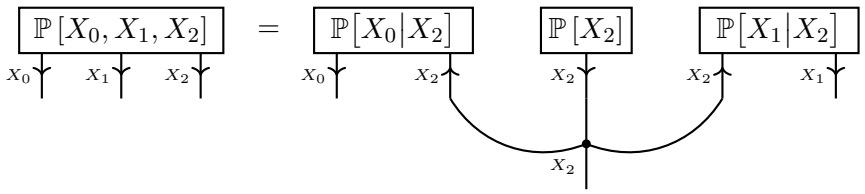
Conditional independence stated in Def. 3.5 has a close connection with independence stated in Def. 3.4. To be more precise, X_0 is independent of X_1 conditioned on X_2 if and only if X_0 is independent of X_1 with respect to any slice $\mathbb{P}\big[X_0, X_1 \big| X_2 = x_2\big]$ of the conditional distribution $\mathbb{P}\big[X_0, X_1 \big| X_2\big]$.

We can further exploit conditional independence to find tensor network decompositions of probabilities as we show in the next corollary.

Corollary 3.6. *Let $\mathbb{P}[X_0, X_1, X_2]$ be a joint distribution. If and only if X_0 is independent of X_1 conditioned on X_2, the distribution satisfies*

$$\mathbb{P}[X_0, X_1, X_2] = \left\langle \mathbb{P}\left[X_0 \middle| X_2\right], \mathbb{P}\left[X_1 \middle| X_2\right], \mathbb{P}[X_2] \right\rangle_{[X_0, X_1, X_2]} .$$

In a diagrammatic notation, this is depicted by



This conditional independence pattern is the basic local building block that is generalized in Markov networks, which we define in the following.

Definition 3.7 (Markov network)**.** *Let $\tau^{\mathcal{G}}$ be a tensor network of non-negative tensors decorating a hypergraph $\mathcal{G}$. Then the Markov Network $\mathbb{P}^{\mathcal{G}}$ to $\tau^{\mathcal{G}}$ is the probability distribution of X_v defined by the tensor*

$$\mathbb{P}^{\mathcal{G}}[X_{\mathcal{V}}] = \frac{\left\langle \{\tau^e \,:\, e \in \mathcal{E}\} \right\rangle_{[X_{\mathcal{V}}]}}{\left\langle \{\tau^e \,:\, e \in \mathcal{E}\} \right\rangle_{[\varnothing]}} = \left\langle \tau^{\mathcal{G}} \right\rangle_{[X_{\mathcal{V}}|\varnothing]} .$$

We call the denominator

$$\mathcal{Z}\left(\tau^{\mathcal{G}}\right) = \left\langle \{\tau^e \,:\, e \in \mathcal{E}\} \right\rangle_{[\varnothing]}$$

the partition function of the tensor network $\tau^{\mathcal{G}}$.

We define graphical models based on hypergraphs to establish a direct connection with tensor networks decorating the hypergraph. In a more canonical way, Markov Networks are instead defined by graphs, where instead of the edges the cliques are decorated by factor tensors (see for example [**?**]). The probabilistic graphical models are along that alternative dual to tensor networks [**?**, **?**].

We can interpret the factors $\tau\left[X_{[d]}\right]$ as activation cores placed on the hyperedges e of the graph. The global activation tensor (and hence the joint distribution) is obtained by contracting this activation network and normalizing by its partition function.

While so far we have defined Markov Networks as decomposed probability distributions, we now want to derive assumptions on a distribution, assuring that such decompositions exist. The sets of conditional independencies encoded by a hypergraph are captured by its separation properties, as we define next.

Definition 3.8 (Separation of hypergraph)**.** *A path in a hypergraph is a sequence of nodes v_k for $k \in [d]$, such that for any $k \in [d-1]$ we find a hyperedge $e \in \mathcal{E}$ such that $(v_k, v_{k+1}) \subset e$. Given disjoint subsets A, B, C of nodes in a hypergraph $\mathcal{G}$, we say that C separates A and B with respect to $\mathcal{G}$ when any path starting at a node in A and ending in a node in B contains a node in C.*

To characterize Markov Networks in terms of conditional independencies, we need to further define the property of clique-capturing. This property establishes a correspondence of hyperedges with maximal cliques in the more canonical graph-based definition of Markov Networks [**?**].

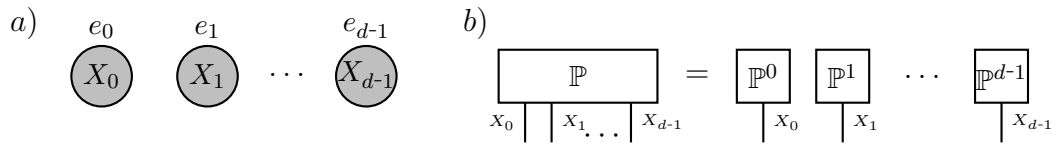


Figure 5: Decomposition of a probability distribution with independent variables (see Example 3.11). The independencies are captured by the elementary hypergraph a), whose edges contain single nodes. The corresponding tensor $\mathbb{P}\left[X_{[d]}\right]$ is then represented by a Markov Network on the elementary hypergraph, where each factor is the marginal distribution of the corresponding variable as visualized in b).

Definition 3.9 (Clique-capturing hypergraph)**.** *We call a hypergraph $\mathcal{G}$* clique-capturing*, if the following holds: Each subset $\mathcal{U} \subset \mathcal{V}$, which fulfills that for any $a, b \in \mathcal{U}$ there is a hyperedge $e \in \mathcal{E}$ with $a, b \in e$, is contained in a hyperedge.*

We now recall a characterization of Markov Networks in terms of conditional independencies.

Theorem 3.10 (Hammersley-Clifford factorization theorem)**.** *Let there be a positive probability distribution $\mathbb{P}\left[X_{\mathcal{V}}\right]$ and a clique-capturing hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Then the following are equivalent:*

i The distribution $\mathbb{P}\left[X_{\mathcal{V}}\right]$ is representable by a Markov Network on $\mathcal{G}$, that is for each edge $e \in \mathcal{E}$ there is a tensor $\tau^e\left[X_e\right]$ such that

$$\mathbb{P}\left[X_{\mathcal{V}}\right] = \left\langle \left\{ \tau^e\left[X_e\right] \,:\, e \in \mathcal{E} \right\} \right\rangle_{[X_{\mathcal{V}}|\varnothing]} .$$

ii For any subsets $A, B, C \subset \mathcal{V}$ such that C separates A from B, we have

$$(X_A \perp X_B) \,|\, X_C \,.$$

Proof. This is shown in Appendix Sect. A. □

By Thm. 3.10 the conditional independence structure of $\mathbb{P}\left[X_{\mathcal{V}}\right]$ determines a global tensor network decomposition of $\mathbb{P}\left[X_{\mathcal{V}}\right]$. We refer to this correspondence between independence structure and tensor network factorization as the *independence mechanism*. Note that the assumption of a positive distribution is required (i.e. for all $x_{[d]}$ we have $\mathbb{P}\left[X_{[d]} = x_{[d]}\right] > 0$). The assumption of positivity is however not required in our characterization of independencies and conditional independencies by the existence of corresponding tensor decompositions (see Def. 3.4 and Def. 3.5).

Example 3.11 (Independent boolean variables)**.** *Let there be d boolean variables $X_{[d]}$, which are i.i.d. drawn from a positive distribution $\mathbb{P}\left[X\right]$. From the pairwise independencies of X_k it follows with the Hammersley-Clifford Factorization Thm. 3.10 that the distribution is representable by an elementary tensor network, that is*

$$\mathbb{P}\left[X_{[d]}\right] = \bigotimes_{k \in [d]} \mathbb{P}^k\left[X_k\right] .$$

The corresponding hypergraph is the elementary graph, with respect to which any two disjoint subsets of nodes are separated (see Figure 5).

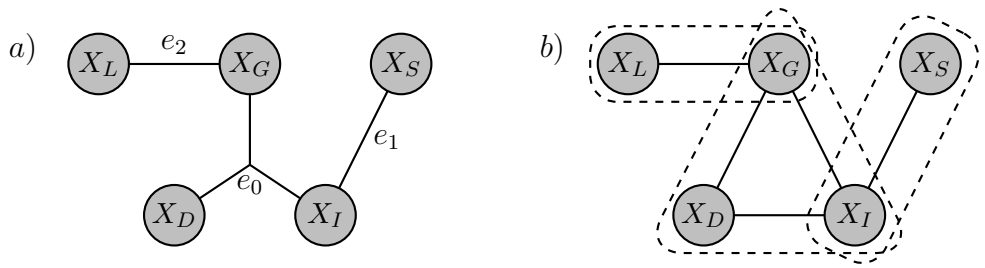


Figure 6: Hypergraph a) capturing the conditional independencies of the student example. The cliques of the node adjacency graph are highlighted in b) and coincide with hyperedges of the hypergraph. The hypergraph is therefore clique-capturing (see Def. 3.9).

Example 3.12. *We consider a classical example of a graphical model (see [**?**, Example 4.3]): A student of intelligence (X_I) and SAT score (X_S) is assigned a test of difficulty (X_D), for which he gets a grade (X_G) depending on which he gets a recommendation letter (X_L) by its teacher. We make the following modelling assumptions:*

- *"The SAT score depends only on the students intelligence: $\left(X_S \perp X_{\{D,G,L\}}\right) \mid X_I$.*
- *"The recommendation letter depends only on the grade: $\left(X_L \perp X_{\{D,I,S\}}\right) \mid X_G$.*

The associated hypergraph capturing these conditional independencies is drawn in Figure 6 a).

3.3 The computation mechanism: Factorization based on sufficient statistics

We now present the computation mechanism of finding tensor network decompositions of probability distributions.

Definition 3.13. *Let $\mathbb{P}\left[X, Z\right]$ be a joint distribution of the m-dimensional variable X and the n-dimensional variable Z and let*

$$t \,:\, [m] \to [n]$$

be a statistic. We are interested in the distribution $\mathbb{P}\left[X, Z, Y_t\right] = \left\langle \mathbb{P}\left[X, Z\right], \beta^t\left[Y_t, X\right]\right\rangle_{[X,Z,Y_t]}$. We say that t is a sufficient statistic for Z if X is independent of Z conditioned on Y_t.

Example 3.14 (Sufficient statistics for the probability)**.** *Let Z be the value $\mathbb{P}\left[X_{[d]} = x_{[d]}\right]$ when drawing $X_{[d]}$ from $\mathbb{P}\left[X_{[d]}\right]$. Then t is a sufficient statistic for $Z = \mathbb{P}\left[X_{[d]}\right]$ if for all y in the image of t we have*

$$\mathbb{P}\big[X_{[d]} = x_{[d]} \big| Y_t = y\big] = \begin{cases} \frac{1}{\left|\{x_{[d]} \,:\, t\left(x_{[d]}\right) = y\}\right|} & \text{if} \quad t(x_{[d]}) = y \\ 0 & \text{else} \end{cases}.$$

When knowing the value $t\left(x_{[d]}\right)$ of the sufficient statistic at a given index $x_{[d]}$, we then also know the probability $\mathbb{P}\left[X_{[d]} = x_{[d]}\right]$. The function t is thus a sufficient statistic for $Z = \mathbb{P}\left[X_{[d]}\right]$ if and only if there is a tensor $\xi\left[Y_{[p]}\right]$ with

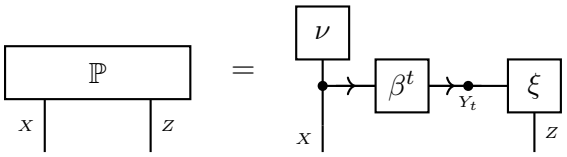
$$\mathbb{P}\left[X_{[d]}\right] = \left\langle \beta^t\left[Y_{[p]}, X_{[d]}\right], \xi\left[Y_{[p]}\right]\right\rangle_{\left[X_{[d]}\right]}.$$

Example 3.14 hints at a connection between sufficient statistics and decompositions into CompActNets. More generally, such decompositions are provided by the Fisher-Neyman Factorization Theorem.

Theorem 3.15 (Fisher-Neyman Factorization Theorem)**.** *Let $\mathbb{P}$ be a joint distribution of variables X, Z with values $\mathrm{val}(X)$, $\mathrm{val}(Z)$. Let there further be a finite set $\mathrm{val}(Y_t)$. Then $t : \mathrm{val}(X) \to \mathrm{val}(Y_t)$ is a sufficient statistic for Z if and only if there are tensors $\nu\left[X\right]$ and $\xi\left[Y_t, Z\right]$ such that*

$$\mathbb{P}\left[X, Z\right] = \left\langle \xi\left[Y_t, Z\right], \beta^t\left[Y_t, X\right], \nu\left[X\right] \right\rangle_{[X,Z]} .$$

We depict this equation diagrammatically by



Proof. Shown in more generality in the appendix, see Thm. A.3. □

Note that the definition of sufficient statistic does not make use of the marginal distribution $\mathbb{P}\left[Z\right]$. We therefore can define sufficient statistics also for families of distributions $\mathbb{P}\left[X \middle| Z\right]$ with respect to arbitrary non-degenerate marginal distributions $\mathbb{P}\left[Z\right]$. We then use Thm. 3.15 to embed such families in CompActNets.

Corollary 3.16. *Let $\mathbb{P}\left[X_{[d]} \middle| Z\right]$ be an arbitrary family of distributions of $X_{[d]}$ and t a sufficient statistic for Z. Then, there is a tensor $\nu\left[X_{[d]}\right]$ and an activation tensors $\xi\left[Y_{[p]}, Z\right]$ such that for any $z \in \mathrm{val}(Z)$ we have*

$$\mathbb{P}\left[X_{[d]} \middle| Z = z\right] \in \Lambda^{t,\mathrm{MAX},\xi} .$$

The Factorization Theorem of Fisher-Neyman provides the fundamental motivation for the CompActNets architecture. Any decomposition of $\beta^t\left[Y_{[p]}, X_{[d]}\right]$ is called a *computation network* and common to all members of a family with sufficient statistic t. The activation tensor $\xi\left[Y_{[p]}, Z = z\right]$, whose decomposition is called the *activation network*, is specific to each member of the family. We now show in two examples how families of distributions can be represented in CompActNets by sufficient statistics.

Example 3.17 (Order statistic for boolean variables)**.** *Let there be d boolean variables $X_{[d]}$ and a family $\mathbb{P}\left[X_{[d]} \middle| Z\right]$ of distributions. The order statistic assigns to each tuple $x_{[d]}$ the ordered tuple, which effectively counts the number of 1 coordinates in the tuple $x_{[d]}$, that is the statistic*

$$t^+ : \bigtimes_{k\in[d]} [m_k] \to [p] , \quad t^+(x_{[d]}) = |\{k : x_k = 1\}| .$$

When the order statistic is sufficient for Z, the detailed order of the outcomes is uninformative about the member z from which the random variables have been drawn. By the Fisher-Neyman Factorization Thm. 3.15, t^+ is a sufficient statistic if and only if there are tensors $\nu\left[X_{[d]}\right]$ and $\xi\left[Y_+, \Theta\right]$ such that for each $\theta \in \Theta$

$$\mathbb{P}\left[X_{[d]} \middle| Z = z\right] = \left\langle \xi\left[Y_+, Z = z\right], \beta^{t^+}\left[Y_+, X_{[d]}\right], \nu\left[X_{[d]}\right] \right\rangle_{\left[X_{[d]}\right]} .$$

For each member z of the family, the probability of each sequence $x_{[d]}$ is thus the product of a base measure factor $\nu\left[X_{[d]} = x_{[d]}\right]$ and a factor $\xi\left[Y_+ = +(x_{[d]}), Z = z\right]$ depending only on the count $+(x_{[d]})$ of 1 coordinates in $x_{[d]}$. We later continue this example in Example 3.20, where further interpretations to the case of i.i.d. variables are provided.

Example 3.18 (Graphical models as a special case of CompActNets)**.** *For graphical models we take the* identity statistic

$$\delta\big(x_{[d]}\big) = x_{[d]}$$

so that the image coordinates coincide with the variables and there are no non-trivial computation cores. The associated basis encoding is just the identity tensor

$$\beta^{\delta}\left[Y_{[d]}, X_{[d]}\right] = \delta\left[X_{[d]}, Y_{[d]}\right] .$$

Therefore, for any activation tensor $\xi\left[Y_{[p]}\right]$ *we obtain*

$$\mathbb{P}\left[X_{[d]}\right] = \left\langle \xi\left[Y_{[p]}\right], \beta^{\delta}\left[Y_{[d]}, X_{[d]}\right]\right\rangle_{\left[X_{[d]}|\varnothing\right]} = \left\langle \xi\left[X_{[d]}\right]\right\rangle_{\left[X_{[d]}|\varnothing\right]} .$$

Put differently, in the graphical–model case the activation tensor coincides with the joint distribution tensor. In this setting, structural properties of the distribution such as (conditional) independences can be read off as algebraic factorization patterns of the activation (and hence joint) tensor.

3.4 Exponential families

We now show that exponential families are specific instances of CompActNets, whose activation tensors have elementary decompositions. The importance of exponential families in statistics stems from their universal properties. A classical theorem by Pitman, Koopman and Darmois (see [**?**]) states, that whenever a family exhibits constant support and a finite sufficient statistic for arbitrary large data sets, then it is in an exponential family. For a discussion of further universal properties of exponential families such as the existence of priors and entropy maximizers, see [**?**].

Definition 3.19 (Exponential family)**.** *Given a base measure* ν *and a statistic*

$$t : \bigtimes_{k\in[d]} [m_k] \to \mathbb{R}^p$$

we enumerate for each coordinate $\ell \in [p]$ *the image* $\operatorname{im}(t_\ell)$ *by an interpretation map*

$$I_\ell : [|\operatorname{im}(t_\ell)|] \to \operatorname{im}(t_\ell) .$$

For any canonical parameter vector $\theta[L] \in \mathbb{R}^p$*, we build the activation cores* $\alpha^{\ell,\theta}[Y_\ell]$ *for each coordinate* $y_\ell \in [|\operatorname{im}(t_\ell)|]$ *by*

$$\alpha^{\ell,\theta}[Y_\ell = y_\ell] = \exp\left[\theta\left[L = \ell\right] \cdot I_\ell(y_\ell)\right]$$

and define the distribution

$$\mathbb{P}^{(t,\theta,\nu)}\left[X_{[d]}\right] = \left\langle \{\nu\left[X_{[d]}\right]\} \cup \{\beta^{t_\ell}\left[Y_\ell, X_{[d]}\right] : \ell \in [p]\} \cup \{\alpha^{\ell,\theta}\left[Y_\ell\right] : \ell \in [p]\}\right\rangle_{\left[X_{[d]}|\varnothing\right]} .$$

We then call the tensor $\mathbb{P}^{t,\nu}\left[X_{[d]}|\Theta\right]$ *with* $\operatorname{val}(\Theta) = \mathbb{R}^p$ *and slices for* $\theta \in \mathbb{R}^p$ *given by*

$$\mathbb{P}^{t,\nu}\left[X_{[d]}|\Theta = \theta\right] = \mathbb{P}^{(t,\theta,\nu)}\left[X_{[d]}\right]$$

the exponential family to the statistic t *and the base measure* ν*.*

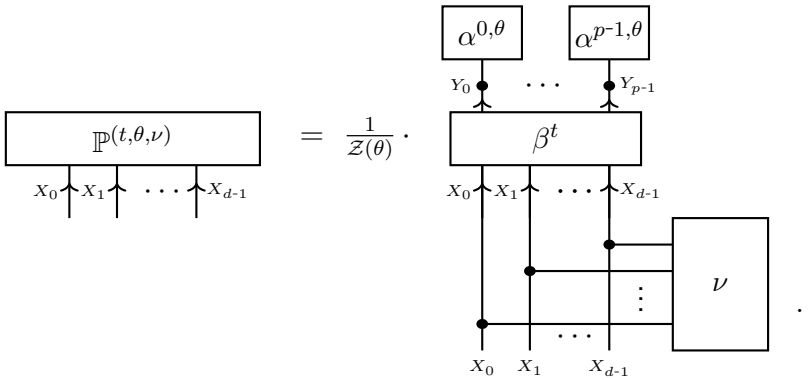


Figure 7: Tensor Network diagram of a member of an exponential family $\mathbb{P}^{t,\nu}\left[X_{[d]}|\Theta=\theta\right]$ before normalization as an CompActNet with elementary activation, that is an element in $\Lambda^{t,\mathrm{EL},\nu}$.

To see that Def. 3.19 is consistent with the typical definition of exponential families (see [**?**]), note that for each $\theta \in \mathbb{R}^p$ the normalization amounts to the division by

$$\mathcal{Z}(\theta) = \left\langle \{\nu\left[X_{[d]}\right]\} \cup \{\beta^{t_\ell}\left[Y_\ell, X_{[d]}\right] \,:\, \ell \in [p]\} \cup \{\alpha^{\ell,\theta}\left[Y_\ell\right] \,:\, \ell \in [p]\} \right\rangle_{[\varnothing]} ,$$

a quantity which is referred to as the partition function. Then, we have for each coordinate $x_{[d]} \in \bigtimes_{k\in[d]}[m_k]$ that

$$\begin{aligned}
&\mathbb{P}^{(t,\theta,\nu)}\left[X_{[d]} = x_{[d]}\right] \\
&= \frac{1}{\mathcal{Z}(\theta)} \left\langle \{\nu\left[X_{[d]}\right]\} \cup \{\beta^{t_\ell}\left[Y_\ell, X_{[d]}\right] \,:\, \ell \in [p]\} \cup \{\alpha^{\ell,\theta}\left[Y_\ell\right] \,:\, \ell \in [p]\} \right\rangle_{\left[X_{[d]}=x_{[d]}\right]} \\
&= \frac{1}{\mathcal{Z}(\theta)} \cdot \nu\left[X_{[d]} = x_{[d]}\right] \cdot \exp\left[\sum_{\ell\in[p]} \theta\left[L=\ell\right] \cdot t_\ell\left[X_{[d]} = x_{[d]}\right]\right] .
\end{aligned}$$

Note that by construction each member of an exponential family is an element in a CompActNet with elementary activation cores, that is

$$\mathbb{P}^{t,\nu}\left[X_{[d]}|\Theta=\theta\right] \in \Lambda^{t,\mathrm{EL},\nu} .$$

Example 3.20 (Exponential family of coin tosses)**.** *Recall the family of distributions of boolean $X_{[d]}$ from Example 3.17, which has the order statistic t^+ as a sufficient statistic. We now in addition assume that the variables $X_{[d]}$ are i.i.d. with respect to any member of the family (see Example 3.17). For the variables to be i.i.d., we need $\nu\left[X_{[d]}\right] = \mathbb{I}\left[X_{[d]}\right]$ and can thus choose a representation such that for $z \in \mathrm{val}(Z)$*

$$\mathbb{P}\left[X_{[d]}\middle|Z=z\right] = \left\langle \beta^{t^+}\left[Y_+, X_{[d]}\right], \xi\left[Y_+, Z=z\right] \right\rangle_{\left[X_{[d]}\right]}$$

where for each $k \in [d+1]$

$$\xi\left[Y_+ = k, Z = z\right] = (1-z)^{d-k} \cdot z^k .$$

The marginal distribution $\mathbb{P}\left[Y_+ \middle| Z = z\right]$ is then the binominal distribution $B(d, z)$. When excluding the case of $z \in \{0, 1\}$, the family is a subset of the exponential family with the head count statistic, where each member is reparametrized by

$$\theta \coloneqq \ln\left[\frac{z}{1-z}\right].$$

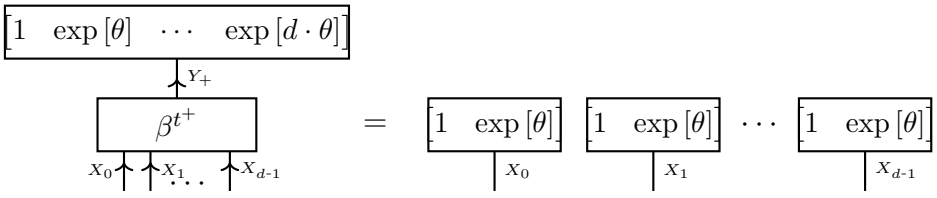
To see that this is true, we observe that the coordinate $y_+ \in [d+1]$ of the activation tensor of $\mathbb{P}^{(t^+,\theta,\mathbb{I})}$ is

$$\alpha^\theta\left[Y_+ = y_+\right] = \exp\left[y_+ \cdot \theta\right] = \frac{z^{y_+}}{(1-z)^{y_+}}.$$

Now, with $\mathcal{Z}(\theta) = \frac{1}{(1-z)^d}$ we have for any $x_{[d]}$ with $\sum_{k\in[d]} x_k = y_+$ that

$$\frac{1}{\mathcal{Z}(\theta)} \cdot \left\langle \beta^{t^+}\left[Y_+, X_{[d]} = x_{[d]}\right], \alpha^\theta\left[Y_+\right]\right\rangle_{[\varnothing]} = (1-z)^d \cdot \frac{z^{y_+}}{(1-z)^{y_+}} = z^{y_+} \cdot (1-z)^{d-y_+}.$$

Comparing with the activation tensor $\xi\left[Y_+\right]$ above, we note that $\mathcal{Z}(\theta)$ is the partition function of the exponential family and $\mathbb{P}^{(t^+,\theta,\mathbb{I})}[X_{[d]}]$ coincides with the member $\mathbb{P}\left[X_{[d]} \middle| Z = z\right]$. We further observe that since the statistic t^+ decomposes as a sum of terms depending on single variables only, we have a decomposition of the corresponding CompActNet by



This reproduces the fact that distributions of independent variables are representable by elementary tensors (see Example 3.11).

3.5 Efficient contractions by message passing

Contractions of tensor networks are generally hard to solve. Here, we investigate message passing algorithms, which decompose global contractions into a sequence of local contractions, whose results are passed as messages through the tensor network. The resulting algorithm is called Belief Propagation (see Algorithm 1). While various scheduling strategies for the message passing exist, we focus on the case of tree hypergraphs, for which exactness and efficiency can be shown. We denote $\mathcal{E}^{\rightarrow}$ to be all tuples (e_0, e_1) of hyperedges $e_0, e_1 \in \mathcal{E}$ such that $e_0 \neq e_1$ and $e_0 \cap e_1 \neq \varnothing$. For our purposes we call a hypergraph $\mathcal{G}$ a tree when the graph with nodes by the hyperedges $\mathcal{E}^{\rightarrow}$ and edges by $\mathcal{E}^{\rightarrow}$ is a tree (for an example see Figure 8).

For an implementation of Algorithm 1 in the python package $\mathrm{tnreason}$, see Sect. B.1.

The following theorem states that the contraction of a whole tensor network can be replaced by local contractions with messages. Since contracting the whole network can be infeasible, this shows that calculating the messages with the Algorithm 1 can be advantageous.

Algorithm 1 Tree Belief Propagation

Require: Tensor network $\tau^{\mathcal{G}}$ on a hypergraph $\mathcal{G}$
Ensure: Messages $\{\chi_{e_0\to e_1}[X_{e_0\cap e_1}] : (e_0,e_1)\in\mathcal{E}^{\to}\}$

Initialize $S=\{(e_2,e_0) : e_2 \text{ a leaf in } (\mathcal{E},\mathcal{E}^{\to})\}]$
while S not empty **do**
Pop a (e_0,e_1) pair from S
Compute the message

$$\chi_{e_0\to e_1}[X_{e_0\cap e_1}] = \langle\{\tau^{e_0}[X_{e_0}]\}\cup\{\chi_{e_2\to e_0}[X_{e_2\cap e_0}] : (e_2,e_0)\in\mathcal{E}^{\to},\, e_2\neq e_1\}\rangle_{[X_{e_0\cap e_1}]}$$

Update S by all messages (e_1,e_3) which have not yet been sent, if all messages (e_2,e_1) with $e_2\neq e_3$ have been sent.
end while
return Messages $\{\chi_{e_0\to e_1}[X_{e_0\cap e_1}] : (e_0,e_1)\in\mathcal{E}^{\to}\}$

Theorem 3.21. *Let $\tau^{\mathcal{G}}$ be a tensor network on a tree hypergraph $\mathcal{G}$ (i.e. the graph $(\mathcal{E},\mathcal{E}^{\to})$ is a tree). The messages in the tree belief propagation Algorithm 1 are contracted to local marginals, meaning that for each $e_0\in\mathcal{E}$ we have*

$$\left\langle\tau^{\mathcal{G}}\right\rangle_{[X_{e_0}]} = \langle\{\tau^{e_0}[X_{e_0}]\}\cup\{\chi_{e_2\to e_0}[X_{e_2\cap e_0}] : (e_2,e_0)\in\mathcal{E}^{\to}\}\rangle_{[X_{e_0}]}\,.$$

We show Thm. 3.21 based on the following lemma. We denote for each pair (e_0,e_1) the subset $\mathcal{E}^{\to(e_0,e_1)}\subset\mathcal{E}$ as the subset of edges $e\in\mathcal{E}$, for which each path to e_1 passes through e_0. The tree hypergraph property makes this definition equivalent to an existing path through e_0, which is used in the proof of the following lemma. Note that by construction $e_0\in\mathcal{E}^{\to(e_0,e_1)}$.

Lemma 3.22. *For any tensor network on a tree hypergraph, Algorithm 1 terminates in the tree-based implementation and returns final messages*

$$\chi_{e_0\to e_1}[X_{e_0\cap e_1}] = \left\langle\{\tau^{e}[X_e] : e\in\mathcal{E}^{\to(e_0,e_1)}\}\right\rangle_{[X_{e_0\cap e_1}]}\,.$$

Proof. We show this property by induction over the size of the edge sets $\mathcal{E}^{\to(e_0,e_1)}$ for pairs $(e_0,e_1)\in\mathcal{E}^{\to}$, such that $\left|\mathcal{E}^{\to(e_0,e_1)}\right|\leq n$. Note that since always $e_0\in\mathcal{E}^{\to(e_0,e_1)}$ we have that $n\geq 1$.

$n=1$: In this case we have $\mathcal{E}^{\to(e_0,e_1)}=\{e_0\}$ and e_0 is a leaf of the tree-hypergraph $\mathcal{G}$. The claimed message property holds thus by definition.

$n\to n+1$: Assume that the message obeys the claimed property for edge sets with cardinality up to n. If there is no edge set with cardinality $n+1$, the property holds also for those with cardinality up to $n+1$. If there is an edge set $\mathcal{E}^{\to(e_0,e_1)}$ with size $n+1$, we have

$$\mathcal{E}^{\to(e_0,e_1)} = \{e_0\}\cup\left(\bigcup_{e_2\in\mathcal{E}^{\to}}\mathcal{E}^{\to(e_2,e_0)}\right)\,.$$

The message $\chi_{e_0\to e_1}$ is sent once all messages $\chi_{e_2\to e_0}$ to $(e_2,e_0)\in\mathcal{E}^{\to}\{(e_1,e_0)\}$ arrived. By definition we have that

$$\chi_{e_0\to e_1}[X_{e_0\cap e_1}] = \langle\{\tau^{e_0}[X_{e_0}]\}\cup\{\chi_{e_2\to e_0}[X_{e_2\cap e_0}] : (e_2,e_0)\in\mathcal{E}^{\to},\, e_2\neq e_1\}\rangle_{[X_{e_0\cap e_1}]}\,.$$

Now we use the induction assumption on $\mathcal{E}^{\to(e_2,e_0)}$ (since its cardinality is at most n) and get

$$\begin{aligned}\chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right] &= \left\langle \left\{\tau^{e_0}\left[X_{e_0}\right]\right\} \cup \left(\bigcup_{(e_2,e_0)\in\mathcal{E}^{\to}, e_2\neq e_1} \left\langle \left\{\tau^{e_3}\left[X_{e_3}\right] \,:\, e_3\in\mathcal{E}^{\to(e_2,e_0)}\right\}\right\rangle_{\left[X_{e_2\cap e_0}\right]} \right)\right\rangle_{\left[X_{e_0\cap e_1}\right]} \\ &= \left\langle \left\{\tau^{e_0}\left[X_{e_0}\right]\right\} \cup \left(\bigcup_{(e_2,e_0)\in\mathcal{E}^{\to}, e_2\neq e_1} \left\{\tau^{e_3}\left[X_{e_3}\right] \,:\, e_3\in\mathcal{E}^{\to(e_2,e_0)}\right\} \right)\right\rangle_{\left[X_{e_0\cap e_1}\right]} \\ &= \left\langle \left\{\tau^{e}\left[X_{e}\right] \,:\, e\in\mathcal{E}^{\to(e_0,e_1)}\right\}\right\rangle_{\left[X_{e_0\cap e_1}\right]} .\end{aligned}$$

Here, we used the commutation of contraction property in the second equation, which is justified by the assumed tree property of the hypergraph. Thus, the message property holds also for any edge sets of size $n+1$.

By induction, the claimed message property therefore holds for all final messages. □

Proof of Thm. 3.21. Since the hypergraph is by assumption a tree, we can partition $\mathcal{E}$ into disjoint subsets $\{e_0\}$ and $\mathcal{E}^{\to(e_2,e_0)}$ for $(e_2,e_0)\in\mathcal{E}^{\to}$. We then have

$$\begin{aligned}\left\langle \tau^{\mathcal{G}}\right\rangle_{\left[X_{e_0}\right]} &= \left\langle \left\{\tau^{e_0}\left[X_{e_0}\right]\right\} \cup \left\{ \left\langle \tau^{e}\left[X_{e}\right] \,:\, e\in\mathcal{E}^{\to(e_2,e_0)}\right\rangle_{\left[X_{e\cap e_2}\right]} \,:\, (e_2,e_0)\in\mathcal{E}^{\to}\right\}\right\rangle_{\left[X_{e_0}\right]} \\ &= \left\langle \left\{\tau^{e_0}\left[X_{e_0}\right]\right\} \cup \left\{\chi_{e_2\to e_0}\left[X_{e_2\cap e_0}\right] \,:\, (e_2,e_0)\in\mathcal{E}^{\to}\right\}\right\rangle_{\left[X_{e_0}\right]} ,\end{aligned}$$

were we used Lem. 3.22 in the second equation. □

We illustrate the usage of Algorithm 1 on the Markov Network of Example 3.12.

Example 3.23 (Continuation of Example 3.12)**.** *We exemplify the Belief Propagation Algorithm 1 on the Markov Network in the student example (see Example 3.12). The directions of the messages result from the hyperedge overlaps (see Figure 8 a) and the resulting directions $\mathcal{E}^{\to}$ are sketched in Figure 8 b). The messages to $\{(e_2,e_0),(e_0,e_2)\}$ are vectors of X_G and the messages $\{(e_0,e_1),(e_1,e_0)\}$ are vectors of X_I.*

Since the hyperedges are minimally connected, we can implement Algorithm 1 by a tree scheduler S:

- *The scheduler is initialized with messages from leafs, in our example $\{(e_2,e_0),(e_1,e_0)\}$.*
- *Each message is placed exactly once on S, when at a hyperedge all but the reverse message have been received. In our example, after execution of (e_2,e_0) the message (e_0,e_1) is placed on S and after execution of (e_1,e_0) the message (e_0,e_2).*

In this implementation, Algorithm 1 terminates after $|\mathcal{E}^{\to}|=4$ iterations of the While *loop. The exact marginals of the edge variables are then*

$$\begin{aligned}\mathbb{P}\left[X_L,X_G\right] &= \left\langle \tau^{e_2}\left[X_L,X_G\right], \chi_{e_0\to e_2}\left[X_G\right]\right\rangle_{\left[X_L,X_G|\varnothing\right]} , \\ \mathbb{P}\left[X_G,X_D,X_I\right] &= \left\langle \tau^{e_0}\left[X_G,X_D,X_I\right], \chi_{e_2\to e_0}\left[X_G\right], \chi_{e_1\to e_0}\left[X_I\right]\right\rangle_{\left[X_G,X_D,X_I|\varnothing\right]} , \\ \mathbb{P}\left[X_I,X_S\right] &= \left\langle \tau^{e_1}\left[X_I,X_S\right], \chi_{e_0\to e_1}\left[X_I\right]\right\rangle_{\left[X_I,X_S|\varnothing\right]} .\end{aligned}$$

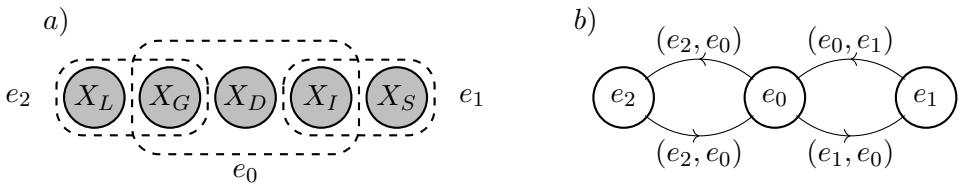


Figure 8: a) Sketch of the overlap of the edges, resulting in the message directions b) $\mathcal{E}^{\rightarrow} = \{(e_2, e_0), (e_0, e_2), (e_0, e_1), (e_1, e_0)\}$.

4 The neural paradigm

The neural paradigm of artificial intelligence exploits the decomposition of functions into neurons, which are aligned in a directed acyclic graph. We show in this section how functions decomposable into neurons can be represented by tensor networks. To this end, we formalize discrete neural models in decomposition graphs and formally prove the corresponding decomposition of their basis encodings. In particular, examples will be presented in the next section via propositional formulas with sparse syntactic descriptions.

4.1 Function decomposition

As a main principle of tensor decompositions, we now show that basis encodings of composition functions are contractions of the basis encodings of their components.

Lemma 4.1. *Let $f\left[X_{[d]}\right]$ be a composition of a p-ary connective function h and functions $f_\ell\left[X_{[d]}\right]$, where $\ell \in [p]$, i.e. for $x_{[d]} \in \bigtimes_{k\in[d]}[2]$ we have*

$$f(x_{[d]}) = h\left(f_0\left[x_{[d]}\right], \ldots, f_{p-1}\left[x_{[d]}\right]\right) .$$

Then, we have (see Figure 9)

$$\beta^f\left[Y_f, X_{[d]}\right] = \left\langle \{\beta^h\left[Y_f, Y_{[p]}\right]\} \cup \{\beta^{f_\ell}\left[Y_\ell, X_{[d]}\right] \,:\, \ell \in [p]\}\right\rangle_{\left[Y_f, X_{[d]}\right]} .$$

Proof. For any $x_{[d]} \in \bigtimes_{k\in[d]}[m_k]$ we have

$$\begin{aligned}
&\left\langle \{\beta^h\left[Y_f, Y_{[p]}\right]\} \cup \{\beta^{f_\ell}\left[Y_\ell, X_{[d]}\right] \,:\, \ell \in [p]\}\right\rangle_{\left[Y_f, X_{[d]}=x_{[d]}\right]} \\
&\quad = \left\langle \{\beta^h\left[Y_f, Y_{[p]}\right]\} \cup \{\beta^{f_\ell}\left[Y_\ell, X_{[d]} = x_{[d]}\right] \,:\, \ell \in [p]\}\right\rangle_{\left[Y_f\right]} \\
&\quad = \left\langle \{\beta^h\left[Y_f, Y_{[p]}\right]\} \cup \{\epsilon_{f_\ell(x_{[d]})}\left[Y_\ell\right] \,:\, \ell \in [p]\}\right\rangle_{\left[Y_f\right]} \\
&\quad = \epsilon_{f(x_{[d]})}\left[Y_f\right] \\
&\quad = \beta^f\left[Y_f, X_{[d]} = x_{[d]}\right] .
\end{aligned}$$

Thus, the tensors on both sides of the equation coincide in all slides to $X_{[d]}$ and are thus equal. □

We now define a more generic decomposition of discrete functions.

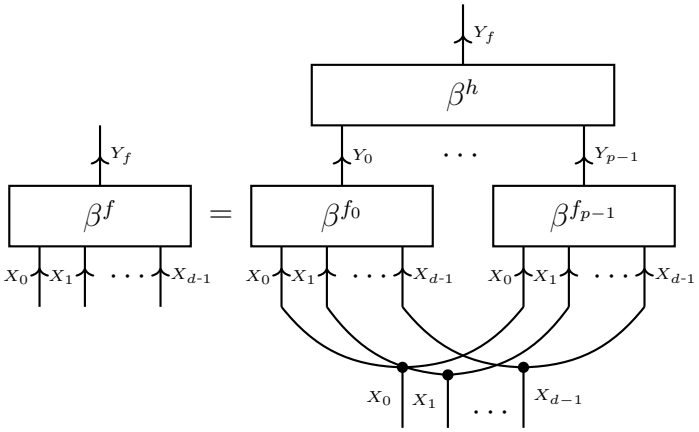


Figure 9: Tensor network decomposition of the basis encoding of a function f, which is the composition of the functions $f_0, \dots, f_{p-1}$ with a function h.

Definition 4.2. *A* decomposition hypergraph *is a directed acyclic hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that the following holds.*

- *Each node $v \in \mathcal{V}$ is decorated by a set $\mathcal{U}^v$ of finite cardinality m_v, a variable X_v, and an index interpretation function*

$$I_v : [m_v] \to \mathcal{U}^v .$$

- *Each directed hyperedge $(e^{\text{in}}, e^{\text{out}})$ has at least one outgoing node, i.e. $e^{\text{out}} \neq \varnothing$, and is decorated by an activation function*

$$g^e : \bigtimes_{v \in e^{\text{in}}} \mathcal{U}^v \to \bigtimes_{v \in e^{\text{out}}} \mathcal{U}^v .$$

- *Each node $v \in \mathcal{V}$ appears at most once as an outgoing node.*
- *The nodes not appearing as an outgoing node are enumerated by $v^{\text{in}}_{[d]}$. We abbreviate the corresponding variables by $X_{v^{\text{in}}_{[d]}} = X_{[d]}$.*
- *The nodes not appearing as an incoming node are enumerated by $v^{\text{out}}_{[p]}$. We abbreviate the variables by $X_{v^{\text{out}}_{[\ell]}} = Y_{[p]}$.*

We assign for each $k \in [d]$ restriction functions

$$\cdot|_{v^{\text{in}}_k} : \bigtimes_{\tilde{k} \in [d]} \mathcal{U}^{v^{\text{in}}_{\tilde{k}}} \to \mathcal{U}^{v^{\text{in}}_k} \quad , \quad x_{[d]}|_k = x_k$$

to the nodes $v^{\text{in}}_{[d]}$ and recursively assign to each further node v a node function

$$q_v : \bigtimes_{k \in [d]} \mathcal{U}^{v^{\text{in}}_k} \to \mathcal{U}^v \quad , \quad q_v(x_{[d]}) = g^{e_v}\left(\bigtimes_{v \in e^{\text{in}}} q_v(x_{[d]}) \right)|_v ,$$

where e_v is to each $v \in e^{\mathrm{in}}$ the unique hyperedge with outgoing nodes $\{v\}$. We then call the function

$$q_{\mathcal{G}} \,:\, \bigtimes_{k\in[d]} \mathcal{U}^{v_k^{\mathrm{in}}} \to \bigtimes_{\ell\in[p]} \mathcal{U}^{v_\ell^{\mathrm{out}}} \quad , \quad q_{\mathcal{G}} = \bigtimes_{\ell\in[p]} q_{v_\ell^{\mathrm{out}}}$$

the composition formula to the decomposition hypergraph $\mathcal{G}$.

The neural paradigm in AI can be modeled by the existence of decomposition hypergraphs for functions on large sets. We now show how decomposition hypergraphs enable the sparse representation of composition functions by tensor networks.

Theorem 4.3. *For any decomposition hypergraph $\mathcal{G}$ with composition formula $q_{\mathcal{G}}$, we have*

$$\beta^{q_{\mathcal{G}}}\left[Y_{[p]}, X_{[d]}\right] = \left\langle \left\{ \beta^{g^e}\left[X_{e^{\mathrm{out}}}, X_{e^{\mathrm{in}}}\right] \,:\, e = (e^{\mathrm{in}}, e^{\mathrm{out}}) \in \mathcal{E} \right\} \right\rangle_{\left[Y_{[p]}, X_{[d]}\right]} .$$

Proof. We show by induction over the number of edges in $\mathcal{G}$ that, for any $x_{[d]} \in \bigtimes_{k\in[d]}[m_k]$ and $v \in \mathcal{V}$, we have

$$\left\langle \left\{ \beta^{g^e}\left[X_{e^{\mathrm{out}}}, X_{e^{\mathrm{in}}}\right] \,:\, e = (e^{\mathrm{in}}, e^{\mathrm{out}}) \in \mathcal{E} \right\} \right\rangle_{\left[X_{\mathcal{V}\setminus v_{[d]}^{\mathrm{in}}}, X_{[d]} = x_{[d]}\right]} = \bigotimes_{v\in\mathcal{V}\setminus v_{[d]}^{\mathrm{in}}} \epsilon_{q_v(x_{[d]})}\left[X_v\right] . \tag{4.1}$$

$|\mathcal{E}| = 1$: If there is a single edge $e = (e^{\mathrm{in}}, e^{\mathrm{out}})$ in $\mathcal{E}$, we have $X_{[d]} = X_{e^{\mathrm{in}}}$ and $\mathcal{V}\setminus v_{[d]}^{\mathrm{in}} = e^{\mathrm{out}}$. In this case (4.1) holds since

$$\beta^{g^e}\left[X_{e^{\mathrm{out}}}, X_{e^{\mathrm{in}}} = x_{e^{\mathrm{in}}}\right] = \bigotimes_{v\in\mathcal{V}\setminus v_{[d]}^{\mathrm{in}}} \epsilon_{q_v(x_{[d]})}\left[X_v\right] .$$

$(|\mathcal{E}| = n) \Rightarrow (|\mathcal{E}| = n+1)$: Let us now assume that (4.1) holds for all graphs with $|\mathcal{E}| \leq n$ and let $\mathcal{G}$ be a graph with $|\mathcal{E}| = n+1$. We find an edge $\tilde{e} = (e^{\mathrm{in}}, e^{\mathrm{out}})$ such that all nodes in e^{in} are only appearing as outgoing nodes in other edges.

$$\begin{aligned}
&\left\langle \left\{ \beta^{g^e}\left[X_e\right] \,:\, e \in \mathcal{E} \right\} \right\rangle_{\left[X_{\mathcal{V}\setminus v_{[d]}^{\mathrm{in}}}, X_{[d]} = x_{[d]}\right]} \\
&\quad = \left\langle \beta^{\tilde{e}}\left[X_{e^{\mathrm{out}}}, X_{e^{\mathrm{in}}}\right], \left\langle \left\{ \beta^{g^e}\left[X_e\right] \,:\, e \in \mathcal{E}\setminus\{\tilde{e}\} \right\} \right\rangle_{\left[X_{\mathcal{V}\setminus\{v_{[d]}^{\mathrm{in}}\cup e^{\mathrm{out}}\}}, X_{[d]} = x_{[d]}\right]} \right\rangle_{\left[X_{\mathcal{V}\setminus v_{[d]}^{\mathrm{in}}}, X_{[d]} = x_{[d]}\right]} \\
&\quad = \left\langle \beta^{\tilde{e}}\left[X_{e^{\mathrm{out}}}, X_{e^{\mathrm{in}}}\right], \bigotimes_{v\in\mathcal{V}\setminus\{v_{[d]}^{\mathrm{in}}\cup e^{\mathrm{out}}\}} \epsilon_{q_v(x_{[d]})}\left[X_v\right] \right\rangle_{\left[X_{\mathcal{V}\setminus v_{[d]}^{\mathrm{in}}}, X_{[d]} = x_{[d]}\right]} \\
&\quad = \bigotimes_{v\in\mathcal{V}\setminus v_{[d]}^{\mathrm{in}}} \epsilon_{q_v(x_{[d]})}\left[X_v\right] .
\end{aligned}$$

Here, in the second equation we used the induction hypothesis on the subgraph $(\mathcal{V}, \mathcal{E}\setminus\{\tilde{e}\})$ with $|\mathcal{E}\setminus\{\tilde{e}\}| = n$ and a generic contraction property of basis encodings in the third equation.

Thus, (4.1) holds always and we have for any $x_{[d]} \in \bigtimes_{k\in[d]}[m_k]$ that

$$\begin{aligned}\left\langle \left\{ \beta^{g^e}\left[X_{e^{\mathrm{out}}}, X_{e^{\mathrm{in}}}\right] \,:\, e=(e^{\mathrm{in}},e^{\mathrm{out}})\in\mathcal{E}\right\}\right\rangle_{\left[Y_{[p]},X_{[d]}=x_{[d]}\right]} &= \left\langle \bigotimes_{v\in\mathcal{V}\setminus v^{\mathrm{in}}_{[d]}} \epsilon_{q_v(x_{[d]})}\left[X_v\right]\right\rangle_{\left[Y_{[p]},X_{[d]}\right]} \\ &= \bigotimes_{v\in v^{\mathrm{out}}_{[p]}} \epsilon_{q_v(x_{[d]})}\left[X_v\right] \\ &= \epsilon_{q_{\mathcal{G}}}\left[Y_{[p]}\right] \\ &= \beta^{q_{\mathcal{G}}}\left[Y_{[p]}, X_{[d]}=x_{[d]}\right]\,.\end{aligned}$$

Keeping $X_{[d]}$ open, the claim is established. □

When neurons have tunable parameters, we can discretize those by sets $\mathcal{U}^k$ and understand them as additional input variables.

Example 4.4 (Sum of integers in m-adic representation)**.** *We develop a tensor network representation of integer summations on the set $[m^d]=\{0,\dots,m^d-1\}$, where $m,d\in\mathbb{N}$,*

$$+\,:\,[m^d]\times[m^d]\to[m^{d+1}] \quad,\quad +(i,\tilde{i}) = i+\tilde{i}\,,$$

which have a m-adic representation of length d. We define an index interpretation map

$$I\,:\,\bigtimes_{k\in[d]}[m]\to[m^d] \quad,\quad I(x_{[d]}) = \sum_{k\in[d]} x_k\cdot m^k\,,$$

which enables the parameterization of $[m^d]$ as the states of d categorical variables $X_{[d]}$ of dimension m. We analogously represent a second set $[m^d]$ by variables $\tilde{X}_{[d]}$ and the set $[m^{d+1}]$ of possible sums by $Y_{[d+1]}$. The basis encoding of the sum is then

$$\beta^{+}\left[Y_{[d+1]},X_{[d]},Y_{[d]}\right] = \sum_{x_{[d]},\tilde{x}_{[d]}} \epsilon_{I^{-1}(I(x_{[d]})+I(\tilde{x}_{[d]}))}\left[Y_{[d+1]}\right]\otimes\epsilon_{x_{[d]}}\left[X_{[d]}\right]\otimes\epsilon_{\tilde{x}_{[d]}}\left[\tilde{X}_{[d]}\right]\,.$$

Note that the tensor space of β^+ is of dimension $m^{3\cdot d+1}$ increasing exponentially in d. Feasible representation of this tensor for large d therefore requires tensor network decompositions, which we now provide based on a decomposition hypergraph. The targeted function to be decomposed is the representation of the integer sum by

$$+^m\,:\,\left(\bigtimes_{k\in[d]}[m]\right)\times\left(\bigtimes_{k\in[d]}[m]\right)\to\bigtimes_{k\in[d+1]}[m] \quad,\quad +^m(x_{[d]},\tilde{x}_{[d]}) = I^{-1}(I(x_{[d]})+I(\tilde{x}_{[d]}))\,.$$

We build a decomposition hypergraph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ (see Def. 4.2) consisting of $4\cdot d$ nodes (see Figure 10a) . The nodes carry the $(3\cdot d+1)$ variables $X_{[d]},Y_{[d]},Y_{[d+1]}$ of dimension m constructed above and $d-1$ auxiliary variables $Z_{[d-1]}$ of dimension 2 representing carry bits. The directed hyperedges of $\mathcal{G}$ are

$$\begin{aligned}\mathcal{E} = &\left\{(\{X_0,\tilde{X}_0\},\{Y_0,Z_0\})\right\}\cup\left\{(\{Z_{k-1},X_k,\tilde{X}_k\},\{Y_k,Z_k\})\,:\,k\in\{1,\dots,d-2\}\right\}\\ &\cup\left\{(\{Z_{d-2},X_{d-1},\tilde{X}_{d-1}\},\{Y_{d-1},Y_d\})\right\}\end{aligned}$$

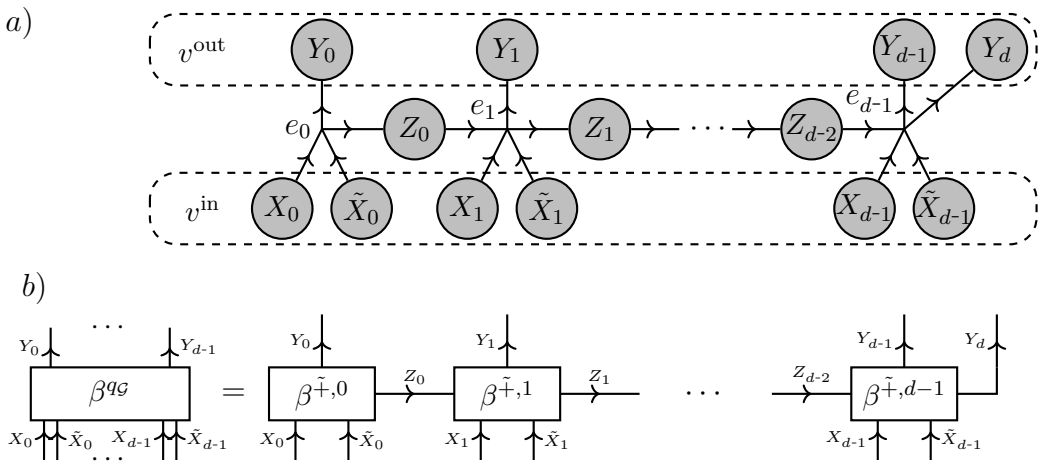


Figure 10: Example of a decomposition hypergraph to the sum of integers (see Example 4.4). a) Hypergraph of directed edges e_k for $k \in [d]$, each decorated by an integer summation $+$ preparing an index Y_k of the resulting sum. b) Corresponding tensor network decomposition of the basis encoded composition function, which is the sum of integers in m-adic representation.

and are decorated by local summation functions

$$\tilde{+} : [2] \times [m] \times [m] \to [m] \times [2] \quad , \quad \tilde{+}(z, x, \tilde{x}) = \left((z + x + \tilde{x}) \mod m, \left\lfloor \frac{z + x + \tilde{x}}{m} \right\rfloor \right) .$$

Since to the first hyperedge we do not have a carry bit, the decorating function is the restriction of the first argument to 0*.*

It is known that the composition of the local summations $\tilde{+}$ *is the global summation* $+^m$ *of integers in* m*-adic representation. Thus, the composition function* $q_\mathcal{G}$ *is* $+^m$*. By Thm. 4.3 we have a decomposition of the basis encoding to* $q_\mathcal{G}$ *(see Figure 10b) as*

$$\begin{aligned} \beta^{+^m}\left[Y_{[d+1]}, X_{[d]}, \tilde{X}_{[d]}\right] = \big\langle \{\beta^{\tilde{+},0}\left[Y_0, Z_0, X_0, \tilde{X}_0\right]\} \cup \\ \{\beta^{\tilde{+},k}\left[Y_k, Z_k, X_k, \tilde{X}_k, Z_{k-1}\right] : k \in \{1, \ldots, d-2\}\} \cup \\ \{\beta^{\tilde{+},d-2}\left[Y_{d-1}, Y_d, X_{d-1}, \tilde{X}_{d-1}, Z_{d-2}\right]\}\big\rangle_{\left[Y_{[d+1]}, X_{[d]}, \tilde{X}_{[d]}\right]} . \end{aligned}$$

4.2 Function evaluation by message passing

We are now concerned with an efficient inference algorithm based on tensor network contractions. To evaluate a function given as a tensor network decomposition of its basis encoding, the whole network has to be contracted. As this can be infeasible for large networks, a message passing algorithm based on local contractions can be applied, compare Algorithm 1 for a message passing algorithm for tensor networks on a on tree hypergraph.

We apply the Directed Belief Propagation, Algorithm Algorithm 2, on a decomposition hypergraph, where we add hyperedges to each leaf node and assign one-hot encodings of input states. We then show that the messages are the one-hot encodings to the evaluations of the node functions.

Algorithm 2 Directed Belief Propagation

Require: Tensor network $\tau^{\mathcal{G}}$ on a directed hypergraph $\mathcal{G}$
Ensure: Messages $\{\chi_{e_0\to e_1}[X_{e_0\cap e_1}] : (e_0,e_1)\in\mathcal{E}^{\to}\}$

Prepare directed message directions

$$\mathcal{E}^{\to} = \left\{\left((e_0^{\mathrm{in}},e_0^{\mathrm{out}}),(e_1^{\mathrm{in}},e_1^{\mathrm{out}})\right) : e_0^{\mathrm{in}}\cap(e_1^{\mathrm{in}},e_1^{\mathrm{out}})=\varnothing,\ e_1^{\mathrm{out}}\cap(e_0^{\mathrm{in}},e_0^{\mathrm{out}})=\varnothing,\ e_0^{\mathrm{out}}\cap e_1^{\mathrm{in}}\neq\varnothing\right\}$$

Initialize a message queue $S=\{(e_2,e_0) : e_2 \text{ has empty incoming nodes}\}$
while S not empty **do**
Pop a (e_0,e_1) pair from S
Update the message

$$\chi_{e_0\to e_1}[X_{e_0\cap e_1}] = \langle\{\tau^{e_0}[X_{e_0}]\}\cup\{\chi_{e_2\to e_0}[X_{e_2\cap e_0}] : (e_2,e_0)\in\mathcal{E}^{\to},\ e_2\neq e_1\}\rangle_{[X_{e_0\cap e_1}]}$$

Update S by all messages (e_1,e_3) which have not yet been sent, if all messages (e_2,e_1) have been sent.
end while
return Messages $\{\chi_{e_0\to e_1}[X_{e_0\cap e_1}] : (e_2,e_0)\in\mathcal{E}^{\to}\}$

Theorem 4.5. *Let $\mathcal{G}$ be a decomposition graph and let us add hyperedges containing single input nodes, which are decorated by one-hot encodings. Then the messages computed in Algorithm 2 are characterized by*

$$\chi_{e_0\to e_1}[X_{e_0\cap e_1}] = \bigotimes_{v\in e_0\cap e_1}\epsilon_{q_v(x_{[d]})}[X_v]\ .$$

Proof. We show the theorem inductively over the messages computed in Algorithm 2. The first message is sent from an input edge $\{[k]\}$ to an edge e of the decomposition graph and is by assumption the one-hot encoding of an input state $\epsilon_{x_k}[X_k]$.

We now assume that all previous messages satisfy the claimed equation at an arbitrary stage of the algorithm. The message computed in the while loop is then a contraction of one-hot encodings with basis encodings and

$$\begin{aligned}\chi_{e_0\to e_1}[X_{e_0\cap e_1}] &= \left\langle\{\beta^{g^{e_0}}[X_{e^{\mathrm{out}}},X_{e^{\mathrm{in}}}]\}\cup\{\chi_{e_2\to e_0}[X_{e_2\cap e_0}] : (e_2,e_0)\in\mathcal{E}^{\to}\}\right\rangle_{[X_{e_0\cap e_1}]}\\ &= \left\langle\{\beta^{g^{e_0}}[X_{e^{\mathrm{out}}},X_{e^{\mathrm{in}}}]\}\cup\{\epsilon_{q_v(x_{[d]})}[X_v] : v\in e^{\mathrm{in}}\}\right\rangle_{[X_{e_0\cap e_1}]}\\ &= \bigotimes_{v\in e_0\cap e_1}\epsilon_{q_v(x_{[d]})}[X_v]\ .\end{aligned}$$

Thus, the new message is also the tensor product of the one-hot encodings of the evaluated node functions. By induction, the property is therefore true for all messages. □

We notice that we can interpret any directed acyclic hypergraph for which each node appears exactly once as an outgoing node and which is decorated by boolean and directed tensors $\tau^{\mathcal{G}}$. Edges with empty incoming sets are carrying one-hot encodings of input states and all further edges carry functions.

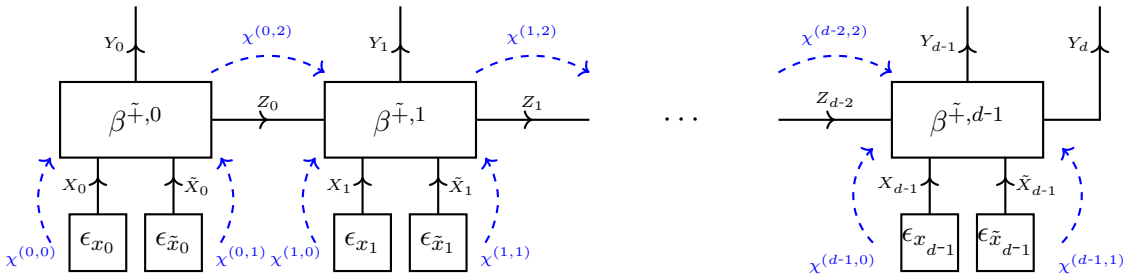


Figure 11: Computation of the integer sum in m-adic representation by the Directed Belief Propagation Algorithm 2 (see Example 4.6). The summands are represented by one-hot encodings of the digits $x_{[d]}$ and $\tilde{x}_{[d]}$, from which the messages start. The k-th digit (for $k \in \{0, \dots, d-1\}$) of the sum is computed based on the first messages of the epoch labeled by $\chi^{(k,[2])}$, The third message $\chi^{(k,2)}$ in each epoch communicates the carry bit to the next digit summation core. In the last message epoch the digit $d-1$ and d are computed based.

Example 4.6 (Continuation of Example 4.4)**.** *We now show how Algorithm 2 can be exploited to compute an efficient message passing algorithm for the digits of the m-adic sum. Given two numbers in m-adic representation by the tuples $x_{[d]}$ and $\tilde{x}_{[d]}$, we add the hyperedges with empty incoming nodes and single outgoing node*

$$\bigcup_{k\in[d]} \Big\{(\varnothing, \{X_k\}), (\varnothing, \{\tilde{X}_k\})\Big\}$$

to the hypergraph of Example 4.4 and decorate them by the digit one-hot encodings $\epsilon_{x_k}[X_k]$ and $\epsilon_{\tilde{x}_k}\left[\tilde{X}_k\right]$ (see Figure 11). We then apply the Directed Belief Propagation Algorithm 2. The initial messages queue then consists of the messages from the digit encoding. As sketched in Figure 11, to each digit there are three messages (with the exception of the first being two), which can be scheduled in consecutive epochs $\chi^{(k,[3])}$. We then have by Thm. 4.5 for $k \in [d-1]$ that

$$\left\langle \beta^{\tilde{+},k}\left[Y_k, Z_k, X_k, \tilde{X}_k, Z_{k-1}\right], \chi^{(k-1,2)}[Z_{k-1}], \chi^{(k,0)}[X_k], \chi^{(k,1)}[\tilde{X}_k]\right\rangle_{[Z_k]} = \epsilon_{z_k}[Z_k] \ ,$$

where z_k is the value of the k-th carry bit. The k-th digit of the sum y_k can further be obtained by the contraction

$$\left\langle \beta^{\tilde{+},k}\left[Y_k, Z_k, X_k, \tilde{X}_k, Z_{k-1}\right], \chi^{(k-1,2)}[Z_{k-1}], \chi^{(k,0)}[X_k], \chi^{(k,1)}[\tilde{X}_k]\right\rangle_{[Y_k]} = \epsilon_{y_k}[Y_k] \ .$$

Note that the hypergraph representing this instance is a tree and by Thm. 3.21 also the message passing scheme of Algorithm 1 is guaranteed to produce the exact contractions. We can exploit this fact for example in the efficient computation of averages of the summation digits, when we have an elementary distribution of input digits. We emphasize that the directed belief propagation Algorithm 1 is exact even if the hypergraph fails to be a tree, provided that we have directed and boolean tensors..

5 The logical paradigm

A tensor-based representation of propositional logic is developed by encoding boolean variables into vectors, defining formulas as boolean tensors, and showing how logical connectives and normal forms can be expressed as tensor contractions.

5.1 Propositional semantics by boolean tensors

Starting with the introduction of propositional formulas as boolean tensors their decomposition is discussed with respect to a basis encoding. The basis encoding representation enables dealing with the negation operation of propositional formulas still in tensor format. Furthermore, the propositional formula then fits into the concept of CompActNets.

INTRO

Definition 5.1. *A* propositional formula $f\left[X_{[d]}\right]$ *depending on* d *boolean variables* X_k *is a boolean-valued tensor*

$$f\left[X_{[d]}\right] : \bigtimes_{k\in[d]} [2] \to \{0,1\} \subset \mathbb{R}\,.$$

We call a state $x_{[d]} \in \bigtimes_{k\in[d]}[2]$ *a* model *of a propositional formula* f*, if*

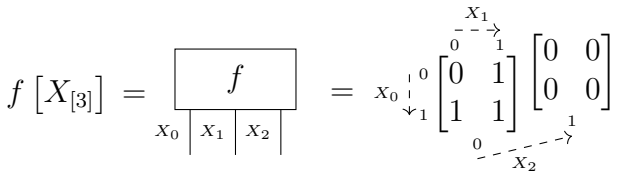
$$f\left[X_{[d]} = x_{[d]}\right] = 1\,,$$

where we associate $\mathrm{True} \leftrightarrow 1$ *and* $\mathrm{False} \leftrightarrow 0$*. If there is a model to a propositional formula, we say the formula is* satisfiable*.*

Example 5.2. *Let there be* $d = 3$ *boolean variables* $X_{[3]}$ *and a propositional formula*

$$f\left[X_{[3]}\right] = (X_0 \vee X_1) \wedge \neg X_2\,.$$

In a graphical depiction and in the coordinatewise representation this formula can be represented as



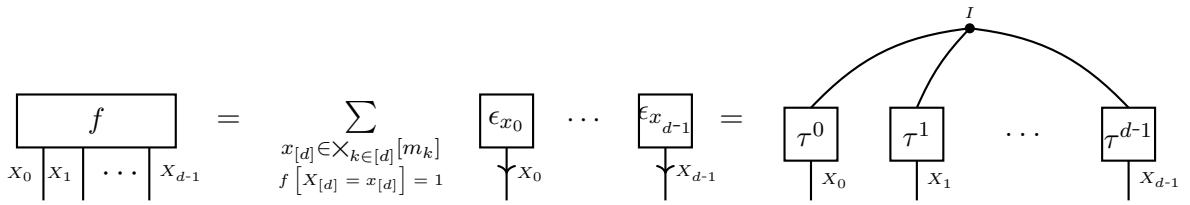
In the state set $\bigtimes_{k\in[d]}[2] = \{0,1\} \times \{0,1\} \times \{0,1\}$ *we have three models of the formula by the positions of the non-zero entries in the tensor, i.e.* $f\left[X_{[3]} = x_{[3]}\right] = 1$ *if and only if*

$$x_{[3]} \in \{(1,0,0),(0,1,0),(1,1,0)\}\,.$$

The formula f *is therefore satisfiable.*

Model counts by contraction Each coordinate of the propositional formula is either a 1 or 0, indicating whether the indexed state is a model of the formula or not. In this way, the contraction $\langle f \rangle_{[\varnothing]}$ counts the number of models of the propositional formula f. One can therefore decide the satisfiability of a formula by testing for $\langle f \rangle_{[\varnothing]} > 0$.

CP decomposition Since the tensor $f\left[X_{[d]}\right]$ is equal to one at index $x_{[d]}$ if and only if $x_{[d]}$ is a model of f, a propositional formula can be written as the sum over the one-hot encodings of its models.



As already depicted, one can exploit this summation to find a CP decomposition of the formula. To this end, we enumerate the models $x^i_{[d]}$ of f by a decomposition variable I with values $i \in [\langle f \rangle_{[\varnothing]}]$ and define, for $k \in [d]$, cores with slices

$$\tau^k [X_k, I = i] = \epsilon_{x^i_k} [X_k] \ .$$

Example 5.3. *For the formula described in Example 5.2, we have*

$$\begin{aligned} f \left[X_{[3]}\right] = &(\epsilon_1 [X_0] \otimes \epsilon_0 [X_1] \otimes \epsilon_0 [X_2]) + (\epsilon_0 [X_0] \otimes \epsilon_1 [X_1] \otimes \epsilon_0 [X_2]) \\ &+ (\epsilon_1 [X_0] \otimes \epsilon_1 [X_1] \otimes \epsilon_0 [X_2]) \, . \end{aligned}$$

Note that we have $\left\langle f \left[X_{[d]}\right] \right\rangle_{[\varnothing]} = 3$ *and we can interpret this sum as a* CP *decomposition of* f *with rank* 3*. We use the decomposition to evaluate the formula* f *at* $x_{[3]} = (1, 1, 0)$ *and get*

$$\begin{aligned} f \left[X_{[3]} = x_{[3]}\right] &= (\epsilon_1 [X_0 = 1] \otimes \epsilon_0 [X_1 = 1] \otimes \epsilon_0 [X_2 = 0]) \\ &\quad + (\epsilon_0 [X_0 = 1] \otimes \epsilon_1 [X_1 = 1] \otimes \epsilon_0 [X_2 = 0]) \\ &\quad + (\epsilon_1 [X_0 = 1] \otimes \epsilon_1 [X_1 = 1] \otimes \epsilon_0 [X_2 = 0]) \\ &= 1 \cdot 0 \cdot 1 + 0 \cdot 1 \cdot 1 + 1 \cdot 1 \cdot 1 = 1 \, , \end{aligned}$$

which verifies that $x_{[3]} = (1, 1, 0)$ *is a model of the formula* f*.*

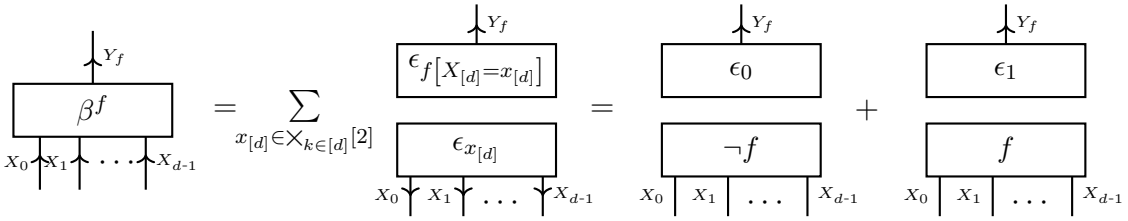
Basis encoding Representing Booleans by elements in $\{0, 1\}$ leads to the problem that negation is an affine transformation and cannot be represented by multilinear tensors. Therefore, instead of using this *coordinate calculus* an approach based on *basis calculus* is employed, which is explained in this section. To be able to express different kinds of connectives and, finally, any propositional formula by multilinear tensors, booleans are encoded by one-hot encodings as defined in Def. 2.9. Propositional formulas f can be expressed in terms of a tensor describing the mapping and its negation by

$$\beta^f \left[Y_f = y_f, X_{[d]} = x_{[d]}\right] = \begin{cases} 1 & \text{if} \quad f \left[X_{[d]} = x_{[d]}\right] = y_f \\ 0 & \text{else} \end{cases} \, . \tag{5.1}$$

This basis encoding $\beta^f \left[Y_f, X_{[d]}\right] \in \{0, 1\}^{2 \times 2^d}$ then has the form

$$\beta^f \left[Y_f, X_{[d]}\right] = \epsilon_1 [Y_f] \otimes f \left[X_{[d]}\right] + \epsilon_0 [Y_f] \otimes \neg f \left[X_{[d]}\right] \, . \tag{5.2}$$

In our graphical notation this property is visualized by



We further provide a more detailed example in coordinate sensitive notation in the following.

Example 5.4 (Logical negation and conjunction)**.** *The basis encodings of the negation* $\neg : [2] \to [2]$ *is the matrix*

$$\beta^{\neg}\left[Y_{\neg}, X\right] = \begin{array}{c} \overset{Y_\neg}{\overset{0 \quad 1}{\dashrightarrow}} \\ X_0 \begin{array}{|} 0 \\ \downarrow 1 \end{array} \end{array} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The 2*-ary conjunctions* $\wedge : [2] \times [2] \to [2]$ *is encoded by the order-*3 *tensor*

$$\beta^{\wedge}\left[Y_{\wedge}, X_0, X_1\right] = \begin{array}{c} \\ Y_\wedge \begin{array}{|} 0 \\ \downarrow 1 \end{array} \end{array} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{array}{c} \overset{X_1}{\overset{0 \quad 1}{\dashrightarrow}} \\ X_0 \begin{array}{|} 0 \\ \downarrow 1 \end{array} \end{array} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} + \begin{array}{c} \\ Y_\wedge \begin{array}{|} 0 \\ \downarrow 1 \end{array} \end{array} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{array}{c} \overset{X_1}{\overset{0 \quad 1}{\dashrightarrow}} \\ X_0 \begin{array}{|} 0 \\ \downarrow 1 \end{array} \end{array} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{array}{c} \overset{X_1}{\overset{0 \quad 1}{\dashrightarrow}} \\ X_0 \begin{array}{|} 0 \\ \downarrow 1 \end{array} \end{array} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{array}{c} \\ \underset{Y_\wedge}{\overset{0 \quad 1}{\dashrightarrow}} \end{array}$$

Further, the 2*-ary disjunction* $\vee : [2] \times [2] \to [2]$ *is encoded by the order-*3 *tensor*

$$\beta^{\vee}\left[Y_{\vee}, X_0, X_1\right] = \begin{array}{c} \overset{X_1}{\overset{0 \quad 1}{\dashrightarrow}} \\ X_0 \begin{array}{|} 0 \\ \downarrow 1 \end{array} \end{array} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{array}{c} \\ \underset{Y_\vee}{\overset{0 \quad 1}{\dashrightarrow}} \end{array}$$

Interpretation as CompActNets The propositional formula and its negation can be represented by this tensor via

$$f\left[X_{[d]}\right] = \left\langle \epsilon_1\left[Y_f\right], \beta^f\left[Y_f, X_{[d]}\right] \right\rangle_{\left[X_{[d]}\right]} \quad \text{and} \quad \neg f\left[X_{[d]}\right] = \left\langle \epsilon_0\left[Y_f\right], \beta^f\left[Y_f, X_{[d]}\right] \right\rangle_{\left[X_{[d]}\right]}.$$

Both f and $\neg f$ are thus Computation-Activation Networks to the statistic $\{f\}$ and the hard activation tensor $\epsilon_1\left[Y_f\right]$, respectively $\epsilon_0\left[Y_f\right]$.

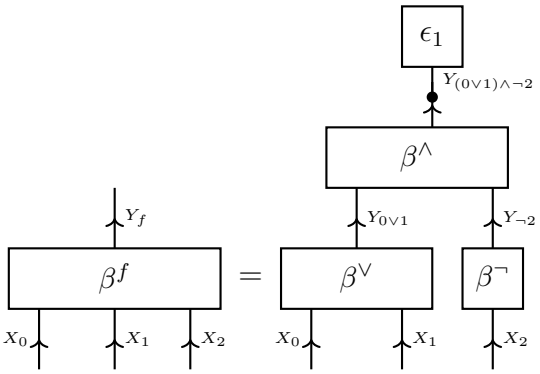
5.2 Syntactical decomposition of propositional formulas

Propositional formulas of concern often have a syntactic specification as composed functions. We can therefore apply the neural paradigm to find efficient representations of them.

Definition 5.5 (Syntactic decompositions)**.** *A syntactic decomposition of a propositional formula* f *is a decomposition hypergraph (see Def. 4.2) such that all nodes are decorated with the dimension* $m_v = 2$ *and the composition function* f*.*

Thus we have a tensor network representation of any propositional formula based on its syntactic decomposition, where the hypergraph of the syntactic decomposition equals the hypergraph of the representing tensor network.

Example 5.6. *For the formula* $f\left[X_{[3]}\right] = (X_0 \vee X_1) \wedge \neg X_2$ *from Example 5.2, we have the following syntactical decomposition of its basis encoding:*



5.3 Entailment decision by contractions

We have already seen that the contraction of a propositional formula counts its models. This allows to define entailment between two propositional formulas as follows. To generalize the treatment, we no longer demand that the variables of a formula are of dimension 2. We further use $\neg f\left[X_{[d]}\right] = \mathbb{I}\left[X_{[d]}\right] - f\left[X_{[d]}\right]$.

Definition 5.7 (Entailment of propositional formulas)**.** *Given two propositional formulas $\mathcal{KB}$ and f we say that $\mathcal{KB}$ entails f, denoted by $\mathcal{KB} \models f$, if any model of $\mathcal{KB}$ is also a model of f, that is*

$$\left\langle \mathcal{KB}\left[X_{[d]}\right], \neg f\left[X_{[d]}\right]\right\rangle_{[\varnothing]} = 0\,.$$

If $\mathcal{KB} \models \neg f$ holds (i.e. $\left\langle \mathcal{KB}\left[X_{[d]}\right], f\left[X_{[d]}\right]\right\rangle_{[\varnothing]}$=0), we say that $\mathcal{KB}$ contradicts f.

Classically (see e.g. [**?**]) entailment in propositional logics is defined as the unsatisfiability of $\mathcal{KB}\wedge\neg f$. This is equivalent to Def. 5.7, since $\left\langle \mathcal{KB}\left[X_{[d]}\right], \neg f\left[X_{[d]}\right]\right\rangle_{[\varnothing]} = 0$ is equivalent to $\left\langle (\mathcal{KB}\wedge(\neg f))[X_{[d]}]\right\rangle_{[\varnothing]} = 0$, which is the unsatisfiability of $\mathcal{KB}\wedge\neg f$.

Example 5.8 ($n^2 \times n^2$ Sudoku)**.** *We index the rows and the columns by tuples $(r0, r1)$ and $(co, c1)$, where $r0, r1, c0, c1 \in [n]$. The first index indicates the block and the second counts the row or column inside that block. For each $r0, r1, c0, c1 \in [n]$ and $i \in [n^2]$ we then define an atomic variable $X_{r0,r1,c0,c1,i} \in \{0, 1\}$ indicating whether in the row $(r0, r1)$ and column $(co, c1)$ the number i is written. The Sudoku rules then amount to the formula*

$$\mathcal{KB}^n := \left(\bigwedge_{r0,r1,c0,c1\in[n]}\left(\bigoplus_{i\in[n^2]}^{(1)} X_{r0,r1,c0,c1,i}\right)\right)\wedge\left(\bigwedge_{r0,r1\in[n],i\in[n^2]}\left(\bigoplus_{c0,c1\in[n]}^{(1)} X_{r0,r1,c0,c1,i}\right)\right)\wedge$$
$$\left(\bigwedge_{c0,c1\in[n],i\in[n^2]}\left(\bigoplus_{c0,c1\in[n]}^{(1)} X_{r0,r1,c0,c1,i}\right)\right)\wedge\left(\bigwedge_{r0,c0\in[n],i\in[n^2]}\left(\bigoplus_{r1,c1\in[n]}^{(1)} X_{r0,r1,c0,c1,i}\right)\right),$$

where $\bigoplus^{(1)}$ is the n^2-ary exclusive or connective (that is 1 if and only if exactly one of the arguments is 1). The four outer brackets in $\mathcal{KB}$ mark the constraints, that at each position exactly one number is assigned, further that in each row each number is assigned once, and similar for the columns and the squares of the board. When solving a specific Sudoku instance, one typically knows from an initial board assignment E^{start} a collection of atomic variables, which hold, and needs to find further atomic variables, which are entailed. This means, we need to decide for each $(r_0, r_1, c_0, c_1, i) \notin$

E^{start} whether the Sudoku rules and the initial board imply that the atomic variable $X_{r0,r1,c0,c1,i}$ (i.e. assignment to the board) is true

$$\mathcal{KB}^n \wedge \left(\bigwedge_{(r_0,r_1,c_0,c_1,i)\in E^{\mathrm{start}}} X_{r0,r1,c0,c1,i} \right) \models X_{r0,r1,c0,c1,i}$$

or false

$$\mathcal{KB} \wedge \left(\bigwedge_{(r_0,r_1,c_0,c_1,i)\in E^{\mathrm{start}}} X_{r0,r1,c0,c1,i} \right) \models \neg X_{r0,r1,c0,c1,i}\,. \tag{5.3}$$

If and only if the Sudoku has a unique solution given the initial board assignment E^{start}, exactly one of these entailment statements holds for each $(r_0, r_1, c_0, c_1, i) \notin E^{\mathrm{start}}$. Deciding which is equivalent to solving of the Sudoku.

For a more detailed example, let $n = 2$ and

$$E^{\mathrm{start}} = \{(0,0,0,0,0),(0,0,1,0,2),(0,0,1,1,1),(0,1,0,1,1),\\ (1,0,1,0,3),(1,1,0,0,3),(1,1,0,1,2)\}\,.$$

We visualize this evidence by writing $i{+}1$ in a grid cell $(r0, r1, c0, c1)$ to indicate that $(r0, r1, c0, c1, i) \in E^{\mathrm{start}}$:

1		3	2
	2		
		4	
4	3		

We will later demonstrate in Example 5.8 a solution algorithm to solve this instance, after we have derived sparse tensor network representations in Example 5.12.

5.4 Efficient representation of kowledge bases

We now investigate the representation of propositional knowledge bases $\mathcal{KB} = \{f_\ell : \ell \in [p]\}$, which are sets of propositional formulas f_ℓ. The conjunction of these formulas is the knowledge base formula

$$\mathcal{KB}\left[X_{[d]}\right] = \bigwedge_{\ell\in[p]} f_\ell\left[X_{[d]}\right]\,.$$

To show efficient representations, we will use the following identities.

Lemma 5.9 (Computation Network Symmetries)**.** *We have for the d-ary $\wedge$-connective (where $d \in \mathbb{N}$) and the unary $\neg$-connective that*

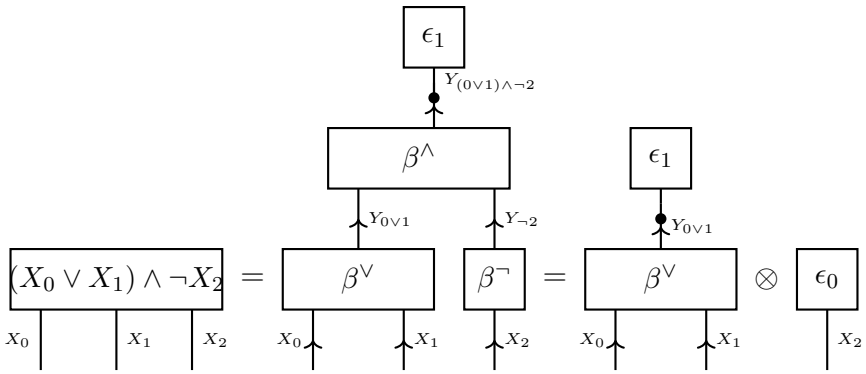
$$\left\langle \epsilon_1\left[Y\right], \beta^{\wedge}\left[Y, X_{[d]}\right]\right\rangle_{\left[X_{[d]}\right]} = \bigotimes_{k\in[d]} \epsilon_1\left[X_k\right] \quad \textit{and} \quad \left\langle \epsilon_1\left[Y\right], \beta^{\neg}\left[Y, X\right]\right\rangle_{[X]} = \epsilon_0\left[X\right]\,.$$

Proof. Follows directly from the definitions of the basis encodings and the connectives. □

Example 5.10. *For the propositional formula from Example 5.2*

$$f\left[X_{[3]}\right] = (X_0 \vee X_1) \wedge \neg X_2\,,$$

we can write the formula in terms of a Computation-Activation Network with activation tensor ϵ_1 and computation network decomposed by the basis encodings. First, it is written with one activation vector. Second, we see that it can also be interpreted with multiple features.



We use this to decompose knowledge bases into their individual formulas as follows.

Theorem 5.11. *For any knowledge base $\mathcal{KB}\left[X_{[d]}\right] = \bigwedge_{\ell\in[p]} f_\ell\left[X_{[d]}\right]$ it holds that*

$$\mathcal{KB}\left[X_{[d]}\right] = \left\langle \left\{ f_\ell\left[X_{[d]}\right] \,:\, \ell \in [p] \right\} \right\rangle_{\left[X_{[d]}\right]}.$$

Proof. With Lem. 5.9 we have

$$\begin{aligned}
\mathcal{KB}\left[X_{[d]}\right] &= \left\langle \left\{ \epsilon_1\left[Y_\wedge\right], \beta^\wedge\left[Y_\wedge, Y_{[p]}\right] \right\} \cup \left\{ \beta^{f_\ell}\left[Y_\ell, X_{[d]}\right] \,:\, \ell \in [p] \right\} \right\rangle_{\left[X_{[d]}\right]} \\
&= \left\langle \bigcup_{\ell\in[p]} \left\{ \epsilon_1\left[Y_\ell\right], \beta^{f_\ell}\left[Y_\ell, X_{[d]}\right] \,:\, \ell \in [p] \right\} \right\rangle_{\left[X_{[d]}\right]} \\
&= \left\langle \left\{ f_\ell\left[X_{[d]}\right] \,:\, \ell \in [p] \right\} \right\rangle_{\left[X_{[d]}\right]}.
\end{aligned}$$

□

Example 5.12 (Sparse representation of Sudoku rule knowledge base)**.** *We now exploit Thm. 5.11 to find efficient tensor network representation of the Sudoku knowledge base from Example 5.8. We directly get, that the knowledge base $\mathcal{KB}^n$ of Sudoku rules is a tensor network of the $4 \cdot n^4$ constraint formulas using the n^2-ary connective $\bigoplus^{(1)}$, and the evidence E^{start} can be encoded by vectors $\epsilon_1\left[X_{(r_0,r_1,c_0,c_1,i)}\right]$. To get a representation by matrices instead of tensors of order n^2, we introduce a hidden variable I taking values in $[n^2]$ for each of the constraints, one can further increase the sparsity of the representation. With the usage of matrices*

$$\tau^k\left[X_k, I\right] = \epsilon_0\left[X_k\right] \otimes \mathbb{I}\left[I\right] + \left(\epsilon_1\left[X_k\right] - \epsilon_0\left[X_k\right]\right) \otimes \epsilon_k\left[I\right]$$

we have the decomposition

$$\bigoplus^{(1)}[X_{[n^2]}] = \left\langle \left\{ \tau^k\left[X_k, I\right] \,:\, k \in [n^2] \right\} \right\rangle_{\left[X_{[n^2]}\right]},$$

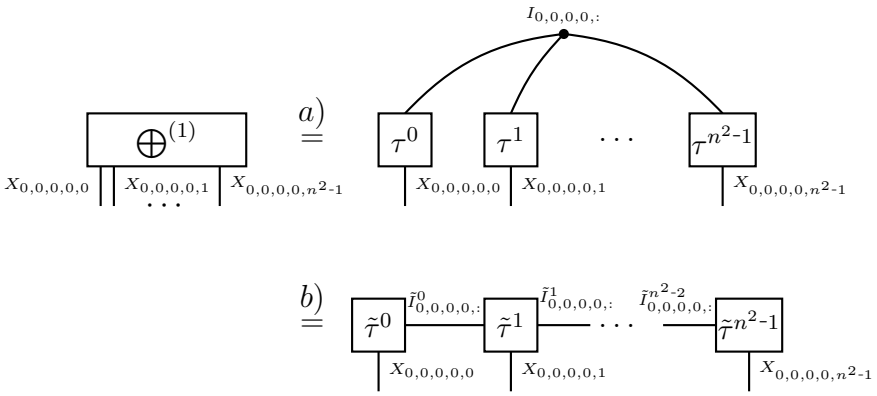


Figure 12: Decomposition of the position constraint $\bigoplus^{(1)}$ at position $(r0, r1, c0, c1) = (0, 0, 0, 0)$ into a) a CP decomposition with hidden variable $I_{0,0,0,0,:}$ and b) a TT decomposition with $d-1$ hidden variables $I^k_{0,0,0,0,:}, k \in [d-1]$.

which is a CP *decomposition (see Example 2.4) depicted in Figure 12 a).*

Alternatively there is a TT *decomposition (see Example 2.5) of the constraint* $\bigoplus^{(1)}$*, which we depict in Figure 12 b). We introduce for* $k \in [d-1]$ *hidden variables* I^k *of dimension* 2*, which are interpreted as the indicator, whether one of the variables* $X_{[k]}$ *is true. Following this interpretation we introduce* TT *cores*

$$\tilde{\tau}^0\left[X_0, \tilde{I}^0\right] = \epsilon_1\left[X_0\right] \otimes \epsilon_1\left[\tilde{I}^0\right] + \epsilon_0\left[X_0\right] \otimes \epsilon_0\left[\tilde{I}^0\right]$$
$$\tilde{\tau}^{d-1}\left[\tilde{I}^{d-2}, X_{d-1}\right] = \epsilon_0\left[\tilde{I}^{d-2}\right] \otimes \epsilon_1\left[X_{d-1}\right] + \epsilon_1\left[\tilde{I}^{d-2}\right] \otimes \epsilon_0\left[X_{d-1}\right]$$

and for $k \in \{1, \ldots, d-2\}$

$$\tilde{\tau}^k\left[\tilde{I}^{k-1}, X_k, \tilde{I}^k\right] = \delta\left[\tilde{I}^{k-1}, \tilde{I}^k\right] \otimes \epsilon_0\left[X_k\right] + \epsilon_0\left[\tilde{I}^{k-1}\right] \otimes \epsilon_1\left[X_k\right] \otimes \epsilon_1\left[\tilde{I}^{k-1}\right] .$$

We notice, that the TT *decomposition of the constraint* $\bigoplus^{(1)}$ *introduces* $d-1$ *many hidden variables of dimension* 2*, whereas the* CP *decomposition introduces a single hidden variable of dimension* d*. In the following we will further apply the* CP *decomposition.*

Given evidence E^{start} *we denote the Sudoku Knowledge Base* $\mathcal{KB}^{n,E^{\mathrm{start}}}$*. We model the Sudoku Knowledge Base* $\mathcal{KB}^{n,E^{\mathrm{start}}}$ *as a tensor network on a hypergraph* $\mathcal{G}^{\mathrm{Sudoku},n}$ *consistent in*

- $n^6+4\cdot n^4$ *nodes by* n^6 *categorical variables* $X_{(r0,r1,c0,c1,i)}$ *and by* $4\cdot n^4$ *decomposition variables to the constraints*
- $5 \cdot n^6$ *edges*

$$\mathcal{E} = \bigcup_{r0,r1,c0,c1 \in [n]} \Big\{\{X_{(r0,r1,c0,c1,i)}\}, \{X_{(r0,r1,c0,c1,i)}, I_{r0,r1,c0,c1,:}\}, \{X_{(r0,r1,c0,c1,i)}, I_{r0,r1,:,:,i}\},$$
$$\{X_{(r0,r1,c0,c1,i)}, I_{:,:,c0,c1,i}\}, \{X_{(r0,r1,c0,c1,i)}, I_{r0,:,c0,:,i}\}\Big\}$$

We denote the decomposition variables to the position, row, column and square constraints by $I_{r0,r1,c0,c1,:}, I_{r0,r1,:,:,i}, I_{:,:,c0,c1,i}$ *and* $I_{r0,:,c0,:,i}$*.*

Each edge containing a decomposition variable is decorated by a matrix $\tau^k[X, I]$ corresponding to a core in the CP *decomposition of a constraint. Here k is determined by the tuple $(r0, r1, c0, c1, i)$ and the type of the constraint (for example, for the variable $X_{(0,1,1,2,1)}$ and the row constraint $I_{(0,1,:,:,1)}$ we have $k = 1 \cdot n + 2$. We further assign to each edge containing a single variable $\{X_{(r0,r1,c0,c1,i)}\}$ either the vector $\epsilon_1\left[X_{(r0,r1,c0,c1,i)}\right]$ if $(r0, r1, c0, c1, i) \in E^{\text{start}}$ or the trivial vector $\mathbb{I}\left[X_{(r0,r1,c0,c1,i)}\right]$.*

5.5 Entailment decision by message passing

Since contracting the whole tensor network is often infeasible, local contractions can be considered to decide entailment in some cases. Here a local contraction describes the calculation of contractions along few closely connected tensors in the network. Before presenting the resulting Constraint Propagation algorithm, we first show two important properties of local entailment motivating the procedure.

Theorem 5.13 (Monotonicity of propositional logics)**.** *If $\tilde{\mathcal{KB}} \subset \mathcal{KB}$ and $\tilde{\mathcal{KB}} \models f$ then also $\mathcal{KB} \models f$.*

Proof. Since $\tilde{\mathcal{KB}} \models f$ it holds that $\left\langle \tilde{\mathcal{KB}}[X_{[d]}], \neg f[X_{[d]}] \right\rangle_{[\varnothing]} = 0$ and thus

$$\left\langle \tilde{\mathcal{KB}}[X_{[d]}], \neg f[X_{[d]}] \right\rangle_{\left[X_{[d]}\right]} = 0\left[X_{[d]}\right] .$$

Denoting by $\mathcal{KB}/\tilde{\mathcal{KB}}$ the conjunctions of formulas in $\mathcal{KB}$ not in $\tilde{\mathcal{KB}}$, we have

$$\begin{aligned}
\left\langle \mathcal{KB}\left[X_{[d]}\right], \neg f\left[X_{[d]}\right] \right\rangle_{[\varnothing]} &= \left\langle (\mathcal{KB}/\tilde{\mathcal{KB}})[X_{[d]}], \tilde{\mathcal{KB}}[X_{[d]}], \neg f\left[X_{[d]}\right] \right\rangle_{[\varnothing]} \\
&= \left\langle (\mathcal{KB}/\tilde{\mathcal{KB}})[X_{[d]}], \left\langle \tilde{\mathcal{KB}}[X_{[d]}], \neg f\left[X_{[d]}\right] \right\rangle_{\left[X_{[d]}\right]} \right\rangle_{[\varnothing]} \\
&= \left\langle (\mathcal{KB}/\tilde{\mathcal{KB}})[X_{[d]}], 0\left[X_{[d]}\right] \right\rangle_{[\varnothing]} \\
&= 0 .
\end{aligned}$$

□

To decide entailment, we can therefore investigate entailment on smaller parts of the knowledge base. This is sound by the above theorem, but not complete, since it can happen that no smaller part of the knowledge base entails the formula, but the whole knowledge base does. We can furthermore add entailed formulas to the knowledge base without changing it, as we show next.

Theorem 5.14 (Invariance of adding entailed formulas)**.** *If and only if $\mathcal{KB} \models f$ we have*

$$\mathcal{KB}\left[X_{[d]}\right] = \left\langle \mathcal{KB}\left[X_{[d]}\right], f\left[X_{[d]}\right] \right\rangle_{\left[X_{[d]}\right]} .$$

Proof. We use that $f\left[X_{[d]}\right] + \neg f\left[X_{[d]}\right] = \mathbb{I}\left[X_{[d]}\right]$ and thus

$$\begin{aligned}
\mathcal{KB}\left[X_{[d]}\right] &= \left\langle \mathcal{KB}\left[X_{[d]}\right], (f\left[X_{[d]}\right] + \neg f\left[X_{[d]}\right]) \right\rangle_{\left[X_{[d]}\right]} \\
&= \left\langle \mathcal{KB}\left[X_{[d]}\right], f\left[X_{[d]}\right] \right\rangle_{\left[X_{[d]}\right]} + \left\langle \mathcal{KB}\left[X_{[d]}\right], \neg f\left[X_{[d]}\right] \right\rangle_{\left[X_{[d]}\right]}
\end{aligned}$$

Since $\left\langle \mathcal{KB}\left[X_{[d]}\right], \neg f\left[X_{[d]}\right]\right\rangle_{\left[X_{[d]}\right]}$ is boolean, we thus have that

$$\mathcal{KB}\left[X_{[d]}\right] = \left\langle \mathcal{KB}\left[X_{[d]}\right], f\left[X_{[d]}\right]\right\rangle_{\left[X_{[d]}\right]}$$

if and only if $\left\langle \mathcal{KB}\left[X_{[d]}\right], \neg f\left[X_{[d]}\right]\right\rangle_{[\varnothing]} = 0$, that is $\mathcal{KB} \models f$. □

The mechanism of Thm. 5.14 provides us with a means to store entailment information in small-order auxiliary tensors. One way to exploit this accessibility of local entailment information are message passing schemes similar to Algorithm 1 propagating the information. This approach decides local entailment by iteratively adding entailed formulas to the knowledge base and checking further entailment on neighboring tensors of the knowledge base. Since for entailment decisions the support of the contractions is sufficient, we can apply non-zero indicators before sending contraction messages. We then schedule new messages in the direction (e_0, e_1), once the support of a message received at e_0 has been changed. Note that such a scheduling system is guaranteed to converge, since there can only be a finite number of message changes. We further directly reduce the computation of messages to their support and call the resulting Algorithm 3 Constraint Propagation.

Algorithm 3 Constraint Propagation

Require: Tensor network $\tau^{\mathcal{G}}$ on a hypergraph $\mathcal{G}$
Ensure: Messages $\{\chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right] : (e_0, e_1) \in \mathcal{E}^{\to}\}$ containing entailment statements

Initialize a queue $S = \mathcal{E}^{\to}$ of message directions
Initialize messages $\chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right] = \mathbb{I}\left[X_{e_0\cap e_1}\right]$ for $(e_0, e_1) \in \mathcal{E}^{\to}$
while S not empty **do**
  Pop a (e_0, e_1) pair from S
  Update the message

$$\chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right] = \mathbb{I}_{\neq 0}\left(\left\langle \{\tau^{e_0}\left[X_{e_0}\right]\} \cup \{\chi_{e_2\to e_0}\left[X_{e_2\cap e_0}\right] : (e_2, e_0) \in \mathcal{E}^{\to},\, e_2 \neq e_1\}\right\rangle_{\left[X_{e_0\cap e_1}\right]}\right)$$

  if $\tau\left[X_{e_0\cap e_1}\right] \neq \chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right]$ **then**
    Update the message: $\chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right] \coloneqq \tau\left[X_{e_0\cap e_1}\right]$
    Add $S = S \cup \{(e_1, e_2) : (e_1, e_2) \in \mathcal{E}^{\to}\}$
  end if
end while
return Messages $\{\chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right] : (e_0, e_1) \in \mathcal{E}^{\to}\}$

Theorem 5.15. *All messages during constraint propagation are sound, that is for all* $(e_0, e_1) \in \mathcal{E}^{\to}$ *it holds that*

$$\mathbb{I}_{\neq 0}\left(\left\langle \tau^{\mathcal{G}}\right\rangle_{\left[X_{e_0\cap e_1}\right]}\right) \prec \chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right].$$

Proof. We show this theorem by induction over the While loop of Algorithm 3. At the first iteration, we have for all messages $\chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right] = \mathbb{I}\left[X_{e_0\cap e_1}\right]$ and thus

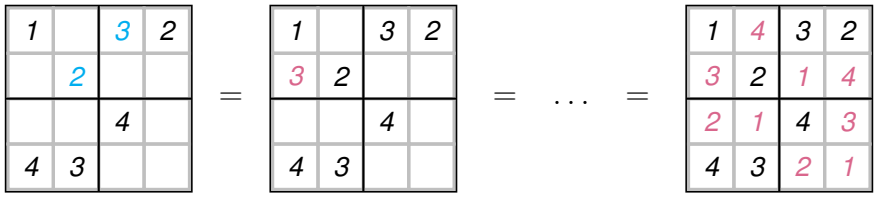
$$\tau^{\mathcal{G}} = \left\langle \{\tau^{\mathcal{G}}\} \cup \{\chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right] : (e_0, e_1) \in \mathcal{E}^{\to}\}\right\rangle_{[X_{\mathcal{V}}]}. \tag{5.4}$$

By Thm. 5.13 we then have for the first message send along the pair (e_0, e_1) that

$$\begin{aligned}\mathbb{I}_{\neq 0}\left(\left\langle \tau^{\mathcal{G}}\right\rangle_{\left[X_{e_0\cap e_1}\right]}\right) \prec & \mathbb{I}_{\neq 0}\left(\left\langle \{\tau^{e_0}\left[X_{e_0}\right]\} \cup \{\chi_{e_2\to e_0}\left[X_{e_2\cap e_0}\right] : (e_2, e_0) \in \mathcal{E}^{\to},\, e_2 \neq e_1\}\right\rangle_{\left[X_{e_0\cap e_1}\right]}\right) \\ & = \chi_{e_0\to e_1}\left[X_{e_0\cap e_1}\right].\end{aligned}$$

Let us now assume that at an arbitrary state of the algorithm the inequality holds for all previously sent messages. By Thm. 5.14 we can contract the messages with the tensor network without changing it, and (5.4) thus still holds. We then conclude with Thm. 5.13 that the claimed property also holds for the new message. □

Example 5.16 (Message passing for the Sudoku instance of Example 5.8)**.** *We iteratively solve a Sudoku puzzle by determining a possible value based on neighboring cells, rows and squares (using Thm. 5.13) and adding to our knowledge (using Thm. 5.14). For example, consider the following* $n = 2$ *Sudoku puzzle, where a first entailment step uses only the knowledge of the rules and the blue cells to determine the value* 3 *in the first square:*



To illustrate the first reasoning step of assigning $X_{0,1,0,0,2}$ *we make the following entailment steps applying Thm. 5.13. We also depict in Figure 13 the corresponding messages in the Constraint Propagation Algorithm on the hypergraph* $\mathcal{G}^{\mathrm{Sudoku},n}$*.*

- *From* $X_{0,1,0,1,1}$ *(i.e. the* 2 *in the cell* $(0,1,0,1)$*) and the Sudoku rule that at the cell* $(0,1,0,1)$ *exactly one number is assigned, we get*

$$\left(\bigoplus_{i\in[n^2]}^{(1)} X_{0,1,0,1,i}\right) \wedge X_{0,1,0,1,1} \models \neg X_{0,1,0,1,2}\,,$$

That is, that the number 3 *is not in the cell* $(0,1,0,1)$*. This entailment step is performed by three consecutive messages (see* $\chi^{(0,[3])}$ *in Figure 13) along the directions*

$$\begin{aligned}(e_0,e_1) \in \big[&(\{X_{0,1,0,0,1,1}\},\{X_{0,1,0,0,1,1},I_{0,1,0,1,:}\}),(\{X_{0,1,0,0,1,1},I_{0,1,0,1,:}\},\{X_{0,1,0,0,1,2},I_{0,1,0,1,:}\}),\\ &(\{X_{0,1,0,0,1,2},I_{0,1,0,1,:}\},\{X_{0,1,0,0,1,2},I_{0,:,0,:,2}\})\big]\,.\end{aligned}$$

Intuitively, the messages commmunicate to the square constraint $I_{0,:,0,:,2}$*, that by the position constraint* $I_{0,1,0,1,:}$ *the variable* 3 *cannot be assigned at* $(0,1,0,1)$*.*

- *From* $X_{0,0,1,0,2}$ *(i.e. the* 3 *in the cell* $(0,0,1,0)$*) and the Sudoku rule that at the row* $(0,0)$ *exactly one number is assigned, we get*

$$\left(\bigoplus_{c0,c1\in[n]}^{(1)} X_{0,0,c0,c1,2}\right) \wedge X_{0,0,1,0,2} \models \neg X_{0,0,0,0,2} \wedge \neg X_{0,0,0,1,2}\,,$$

That is, that the number 3 *is neither in the cell* $(0,0,0,0)$ *nor in* $(0,0,0,1)$*. This entailment step is performed by five consecutive messages (see* $\chi^{(1,[5])}$ *in Figure 13) along the directions*

$$\begin{aligned}(e_0,e_1) \in \big[&(\{X_{0,0,1,0,2}\},\{X_{0,0,1,0,2},I_{0,0,:,:,2}\}),(\{X_{0,0,1,0,2},I_{0,0,:,:,2}\},\{X_{0,0,0,0,2},I_{0,0,:,:,2}\}),\\ &(\{X_{0,0,1,0,2},I_{0,0,:,:,2}\},\{X_{0,0,0,1,2},I_{0,0,:,:,2}\}),(\{X_{0,0,0,0,2},I_{0,0,:,:,2}\},\{X_{0,0,0,0,2},I_{0,:,0,:,2}\})\\ &(\{X_{0,0,0,1,2},I_{0,0,:,:,2}\},\{X_{0,0,0,1,2},I_{0,:,0,:,2}\})\big]\,.\end{aligned}$$

The messages communicate that based on the decomposition cores of the constraint to the number $i = 3$ *in the first row* $(r_0,r_1) = (0,0)$*, that the number* 3 *cannot be assigned at* $(0,0,0,0)$ *and* $(0,0,0,1)$*.*

We add these formulas to our knowledge base (justified by Thm. 5.14) and use the rule, that 3 *appears exactly once in the first square*

$$\left(\bigoplus_{r1,c1\in[n]}^{(1)} X_{0,r1,0,c1,2} \right) \wedge (\neg X_{0,1,0,1,2}) \wedge (\neg X_{0,0,0,0,2} \wedge \neg X_{0,0,0,1,2}) \models X_{0,1,0,0,2} \,.$$

That is, we conclude that the number 3 *must be in the cell* $(0,1,0,0)$*, which information is also included in the updated knowledge base for further reasoning steps. This last entailment step is performed by four consecutive messages (see* $\chi^{(2,[4])}$ *in Figure 13) along the directions*

$$\begin{aligned}(e_0,e_1) \in \big[&(\{X_{0,1,0,1,2}, I_{0,:,0,:,2}\}, \{X_{0,1,0,0,2}, I_{0,:,0,:,2}\}), (\{X_{0,0,0,1,2}, I_{0,:,0,:,2}\}, \{X_{0,1,0,0,2}, I_{0,:,0,:,2}\}),\\ &(\{X_{0,0,1,0,2}, I_{0,:,0,:,2}\}, \{X_{0,1,0,0,2}, I_{0,:,0,:,2}\}), (\{X_{0,1,0,0,2}, I_{0,:,0,:,2}\}, \{X_{0,1,0,0,2}\})\big]\end{aligned}$$

The first three messages communicate, that the 3 *is not possible the positions* $(0,1,0,1), (0,0,0,1)$ *and* $(0,0,1,0)$ *and the fourth message concludes that the* 3 *then has to be at position* $(0,1,0,0)$*.*

We now iteratively apply similar reasoning steps and store the entailed variables in E^{entailed}*, until we arrive at the right side of the above sketch.*

$$\mathcal{KB}^2 \wedge \left(\bigwedge_{(r_0,r_1,c_0,c_1,i)\in E^{\text{start}}} X_{r0,r1,c0,c1,i} \right) \models \left(\bigwedge_{(r_0,r_1,c_0,c_1,i)\in E^{\text{entailed}}} X_{r0,r1,c0,c1,i} \right) .$$

Since all Sudoku rules are satisfied in the final assignment and to each cell (r_0,r_1,c_0,c_1) *we found exactly one* $i \in [n^2]$ *such that* $(r_0,r_1,c_0,c_1,i) \in E^{\text{start}} \cup E^{\text{entailed}}$*, there is a unique solution of the puzzle and we conclude*

$$\begin{aligned}&\mathcal{KB}^2 \wedge \left(\bigwedge_{(r_0,r_1,c_0,c_1,i)\in E^{\text{start}}} X_{r0,r1,c0,c1,i} \right)\\ &= \left(\bigwedge_{(r_0,r_1,c_0,c_1,i)\in E^{\text{start}}} X_{r0,r1,c0,c1,i} \right) \wedge \left(\bigwedge_{(r_0,r_1,c_0,c_1,i)\in E^{\text{entailed}}} X_{r0,r1,c0,c1,i} \right) .\end{aligned}$$

6 Hybrid Logic Networks

Let us now exploit the common formulation of logical formulas and probabilistic models in CompActNets to define hybrid models that combine both aspects. We call CompActNets Hybrid Logic Networks in the special case of Boolean statistics t and elementary activations.

6.1 Parametrization

We first introduce Hybrid Logic Networks, which can be regarded as a unification of logical and probabilistic models.

Definition 6.1 (Hybrid Logic Network (HLN))**.** *Given a Boolean statistic* t*, we call any element of* $\Lambda^{t,\mathrm{EL}}$ *a Hybrid Logic Network. The extended canonical parameter set for* t *is the set*

$$\mathcal{P}_p := \{(A, y_A) \,:\, A \subset [p],\, y_A \in \bigtimes_{\ell\in A} [2]\} \times \mathbb{R}^p \,.$$

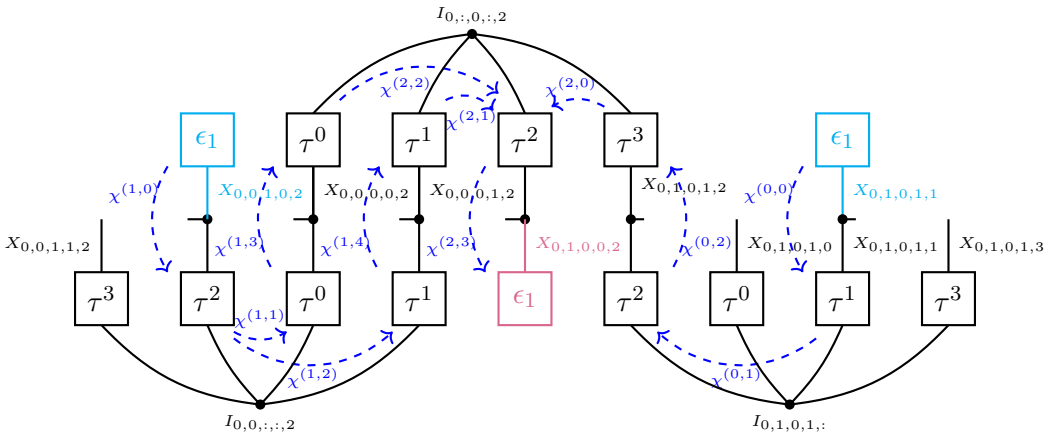


Figure 13: The tensor network decomposition of 3 out of $4 \cdot 2^2 = 64$ rules in the $2^2 \times 2^2$ Sudoku knowledge base (see Example 5.12), namely to the number 3 appearing once in the $(0,0)$-square (top), the number 3 appearing once in the $(0,0)$-row (bottom left) and a unique number appearing at the $(0,1,0,1)$-position (bottom right). The evidence of the number 3 already being assigned at the position $(0,0,1,0)$ is sketched by a basis vector ϵ_1 on the left side, and the number 2 assigned at position $(0,1,0,1)$ analogously on the right side. During Constraint Propagation Algorithm 3 on the hypergraph of Sudoku rules and evidence (see Example 5.16), this evidence is in three epochs of messages propagated to the constraints by partial entailment steps and imply that $X_{0,1,0,0,2}$ is true, i.e. that at the position $(0,1,0,0)$ the number 3 needs to be assigned. We depict the messages between the cores by dashed lines labeled by $\chi^{(0,[3])}, \chi^{(1,[5])}$ and $\chi^{(2,[4])}$ and provide further interpretation in Example 5.16.

For each Hybrid Logic Network $\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right]$*, we can associate a tuple* (A, y_A, θ) *consisting of a subset* $A \subset [p]$*, a tuple* $y_A \in \bigtimes_{\ell \in A}[2]$*, and* $\theta\,[L] \in \mathbb{R}^p$ *such that*

$$\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right] = \left\langle \beta^t\left[Y_{[p]}, X_{[d]}\right], \xi^{(A,y_A,\theta)}\left[Y_{[p]}\right]\right\rangle_{\left[X_{[d]}|\varnothing\right]}$$

where the activation core is

$$\xi^{(A,y_A,\theta)}\left[Y_{[p]}\right] = \left\langle \alpha^{\theta}\left[Y_{[p]}\right], \kappa^{(A,y_A)}\left[Y_{[p]}\right]\right\rangle_{\left[Y_{[p]}\right]}.$$

We notice that the parametrization by $\mathcal{P}_p$ is one-to-one for any non-vanishing elementary activation tensor up to a scalar factor. Given an arbitrary elementary activation tensor $\bigotimes_{\ell\in[p]} \xi^{\ell}\left[Y_\ell\right]$, we can always find a corresponding tuple in $\mathcal{P}_p$ by choosing[1]

$$A = \left\{\ell \,:\, \mathbb{I}_{\neq 0}\left(\xi^{\ell}\left[Y_\ell\right]\right) \neq \mathbb{I}\left[Y_\ell\right]\right\},$$

further for all $\ell \in A$

$$y_\ell = \begin{cases} 0 & \text{if} \quad \mathbb{I}_{\neq 0}\left(\xi^{\ell}\left[Y_\ell\right]\right) = \epsilon_0\left[Y_\ell\right] \\ 1 & \text{if} \quad \mathbb{I}_{\neq 0}\left(\xi^{\ell}\left[Y_\ell\right]\right) = \epsilon_1\left[Y_\ell\right] \end{cases}$$

and a parameter vector $\theta\,[L] \in \mathbb{R}^p$ defined for all $\ell \in [p]$ as

$$\theta\,[L = \ell] = \begin{cases} 0 & \text{if} \quad \ell \in A \\ \ln\left[\frac{\xi^{\ell}[Y_\ell=1]}{\xi^{\ell}[Y_\ell=0]}\right] & \text{if} \quad \ell \notin A\,. \end{cases}$$

[1] Here $\mathbb{I}_{\neq 0}(\cdot)$ is the indicator of non-zero entries acting coordinatewise and $\mathbb{I}\left[Y_\ell\right]$ is the vector $[1,1]^T$.

Then we have by construction that there is $\lambda > 0$ with

$$\bigotimes_{\ell \in [p]} \xi^{\ell} \left[Y_{\ell} \right] = \lambda \cdot \xi^{(A, y_A, \theta)} \left[Y_{[p]} \right] .$$

Let us demonstrate the utility of Hybrid Logic Networks with an example from accounting.

Example 6.2 (Hybrid Logic Network for a toy accounting model)**.** *Let us consider a system of three variables $A1$ Account 1 is booked, $A2$ Account 2 is booked, F a feature on an invoice. We respect two rules*

- *Exactly one account must be booked.*
- *If feature F is present on the invoice, the account $\mathrm{A1}$ is typically booked.*

We formalize this with the statistic

$$t = (X_{A1} \oplus X_{A2}, X_F \Rightarrow X_{A1}) .$$

While the first formula is a hard feature, the second is soft since prone to exceptions. We parameterize the first output of the statistic with the hard parameters by setting the set of indices to be initialized with hard logic $A = \{0\}$ and the corresponding initialization $y_0 = 1$ meaning, that the first output of the statistic has to be true for the input to have positive probability. Then "hard logic activation tensor", should be indifferent to the second part of the statistic, and only impose rules on the first part, leading to

$$\kappa^{(A, y_A)}[Y_0, Y_1] = \epsilon_{y_0} \left[Y_0 \right] \otimes \mathbb{I} \left[Y_1 \right] = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \end{bmatrix} .$$

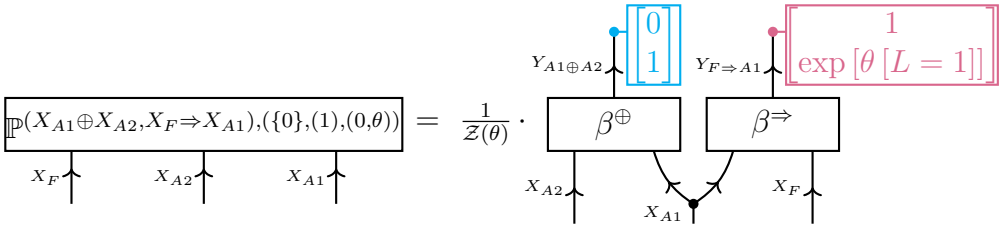
Since the first feature is hard, the ßoft logic activation tensorßhould be invariant under the first coordinate of the canonical parameter and we set $\theta \left[L = 0 \right] = 0$. We choose the soft parameters as $\theta[L] = [0, \theta[L = 1]]^{\intercal}$ to achieve

$$\alpha^{\theta}[Y_0, Y_1] = \alpha^{0,0} \left[Y_0 \right] \otimes \alpha^{1, \theta[L=1]} \left[Y_1 \right] = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ \exp \left[\theta \left[L = 1 \right] \right] \end{bmatrix} .$$

The activation tensor of the hybrid network then has the from

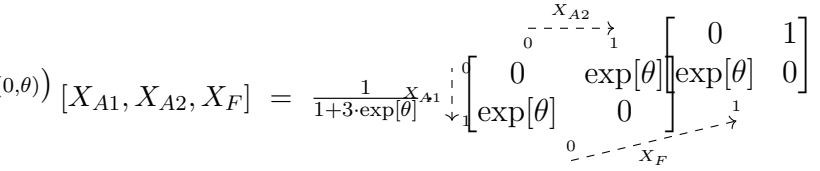
$$\xi^{(A, y_A, \theta)}[Y_0, Y_1] = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ \exp \left[\theta \left[L = 1 \right] \right] \end{bmatrix} .$$

We get a tensor network representation of the Hybrid Logic Network representing the toy accounting example, before normalization to a distribution



The resulting Hybrid Logic Network is a tensor $\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{A1}, X_{A2}, X_F\right]$ *of order* 3*. With* $Y_{F\Rightarrow A_1} = 1$ *for* $F = 0$ *and any* A_1 *it has the coordinates*

$$\mathbb{P}^{(X_{A1}\oplus X_{A2}, X_F\Rightarrow X_{A1}),\left(\{0\},(1),(0,\theta)\right)}\left[X_{A1}, X_{A2}, X_F\right] \;=\; \frac{1}{1+3\cdot\exp[\theta]}$$



6.2 Parameter estimation in Hybrid Logic Networks

Let us now briefly discuss how Hybrid Logic Networks can be trained on data based on likelihood maximization. Given a dataset $\left((x_0^j, \ldots, x_{d-1}^j) \,:\, j \in [m]\right)$ consisting of m independent and identically distributed samples from an unknown distribution, we want to find a Hybrid Logic Network $\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right]$ with a statistic $t = (f_1, \ldots, f_L)$ that maximizes the data likelihood

$$\mathcal{L}_D\left((A, y_A, \theta)\right) \coloneqq -\frac{1}{m}\sum_{j\in[m]} \ln\left[\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]} = x_{[d]}^j\right]\right] .$$

We can rewrite the loss using the empirical mean vector $\mu_D\left[L\right] \in \mathbb{R}^p$, which is defined for $\ell \in [p]$ as

$$\mu_D\left[L = \ell\right] = \frac{1}{m}\sum_{j\in[m]} f_\ell\left[X_{[d]} = x_{[d]}^j\right] ,$$

by

$$\mathcal{L}_D\left((A, y_A, \theta)\right) = \left\langle \mu_D\left[L\right], \theta\left[L\right]\right\rangle_{[\varnothing]} - \ln\left[\left\langle \xi^{(A,y_A,\theta)}\left[Y_{[p]}\right], \beta^t\left[Y_{[p]}, X_{[d]}\right]\right\rangle_{[\varnothing]}\right] .$$

Since (A, y_A) influences only the second term, the best hard parameters can be found by

$$A = \{\ell \,:\, \mu_D\left[L = \ell\right] \in \{0, 1\}\} \quad \text{and} \quad y_\ell = \mu_D\left[L = \ell\right] \quad \text{for} \quad \ell \in A\,.$$

We further optimize the coordinates $\ell \in [p]\backslash A$ of $\theta\left[L\right] \in \mathbb{R}^p$ alternately by the coordinate descent steps

$$\frac{\partial\mathcal{L}_D\left((A, y_A, \theta)\right)}{\partial\theta\left[L = \ell\right]} = 0 \Leftrightarrow \theta\left[L = \ell\right] = \ln\left[\frac{\mu\left[L = \ell\right]}{\left(1 - \mu\left[L = \ell\right]\right)} \cdot \frac{\tau\left[Y_\ell = 0\right]}{\tau\left[Y_\ell = 1\right]}\right] .$$

where

$$\tau\left[Y_\ell\right] = \left\langle \{\beta^{f_{\tilde\ell}} \,:\, \tilde\ell \in [p]\} \cup \{\alpha^{\tilde\ell,\theta} \,:\, \tilde\ell \in [p],\, \tilde\ell \neq \ell\} \cup \{\nu\}\right\rangle_{[Y_\ell]} .$$

Based on an interpretation of the coordinate descent steps as matching steps for the mean parameters or moments to f_ℓ, we call this method alternating moment matching for Hybrid Logic Networks and provide pseudocode for it it in Algorithm 4. We notice that, during the coordinate descent steps, computing the marginal probability of the variable Y_ℓ with respect to the current network parameters is required. This is the computational bottleneck of the algorithm and can be approached by various approximate inference methods, e.g., variational inference (see for example the CAMEL method [**?**]).

It can be shown that the algorithm converges if and only if there is a Hybrid Logic Network matching the empirical moments of the data. For more details we refer to [**?**, Chapter 9].

Algorithm 4 Alternating Moment Matching for Hybrid Logic Networks

Require: Mean parameter $\mu_D[L]$
Ensure: Parameters (A, y_A, θ) for the approximating HLN $\mathbb{P}^{(t,\theta,\nu)}$

Set

$$A = \left\{ \ell \,:\, \ell \in [p],\, \mu[L=\ell] \in \{0,1\} \right\}$$

and a tuple y_A with $y_\ell = \mu[L=\ell]$ for $\ell \in A$.
Set $\theta[L] = 0[L]$
while Convergence criterion is not met **do**
  for all $\ell \in [p] \backslash A$ **do**
    Compute

$$\tau[Y_\ell] = \left\langle \{\beta^{f_{\tilde{\ell}}} \,:\, \tilde{\ell} \in [p]\} \cup \{\alpha^{\tilde{\ell},\theta} \,:\, \tilde{\ell} \in [p],\, \tilde{\ell} \neq \ell\} \cup \{\nu\} \right\rangle_{[Y_\ell]}$$

    Set

$$\theta[L=\ell] = \ln\left[\frac{\mu[L=\ell]}{(1-\mu[L=\ell])} \cdot \frac{\tau[Y_\ell=0]}{\tau[Y_\ell=1]} \right]$$

  end for
end while
return $(A, y_A, \theta[L])$

Example 6.3 (Continuation of Example 6.2)**.** *Let us recall the statistic of Example 6.2 and consider a dataset of* $m = 20$ *states summarized in the frequency table:*

Frequency in Dataset	x_{A1}	x_{A2}	x_F
0	*0*	*0*	*0*
0	*0*	*0*	*1*
7	*0*	*1*	*0*
2	*0*	*1*	*1*
1	*1*	*0*	*0*
10	*1*	*0*	*1*
0	*1*	*1*	*0*
0	*1*	*1*	*1*

We then have for the satisfaction rates of $f_0 = X_{A1} \oplus X_{A2}$ *and* $f_1 = X_F \Rightarrow X_{A1}$

$$\mu_D[L=0] = \frac{20}{20} = 1 \quad \textit{and} \quad \mu_D[L=1] = \frac{7+1+10}{20} = 0.9\,.$$

Then Algorithm 4 yields with a reasonable convergence criterion choice (such as finite iterations or convergence of $\theta[L]$*)*

$$A = \{0\} \quad , \quad y_A = 1 \quad \textit{and} \quad \theta[L] = \begin{bmatrix} 0 \\ \ln\left[(\frac{0.9}{0.1}) \cdot (\frac{1}{3})\right] \end{bmatrix} = \begin{bmatrix} 0 \\ \ln[3] \end{bmatrix} \approx \begin{bmatrix} 0 \\ 1.098612 \end{bmatrix}\,.$$

To derive this, we notice that Algorithm 4 treats formula f_0 *as a hard constraint and assigns* $A = \{0\}$

and $y_A = 1$. *In the* While *loop we then have for the formula* f_1

$$\tau\left[Y_1\right] = \left\langle \epsilon_1\left[Y_0\right], \beta^{f_0}\left[Y_0, X_F, X_{A1}, X_{A2}\right], \beta^{f_1}\left[Y_1, X_F, X_{A1}, X_{A2}\right]\right\rangle_{[Y_1]} = \begin{bmatrix}1\\3\end{bmatrix}$$

since f_0 *has* 4 *models, of which* 3 *are also models of* f_1 *and* 1 *is instead a model of* $\neg f_1$. *Notice, that the tensor* $\tau\left[Y_1\right]$ *will not change in any further iteration of the* While *and the parameter* $\theta\left[L=1\right]$ *will therefore stay constant until the termination of the algorithm.*

6.3 Entailment by Hybrid Logic Networks

Let us now demonstrate a further use of our unified treatment of probabilistic and logical models by investigating a generalized concept of entailment. Entailment can be generalized to probabilistic models by deciding whether a propositional formula is always satisfied given a probabilistic model.

Theorem 6.4. *Let* $\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right]$ *be a Hybrid Logic Network and* $h\left[X_{[d]}\right]$ *a propositional formula. Then* $\mathbb{P}^{t,(A,y_A,\theta)}$ *probabilistically entails* h, *that is,*

$$\left\langle \mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right], h\left[X_{[d]}\right]\right\rangle_{[\varnothing]} = 1\,,$$

if and only if

$$f^{t,(A,y_A)} \models h\,,$$

where

$$f^{t,(A,y_A)}\left[X_{[d]}\right] = \left(\bigwedge_{\ell\in A\,:\,y_\ell=1} f_\ell\left[X_{[d]}\right]\right) \wedge \left(\bigwedge_{\ell\in A\,:\,y_\ell=0} \neg f_\ell\left[X_{[d]}\right]\right).$$

Proof. We have

$$\left\langle \mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right], h\left[X_{[d]}\right]\right\rangle_{[\varnothing]} = 1$$

if and only if

$$\left\langle \left(\mathbb{I}\left[X_{[d]}\right] - \mathbb{I}_{\neq 0}\left(\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right]\right)\right), h\left[X_{[d]}\right]\right\rangle_{[\varnothing]} = 0$$

. We notice that $f^{t,(A,y_A)}\left[X_{[d]}\right]$ is the indicator tensor for the support of $\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right]$. It therefore holds that

$$\begin{aligned}
&\left\langle \left(\mathbb{I}\left[X_{[d]}\right] - \mathbb{I}_{\neq 0}\left(\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right]\right)\right), h\left[X_{[d]}\right]\right\rangle_{[\varnothing]}\\
&\quad= \left\langle h\left[X_{[d]}\right]\right\rangle_{[\varnothing]} - \left\langle f^{t,(A,y_A)}\left[X_{[d]}\right], h\left[X_{[d]}\right]\right\rangle_{[\varnothing]}\\
&\quad= \left\langle h\left[X_{[d]}\right]\right\rangle_{[\varnothing]} - \left\langle f^{t,(A,y_A)}\left[X_{[d]}\right], \left(\mathbb{I}\left[X_{[d]}\right] - \neg h\left[X_{[d]}\right]\right)\right\rangle_{[\varnothing]}\\
&\quad= \left\langle f^{t,(A,y_A)}\left[X_{[d]}\right], h\left[X_{[d]}\right]\right\rangle_{[\varnothing]}.
\end{aligned}$$

By Def. 5.7 this vanishes if and only if $f^{t,(A,y_A)} \models h$. □

Example 6.5 (Continuation of Example 6.3)**.** *Let us consider again the Hybrid Logic Network* $\mathbb{P}^{(X_{A1}\oplus X_{A2}, X_F \Rightarrow X_{A1}),\left(\{0\}\right.}$ *from Example 6.3 and assume we want to decide the probabilistic entailment of the formula*

$$h\left[X_{A1}, X_{A2}, X_F\right] = \neg X_{A1} \vee \neg X_{A2} \vee \neg X_F\,,$$

which has all states but $(1,1,1)$ *as a model (and is therefore refered to as a maxterm). Using Thm. 6.4 we have that*

$$\left\langle \mathbb{P}^{(X_{A1}\oplus X_{A2}, X_F \Rightarrow X_{A1}),\left(\{0\},(1),(0,\ln[3])\right)}\left[X_{A1}, X_{A2}, X_F\right], h\left[X_{A1}, X_{A2}, X_F\right] \right\rangle_{[\varnothing]} = 1$$

if and only if $X_{A1} \oplus X_{A2} \models \neg X_{A1} \vee \neg X_{A2} \vee \neg X_F$*. By Def. 5.7 this entailment holds, since by the De-Morgan rule*

$$\begin{aligned}\left\langle X_{A1} \oplus X_{A2}, \neg\left(\neg X_{A1} \vee \neg X_{A2} \vee \neg X_F\right)\right\rangle_{[\varnothing]} &= \left\langle X_{A1} \oplus X_{A2}, X_{A1}, X_{A2}, X_F\right\rangle_{[\varnothing]} \\ &= \left\langle X_F\right\rangle_{[\varnothing]} \cdot \left\langle X_{A1} \oplus X_{A2}, X_{A1}, X_{A2}\right\rangle_{[\varnothing]} \\ &= 0\,.\end{aligned}$$

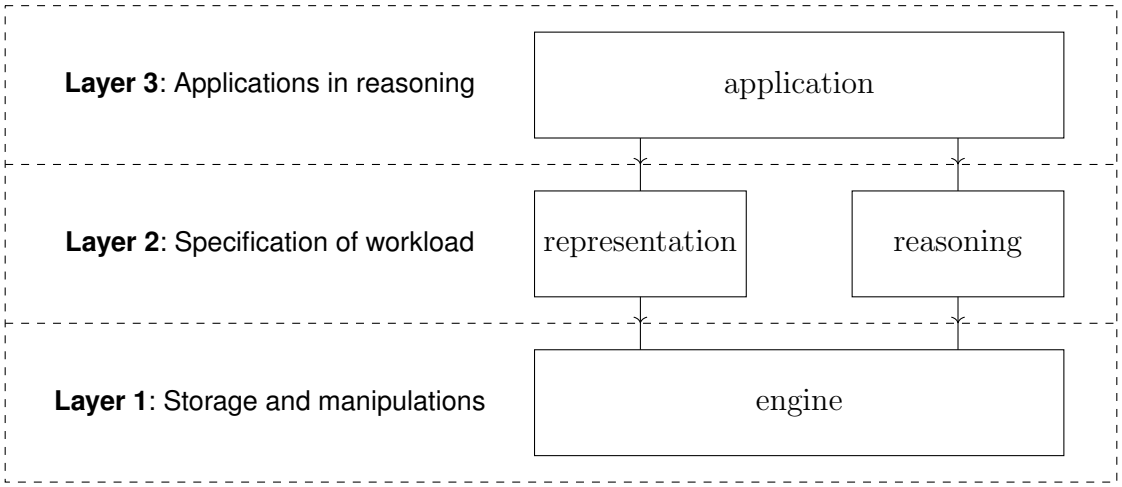
We thus conclude, that h *is probabilistically entailed by* $\mathbb{P}^{(X_{A1}\oplus X_{A2}, X_F \Rightarrow X_{A1}),\left(\{0\},(1),(0,\ln[3])\right)}$.

7 Implementation in the python library tnreason

The concepts presented in this paper have been implemented in the python library tnreason[2]. In this section, we explain the basic design and functionality of this library; Appendix B provides detailed implementations of the algorithms and examples in this work.

7.1 Architecture

The package consists of four subpackages and three layers of abstraction:



[2] tnreason is available in version 2.0.0 at `pypi.org/tnreason` and maintained at `github.com/tnreason/tnreason-py`.

In the subpackage tnreason.engine we implement tensors, tensor networks, contractions, and normalizations. In the subpackage tnreason.representation the basic tensor encoding schemes such as basis encodings are available. In the subpackage tnreason.reasoning we implement reasoning algorithms, such as generalizations of the message passing algorithms presented in Algorithm 1, Algorithm 3, and Algorithm 4. In the subpackage tnreason.application one can construct tensor network encodings of propositional formulas and datasets.

7.2 Basic usage

We demonstrate the basic usage of the tnreason package with the implementation of Example 5.2. We first install the package (e.g. by `pip install tnreason == 2.0.0`) and import it by

```
from tnreason import engine, application
```

Keeping Def. 2.1 in mind, the tensor instances in `shape` and `colors` arguments are `list` instances specifying the `int` dimension m_k and a `str` identifier for X_k. The formula in Example 5.2 is a sum of the one-hot encodings of its three models (see Example 5.3) and is created by

```
formula = engine.create_from_slice_iterator(shape=[2,2,2],
↪  colors=["X_0","X_1","X_2"],
↪  sliceIterator=[(1,{"X_0":0,"X_1":1,"X2":0}),
↪  (1,{"X_0":1,"X_1":0,"X2":0}), (1,{"X_0":1,"X_1":1,"X2":0})])
```

The slice iterator is an iterator over tuples `(val,posDict)`, which specifies elementary tensors to be summed. The `posDict` are `dict` instances, where the keys are the `str` tensor colors and the values are `int`. Each `posDict` collects leg vectors of the corresponding elementary tensor that are not trivial. These leg vectors are the basis vectors enumerated by the corresponding `int` value.

Single tensor coordinates can be retrieved by indexing with a `posDict`. We can, for example, check whether `{"X_0":0,"X_1":1,"X_2":0}` is a model:

```
assert formula[{"X_0":0,"X_1":1,"X_2"}] == 1
```

By default the tensor is created as a `engine.NumpyCore` instance, where coordinates are stored as instances of `numpy.array`. Further core types exploiting different sparsity principles can be chosen by the argument `coreType`, see [**?**, Appendix A].

Following Def. 2.3, tensor networks are implemented as tensor valued `dict` instances with `str` keys. For example a tensor network is created from the propositional syntax of the above formula (see Example 5.10):

```
fDecomp = application.create_cores_to_expressionsDict({"f0":
↪  ["and",["or","X_0","X_1"],["not","X_2"]]})
```

Here we apply a nested-list description of syntactic hypergraphs (see Def. **??**) with a specification of the logical connectives in the first position of the list (by `"and"`,`"or"`,`"not"` we refer to the connectives $\wedge, \vee, \neg$). Equivalently, we can exploit the $\wedge$ symmetry and create it by multiple formulas:

```
fDecomp = application.create_cores_to_expressionsDict({"f0":
↪  ["or","X_0","X_1"], "f1": ["not","X_2"]})
```

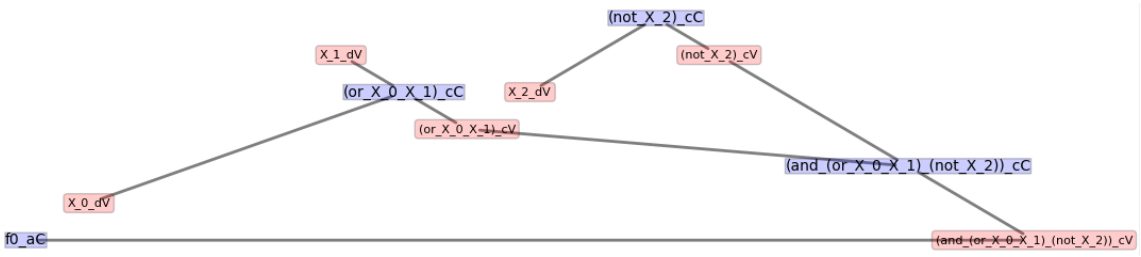


Figure 14: Factor graph highlighting a tensor network decomposition of the syntactic decomposition of the propositional formula of Example 5.10. Blue blocks highlight hyperedges carrying tensors and red blocks highlight variables. The tensor label suffices `"_cC"` and `"_aC"` indicate whether the tensor is part of the computation network or the activation network. The variable label suffices `"_dV"` and `"_cV"` indicate whether the variable is distributed or computed and therefore auxiliary. This graph has been generated with the method `tnreason.engine.draw_factor_graph` of tnreason.

A depiction of the underlying hypergraph as a factor graph, which highlights edges as blue blocks and nodes as red blocks, can be created with `engine.draw_factor_graph(fDecomp)` (see Figure 14). Single tensors can be obtained by contracting a tensor network while specifying the open variables (for an explanation of the suffixes, see Figure 14), for example:

```
contracted = engine.contract(fDecomp,
↪  openColors=["X_0_dV","X_1_dV","X_2_dV"])
```

By default the contractions are performed using `numpy.einsum` and further execution schemes can be selected with the argument `contractionMethod`, see [**?**, Appendix A].

8 Conclusion & outlook

This work developed a tensor network formalism to capture the main concepts of AI, which build the core of the probabilistic, neural and logical approaches. We introduced Computation-Activation Networks (CompActNets) as a generic architecture to represent classes of propositional knowledge bases, graphical models and more generic exponential families. Moreover, we demonstrated the representation and training of hybrid models combining logical and probabilistic aspects, showing that CompActNets are an interesting framework for Neuro-Symbolic AI.

We have shown that model inference such as the calculating marginal distributions and deciding entailment correspond with tensor network contractions. To efficiently perform these inferences, we presented message passing schemes, which have been shown to be exact in specific cases. In general, however, the efficient computation of contractions is not possible, since they are related to the NP-hardness of probabilistic inferences in graphical models (see [**?**]) and of logical reasoning (see [**?**]). In cases where exact inference is not feasible, the derivation of error bounds for approximate inference schemes on CompActNets is an interesting direction for future research.

Further approximation schemes to overcome this bottleneck are summarized under the umbrella of variational inference (see [**?**]), such as generic expectation-propagation methods or mean field methods. While these schemes are developed either for graphical models or more general exponential families, we plan to derive similar methods for more general CompActNets, such as Hybrid Logic Networks. Further frequently applied schemes are particle-based inference schemes such as Gibbs sampling.

The integration of symbolic and sub-symbolic methods is an active research area (see [**?**] for a systematic review). The CompActNets framework enables both the symbolic logical and probabilistic models, but enables also the representation of generic functions. CompActNets based on architectures combining both symbolically verbalizable and more generic neural parts are thus a promising candidate for Neuro-Symbolic AI.

The CompActNets framework offers an immediate practical application as a verifiable reasoning engine for AI agents in high-stakes domains such as regulatory compliance, clinical decision support or accounting. By leveraging the framework's inherent flexibility, Large Language Models [**?**] can be adapted to function as semantic translators that dynamically construct problem-specific tensor networks in the form of CompActNets from natural language descriptions, effectively treating the reasoning engine as an external tool. This approach mitigates the hallucination risks of probabilistic models by delegating complex logical execution to the exact linear algebra of the tensor network, ensuring that the inference process is both rigorous and reproducible. Consequently, this synergy enables the deployment of reliable AI systems where the intuitive power of the Large Language Model is grounded by the explainable, instance-adaptive topology of the CompActNets.

A Proof of the Factorization Theorems

Let us now provide proofs for the factorization theorems stated in Sect. 3. These proofs are classically known (see e.g. [**?**] for Hammersley-Clifford and [**?**] for Fisher-Neyman). We here provide them in our tensor networks notation and for hypergraphs for completeness.

A.1 Hammersley-Clifford

Different to the original statement (see [**?**]), we here proof the analogous statement for hypergraphs, where we have to demand the property of clique-capturing defined in Def. 3.9. We start with showing the following Lemmata to be exploited in the proof.

Lemma A.1. *Let $\tau\left[X_{\mathcal{V}}\right]$ be a positive tensor and $y_{\mathcal{V}}$ an arbitrary index. Then we have*

$$\tau\left[X_{\mathcal{V}}\right] = \left\langle \left(\langle \tau \rangle_{\left[X_{\mathcal{V}/\mathcal{W}}, X_{\mathcal{W}}=y_{\mathcal{W}}\right]} \right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}} : \mathcal{W} \subset \mathcal{U} \subset \mathcal{V} \right\rangle_{\left[X_{\mathcal{V}}\right]} ,$$

where the exponentiation is performed coordinatewise and positivity of τ ensures the well-definedness.

Proof. It suffices to show, that for an arbitrary index $x_{\mathcal{V}}$ be an arbitrary index we have

$$\tau\left[X_{\mathcal{V}} = x_{\mathcal{V}}\right] = \prod_{\mathcal{U}\subset\mathcal{V}} \prod_{\mathcal{W}\subset\mathcal{U}} \left(\langle \tau \rangle_{\left[X_{\mathcal{V}/\mathcal{W}}=x_{\mathcal{V}/\mathcal{W}}, X_{\mathcal{W}}=y_{\mathcal{W}}\right]} \right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}} .$$

We do this by applying a logarithm on the right hand side and grouping the terms by $\mathcal{W}$ as

$$\begin{aligned}
&\ln\left[\prod_{\mathcal{U}\subset\mathcal{V}}\prod_{\mathcal{W}\subset\mathcal{U}}\langle\tau\rangle_{\left[X_{\mathcal{V}/\mathcal{W}}=x_{\mathcal{V}/\mathcal{W}},X_{\mathcal{W}}=y_{\mathcal{W}}\right]}{}^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}}\right]\\
&=\sum_{\mathcal{W}\subset\mathcal{V}}\ln\left[\langle\tau\rangle_{\left[X_{\mathcal{V}/\mathcal{W}}=x_{\mathcal{V}/\mathcal{W}},X_{\mathcal{W}}=y_{\mathcal{W}}\right]}\right]\left(\sum_{\mathcal{U}\subset\mathcal{V}\,:\,\mathcal{W}\subset\mathcal{U}}(-1)^{|\mathcal{U}|-|\mathcal{W}|}\right)\\
&=\sum_{\mathcal{W}\subset\mathcal{V}}\ln\left[\langle\tau\rangle_{\left[X_{\mathcal{V}/\mathcal{W}}=x_{\mathcal{V}/\mathcal{W}},X_{\mathcal{W}}=y_{\mathcal{W}}\right]}\right]\left(\sum_{i\in[|\mathcal{V}|-|\mathcal{W}|]}(-1)^i\binom{|\mathcal{V}|-|\mathcal{W}|}{i}\right)
\end{aligned}$$

Now, by the generic binomial theorem we have that for $n\in\mathbb{N}, n\neq 0$

$$0=(1-1)^n=\sum_{i\in[n]}(-1)^i\binom{n}{i}\,.$$

Therefore, the summands for $\mathcal{W}\neq\mathcal{V}$ vanish and we have

$$\begin{aligned}
&\ln\left[\prod_{\mathcal{U}\subset\mathcal{V}}\prod_{\mathcal{W}\subset\mathcal{U}}\left(\langle\tau\rangle_{\left[X_{\mathcal{V}/\mathcal{W}}=x_{\mathcal{V}/\mathcal{W}},X_{\mathcal{W}}=y_{\mathcal{W}}\right]}\right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}}\right]\\
&=\ln\left[\tau\left[X_{\mathcal{V}}=x_{\mathcal{V}}\right]\right]\left(\sum_{i\in[0]}(-1)^i\binom{0}{i}\right)\\
&=\ln\left[\tau\left[X_{\mathcal{V}}=x_{\mathcal{V}}\right]\right]\,.
\end{aligned}$$

Applying the exponential function on both sides establishes the claim. □

Lemma A.2. *Let τ be a positive tensor, $\mathcal{U}\subset\mathcal{V}$ and arbitrary subset and $x_{\mathcal{U}}$ an arbitrary index. When there are $a,b\in\mathcal{U}$, such that*

$$\langle\tau\rangle_{\left[X_a,X_b|X_{\mathcal{V}/\{a,b\}}\right]}=\left\langle\langle\tau\rangle_{\left[X_a|X_{\mathcal{V}/\{a,b\}}\right]},\langle\tau\rangle_{\left[X_b|X_{\mathcal{V}/\{a,b\}}\right]}\right\rangle_{[X_{\mathcal{U}}]}$$

then

$$\prod_{\mathcal{W}\subset\mathcal{U}}\left(\langle\tau\rangle_{\left[X_{\mathcal{V}/\mathcal{W}}=x_{\mathcal{V}/\mathcal{W}},X_{\mathcal{W}}=y_{\mathcal{W}}\right]}\right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}}=1\,.$$

Proof. We abbreviate

$$Z_{\mathcal{W}}=\langle\tau\rangle_{\left[X_{\mathcal{V}/\mathcal{W}}=x_{\mathcal{V}/\mathcal{W}},X_{\mathcal{W}}=y_{\mathcal{W}}\right]}\,.$$

By reorganizing the sum over $\mathcal{W}\subset\mathcal{U}$ into $\mathcal{W}\subset\mathcal{U}/a\cup b$ we have

$$\prod_{\mathcal{W}\subset\mathcal{U}}\left(Z_{\mathcal{W}}\right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}}=\prod_{\mathcal{W}\subset\mathcal{U}/\{a,b\}}\left(\frac{Z_{\mathcal{W}}\cdot Z_{\mathcal{W}\cup\{a,b\}}}{Z_{\mathcal{W}\cup\{a\}}\cdot Z_{\mathcal{W}\cup\{b\}}}\right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}}\,. \tag{A.1}$$

From the independence assumption it follows that for any index x

$$\begin{aligned}
&\langle\tau\rangle_{\left[X_a=x_a|X_{\mathcal{V}/\mathcal{W}\cup\{a,b\}}=x_{\mathcal{V}/\mathcal{W}\cup\{a,b\}},X_{\mathcal{W}}=y_{\mathcal{W}},X_b=x_b\right]}\\
&\quad=\langle\tau\rangle_{\left[X_a=x_a|X_{\mathcal{V}/\mathcal{W}\cup\{a,b\}}=x_{\mathcal{V}/\mathcal{W}\cup\{a,b\}},X_{\mathcal{W}}=y_{\mathcal{W}}\right]}\\
&\quad=\langle\tau\rangle_{\left[X_a=x_a|X_{\mathcal{V}/\mathcal{W}\cup\{a,b\}}=x_{\mathcal{V}/\mathcal{W}\cup\{a,b\}},X_{\mathcal{W}}=y_{\mathcal{W}},X_b=y_b\right]}
\end{aligned}$$

Applying this in each squares bracket term of (A.1) we get

$$\begin{aligned}\frac{Z_{\mathcal{W}}}{Z_{\mathcal{W}\cup\{a\}}} &= \frac{\langle\tau\rangle_{\left[X_a=x_a|X_{\mathcal{V}/\mathcal{W}\cup\{a,b\}}=x_{\mathcal{V}/\mathcal{W}\cup\{a,b\}},X_{\mathcal{W}}=y_{\mathcal{W}},X_b=x_b\right]}}{\langle\tau\rangle_{\left[X_a=y_a|X_{\mathcal{V}/\mathcal{W}\cup\{a,b\}}=x_{\mathcal{V}/\mathcal{W}\cup\{a,b\}},X_{\mathcal{W}}=y_{\mathcal{W}},X_b=x_b\right]}} \\ &= \frac{\langle\tau\rangle_{\left[X_a=x_a|X_{\mathcal{V}/\mathcal{W}\cup\{a,b\}}=x_{\mathcal{V}/\mathcal{W}\cup\{a,b\}},X_{\mathcal{W}}=y_{\mathcal{W}},X_b=y_b\right]}}{\langle\tau\rangle_{\left[X_a=y_a|X_{\mathcal{V}/\mathcal{W}\cup\{a,b\}}=x_{\mathcal{V}/\mathcal{W}\cup\{a,b\}},X_{\mathcal{W}}=y_{\mathcal{W}},X_b=y_b\right]}} \\ &= \frac{Z_{\mathcal{W}\cup\{b\}}}{Z_{\mathcal{W}\cup\{a,b\}}}\,.\end{aligned}$$

Thus, each factor in (A.1) is trivial, which establishes the claim. □

We are finally ready to prove the Hammersley-Clifford Thm. 3.10 based on the Lemmata above.

Proof of Thm. 3.10. $ii) \Rightarrow i)$ By Lem. A.1 we have for any index $x_{\mathcal{V}}$

$$\mathbb{P}\left[X_{\mathcal{V}}=x_{\mathcal{V}}\right] = \prod_{\mathcal{U}\subset\mathcal{V}}\prod_{\mathcal{W}\subset\mathcal{U}}\left(\mathbb{P}\left[X_{\mathcal{W}}=x_{\mathcal{W}},X_{\mathcal{V}/\mathcal{W}}=y_{\mathcal{V}/\mathcal{W}}\right]\right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}}\,.$$

Using the assumption of Thm. 3.10 we find for any subset $\mathcal{U}\subset\mathcal{V}$, which is not contained in a hyperedge, $a,b\in\mathcal{U}$ such that X_a is independendent on X_b conditioned on $X_{\mathcal{U}/\{a,b\}}$. If no such nodes $a,b\in\mathcal{U}$ exists, $\mathcal{U}$ would be contained in a hyperedge, since the hypergraph is assumed to be clique-capturing. By Lem. A.2 we then have

$$\prod_{\mathcal{W}\subset\mathcal{U}}\left(\mathbb{P}\left[X_{\mathcal{W}}=x_{\mathcal{W}},X_{\mathcal{V}/\mathcal{W}}=y_{\mathcal{V}/\mathcal{W}}\right]\right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}} = 1\,.$$

We label by a function

$$\alpha : \{\mathcal{U} : \exists e\in\mathcal{E} : \mathcal{U}\subset e\} \to \mathcal{E}$$

the remaining node subsets by a hyperedge containing the subset. We build the tensor

$$\tau^e\left[X_e\right] = \prod_{\mathcal{U}\,:\,\alpha(\mathcal{U})=e}\prod_{\mathcal{W}\subset\mathcal{U}}\left(\mathbb{P}\left[X_{\mathcal{W}}=x_{\mathcal{W}},X_{\mathcal{V}/\mathcal{W}}=y_{\mathcal{V}/\mathcal{W}}\right]\right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}}\,.$$

and get, that

$$\begin{aligned}\mathbb{P}\left[X_{\mathcal{V}}\right] &= \left\langle\left\{\tau^e\left[X_e\right]\,:\,e\in\mathcal{E}\right\}\right\rangle_{[X_{\mathcal{V}}]} \\ &= \left\langle\left\{\tau^e\left[X_e\right]\,:\,e\in\mathcal{E}\right\}\right\rangle_{[X_v|\varnothing]}\,.\end{aligned}$$

We have thus constructed a Markov Network with trivial partition function, which contraction coincides with the probability distribution.

$i) \Rightarrow ii)$: Let us now show the converse statement and assume that there is a Markov Network representing the distribution $\mathbb{P}\left[X_{\mathcal{V}}\right]$, and let us choose subsets $A,B,C\subset\mathcal{V}$ such that C separates A from B. Let us denote by $\mathcal{V}_0$ the nodes with paths to A, which do not contain a node in C, and by $\mathcal{V}_1$ the nodes with paths to B, which do not contain a node in C. Further, we denote by $\mathcal{E}_0$ the hyperedges which contain a node in $\mathcal{V}_0$ and by $\mathcal{E}_1$ the hyperedges which contain a node in $\mathcal{V}_1$. By

assumption of separability, both sets $\mathcal{E}_0$ and $\mathcal{E}_1$ are disjoint and no node in A is in a hyperedge in $\mathcal{E}_1$, respectively no node in B is in a hyperedge in $\mathcal{E}_0$, . We then have

$$\begin{aligned}\left\langle \left\{ \tau^e \left[X_e \right] \,:\, e \in \mathcal{E} \right\} \right\rangle_{[X_A, X_B | X_C = x_C]} &= \left\langle \left\{ \tau^e \left[X_e \right] \,:\, e \in \mathcal{E} \right\} \cup \left\{ \epsilon_{x_C} \right\} \right\rangle_{[X_A, X_B | \varnothing]} \\ &= \left\langle \left\{ \tau^e \,:\, e \in \mathcal{E}_0 \right\} \cup \left\{ \epsilon_{x_C} \right\} \right\rangle_{[X_A | \varnothing]} \\ &\quad \otimes \left\langle \left\{ \tau^e \,:\, e \in \mathcal{E}_1 \right\} \cup \left\{ \epsilon_{x_C} \right\} \right\rangle_{[X_B | \varnothing]} .\end{aligned}$$

By Def. 3.5, this is the independence of X_A and X_B conditioned on X_C. □

A.2 Fisher-Neyman

Since sufficient statistics are sometimes introduced based on the data processing inequality (see e.g. [**?**]), we also show that also that definition is equivalent to the factorization of the family.

Theorem A.3 (Fisher-Neyman factorization theorem)**.** *Let $\mathbb{P}$ be a joint distribution of variables Z, X with values $\mathrm{val}\left(Z\right), \mathrm{val}\left(X\right)$ and let $t\left(x\right)$ be a statistic. The following are equivalent:*

i) The Data Processing Inequality holds straight, i.e.

$$I\left(Z;X\right) = I\left(Z;Y_t\right)$$

ii) $Z \to Y_t \to X$ is a Markov Chain, i.e.

$$\left(Z \perp X\right) \mid Y_t$$

iii) There are tensors $\xi\left[Y_t, Z\right]$ and $\nu\left[X\right]$ such that

$$\mathbb{P}\left[Z = z, X = x\right] = \xi\left[Y_t = t\left(x\right), Z = z\right] \cdot \nu\left[X = x\right] .$$

Proof. $i) \Leftrightarrow ii)$: We have always

$$I\left(Z;X\right) = I\left(Z;\left(X,Y_t\right)\right) = I\left(Z;Y_t\right) + I\left(Z;X|Y_t\right)$$

and thus if and only if $i)$ holds

$$I\left(Z;X|Y_t\right) = 0\,.$$

Using the KL-divergence characterization of the mutual information, this is equal to

$$\mathbb{P}\big[Z, X\big|Y_t\big] = \left\langle \mathbb{P}\big[Z\big|Y_t\big], \mathbb{P}\big[X\big|Y_t\big] \right\rangle_{[Z,X,Y_t]} .$$

This is equivalent to the conditional independence statement $ii)$.

$ii) \Rightarrow iii)$: Let us assume $ii)$. For almost all $z \in \mathrm{val}\left(Z\right)$ and $x \in \mathrm{val}\left(X\right)$ we then have

$$\begin{aligned}\mathbb{P}\big[Z = z\big|X = x\big] &= \mathbb{P}\big[Z = z\big|X = x, Y_t = t\left(x\right)\big] \\ &= \mathbb{P}\big[Z = z\big|Y_t = t\left(x\right)\big]\end{aligned}$$

Here we used that Y_t has a deterministic dependence on X. There is thus a tensor ξ such that for all $z \in \mathrm{val}(Z)$ and $x \in \mathrm{val}(X)$

$$\xi[Y_t = t(x), Z = z] = \mathbb{P}\big[Z = z \big| X = x\big].$$

We further define a tensor $\nu[X] = \mathbb{P}[X]$ and get

$$\begin{aligned}\mathbb{P}[Z = z, X = x] &= \mathbb{P}[X = x] \cdot \mathbb{P}\big[Z = z \big| X = x\big] \\ &= \xi[Y_t = t(x), Z = z] \cdot \nu[X = x].\end{aligned}$$

$iii) \Rightarrow ii)$: When assuming $iii)$ we have for all $(x, z) \in \mathrm{val}(Z) \times \mathrm{val}(X)$

$$\begin{aligned}\mathbb{P}\big[Z = z \big| X = x\big] &= \big\langle \xi[Y_t, Z], \beta^t[Y_t, X], \nu[X] \big\rangle_{[Z=z|X=x]} \\ &= \big\langle \xi[Y_t, Z], \beta^t[Y_t, X = x], \nu[X = x] \big\rangle_{[Z=z|\varnothing]} \\ &= \big\langle \xi[Y_t, Z], \epsilon_{t(x)}[Y_t] \big\rangle_{[Z=z|\varnothing]} \\ &= \mathbb{P}\big[Z = z \big| Y_t = t(x)\big].\end{aligned}$$

We further have at almost all $y_t \in \mathrm{val}(Y_t)$, $z \in \mathrm{val}(Z)$ and $x \in \mathrm{val}(X)$ that $y_t = t(x)$ and

$$\mathbb{P}\big[Z = z \big| X = x, Y_t = y_t\big] = \mathbb{P}\big[Z = z \big| X = x\big]$$

and with the above at thus at almost all such pairs

$$\mathbb{P}\big[Z = z \big| X = x, Y_t = y_t\big] = \mathbb{P}\big[Z = z \big| Y_t = y_t\big].$$

This is equivalent to $ii)$. □

Thm. 3.15 follows from Thm. A.3 by the equivalence of $ii)$ and $iii)$.

B Implementation of the algorithms and examples

The implementations of the algorithms and concepts are available at and implemented with tnreason in the version 2.0.0.

B.1 Algorithm 1, 2 and 3 (Tree, Directed Belief and Constraint Propagation)

The three message passing algorithms are implemented as functions as one class `ContractionPropagation`, since they share common structure.

```
from tnreason.engine import contract
from tnreason.engine import create_from_slice_iterator as create


class ContractionPropagation:
    """
    Summary Class for the Tree Belief, Directed Belief and Constraint Propagation
    ↪  Algorithms
    """
    def __init__(self, cores):
        self.cores = cores
        self.directions = {send: [receive for receive in cores if
```

```
                                set(cores[send].colors) & set(
                                    cores[receive].colors) and receive != send]
                            for send in cores}
        self.messages = {receive: {} for receive in self.cores}

    def trivial_message(self, send, receive):
        """
        Prepares trivial message from the send to the receive hyperedge
        """
        commonColors = list(set(self.cores[send].colors) &
        ↪  set(self.cores[receive].colors))
        shape = [self.cores[send].shape[i]
                 for i, c in enumerate(self.cores[send].colors) if c in commonColors]
        return create(shape=shape, colors=commonColors, sliceIterator=[(1, {})])

    def calculate_message(self, send, receive):
        """
        Contract received messages with hypercore to send new
        """
        return contract({send: self.cores[send],
                         **{preSend: self.messages[send][preSend] for preSend in
                         ↪  self.messages[send]
                            if preSend != receive}},
                        openColors=list(set(self.cores[send].colors) &
                        ↪  set(self.cores[receive].colors)))

    def tree_propagation(self):
        """
        Implementation of the Directed Belief Propagation Algorithm:
        Messages are sent starting at the leafs and scheduled if all others received at a
        ↪  core
        """
        schedule = [(send, receive) for send in self.cores for receive in
                    self.directions[send] if len(self.directions[send]) == 1]
        while len(schedule) > 0:
            send, receive = schedule.pop()
            self.messages[receive][send] = self.calculate_message(send, receive)
            for next in self.directions[receive]:
                if (not receive in self.messages[next] and
                        all([(otherSendKey in self.messages[receive] or otherSendKey ==
                        ↪  next or
                              receive not in self.directions[otherSendKey]) for
                             otherSendKey in self.directions])):
                    schedule.append((receive, next))

    def directed_propagation(self, edgeDirections):
        """
        Implementation of the Directed Belief Propagation Algorithm:
        Messages are sent in direction of the hypergraph
        """
        filteredDirections = {
            send: [
                receive for receive in self.directions[send]
                if (common := set(self.cores[send].colors) &
                ↪  set(self.cores[receive].colors))
                   and common.issubset(set(edgeDirections[send][1]))
                   and common.issubset(set(edgeDirections[receive][0]))
            ]
            for send in self.directions
        }

        schedule = [(send, receive) for send in filteredDirections
                    for receive in filteredDirections[send] if
                    ↪  len(edgeDirections[send][0]) == 0]

        while len(schedule) > 0:
            send, receive = schedule.pop()
            self.messages[receive][send] = self.calculate_message(send, receive)
            for x in set(edgeDirections[send][1]) & set(edgeDirections[receive][0]):
                edgeDirections[receive][0].remove(x)
            if len(edgeDirections[receive][0]) == 0:
                schedule = schedule + [(receive, next) for next in
                ↪  filteredDirections[receive]
                                       if (receive, next) not in schedule]

    def constraint_propagation(self, startSendKeys):
        """
        Implementation of the Constraint Propagation Algorithm:
        Messages are resent, when the support of a received message has changed
```

```
        """
        schedule = [(send, receive) for send in startSendKeys for receive in
                    self.directions[send]]
        while len(schedule) > 0:
            send, receive = schedule.pop()
            message = (self.messages[receive][send].clone() if send in
            ↪  self.messages[receive]
                       else self.trivial_message(send, receive))
            cont = self.calculate_message(send, receive)

            messageChanged = False
            for val, pos in message:
                if message[pos] != 0 and cont[pos] == 0:
                    message[pos] = - message[pos]
                    messageChanged = True
            self.messages[receive][send] = message

            for next in self.directions[receive]:
                if messageChanged and next != receive and (receive, next) not in
                ↪  schedule:
                    schedule.append((receive, next))
```

B.1.1 Example 4.4 and 4.6 (Integer Summation in m-adic Representation)

Following the decomposition of summations in m-adic into local summations, the function `get_sum_tn` produces a corresponding tensor network of basis encodings. We test by coordinate retrieval operations, whether the summation is performed correctly.

```
from tnreason import engine
import math

from copy import deepcopy


def get_sum_tn(m, d):
    return {"b_0": engine.create_from_slice_iterator(
        shape=[m, 2, m, m],
        colors=[f"Y_{0}", f"Z_{0}", f"X_{0}", f"TX_{0}"],
        sliceIterator=[(1, {f"Y_{0}": (x + tx) % m, f"Z_{0}": math.floor((x + tx) / m),
                            f"X_{0}": x, f"TX_{0}": tx}) for x in range(m) for tx in
                            ↪  range(m)]),
        **{f"middleBlock{k}": engine.create_from_slice_iterator(
            shape=[m, 2, m, m, 2],
            colors=[f"Y_{k}", f"Z_{k}", f"X_{k}", f"TX_{k}", f"Z_{k - 1}"],
            sliceIterator=[
                (1, {f"Y_{k}": (x + tx + z0) % m,
                     f"Z_{k}": math.floor((x + tx + z0) / m),
                     f"X_{k}": x, f"TX_{k}": tx, f"fZ_{k - 1}": z0}) for x
                in range(m) for tx in range(m) for z0 in range(2)]
        ) for k in range(1, d - 1)},
        **{f"b_{d - 1}": engine.create_from_slice_iterator(
            shape=[m, 2, m, m, 2],
            colors=[f"Y_{d - 1}", f"Y_{d}", f"X_{d - 1}", f"TX_{d - 1}", f"Z_{d - 2}"],
            sliceIterator=[
                (1, {f"Y_{d - 1}": (x + tx + z0) % m, f"Y_{d}": math.floor((x + tx + z0)
                ↪  / m),
                     f"X_{d - 1}": x, f"TX_{d - 1}": tx, f"Z_{d - 2}": z0}) for x
                in range(m) for tx in range(m) for z0 in range(2)]
        )}}


def encode_digits(num0, num1, m):
    return {**{f"X_{len(num0) - 1 - i}_eC": engine.create_from_slice_iterator(shape=[m],
    ↪  colors=[
        f"X_{len(num0) - 1 - i}"], sliceIterator=[(1, {f"X_{len(num0) - 1 - i}":
        ↪  int(digit)})]) for
            i, digit in enumerate(num0)},
        **{f"TX_{len(num1) - 1 - i}_eC": engine.create_from_slice_iterator(shape=[m],
        ↪  colors=[
            f"TX_{len(num1) - 1 - i}"], sliceIterator=[
            (1, {f"TX_{len(num0) - 1 - i}": int(digit)})]) for i, digit in
           enumerate(num1)}}
```

```
assert 1 == encode_digits("0001", "0000", 10)["X_0_eC"][{"X_0": 1}]
assert 0 == encode_digits("0001", "0000", 10)["X_0_eC"][{"X_0": 0}]

## Example: 08+12=020 in basis 10
m = 10
catorder = 2
assert 1 == int(engine.contract(coreDict={**get_sum_tn(m, catorder),
↪  **encode_digits("08", "12", m)},
                                openColors=[f"Y_{k}" for k in range(catorder + 1)])[
                        {"Y_2": 0, "Y_1": 2, "Y_0": 0}])
assert 1 == int(engine.contract(coreDict={**get_sum_tn(m, catorder),
↪  **encode_digits("00", "00", m)},
                                openColors=[])[:])
## Example: 10+11=101 in basis 2
m = 2
catorder = 2
assert 1 == int(engine.contract(coreDict={**get_sum_tn(m, catorder),
↪  **encode_digits("10", "11", m)},
                                openColors=[f"Y_{k}" for k in range(catorder + 1)])[
                        {"Y_2": 1, "Y_1": 0, "Y_0": 1}])
assert 1 == int(engine.contract(coreDict={**get_sum_tn(m, catorder),
↪  **encode_digits("10", "11", m)},
                                openColors=[])[:])

from demonstrations.comp_act_nets.algorithms import propagation as cp

edgeDirections = {
    **{f"X_{i}_eC": [[], [f"X_{i}"]] for i in range(catorder)},
    **{f"TX_{i}_eC": [[], [f"TX_{i}"]] for i in range(catorder)},
    "b_0": [["X_0", "TX_0"], ["Y_0", "Z_0"]],
    **{f"b__{i}": [[f"X_{i}", f"TX_{i}", f"Z_{i - 1}"], [f"Y_{i}", f"Z_{i}"]]
       for i in range(1, catorder - 1)},
    f"b_{catorder - 1}": [[f"X_{catorder - 1}", f"TX_{catorder - 1}", f"Z_{catorder -
    ↪  2}"],
                          [f"Y_{catorder - 1}", f"Y_{catorder}"]],
}

propagator = cp.ContractionPropagation({**get_sum_tn(m, catorder), **encode_digits("01",
↪  "01", m)})
propagator.directed_propagation(edgeDirections=deepcopy(edgeDirections))

## Check whether the message arrived at b_1 states that the carry bit is 1
assert propagator.messages["b_1"]["b_0"][{"Z_0": 0}] == 0
assert propagator.messages["b_1"]["b_0"][{"Z_0": 1}] == 1

propagator = cp.ContractionPropagation({**get_sum_tn(m, catorder),
                                        **encode_digits("10", "10", m)})
propagator.directed_propagation(edgeDirections=deepcopy(edgeDirections))

## Check whether the message arrived at b_1 states that the carry bit is 1
assert propagator.messages["b_1"]["b_0"][{"Z_0": 0}] == 1
assert propagator.messages["b_1"]["b_0"][{"Z_0": 1}] == 0
```

B.1.2 Example 3.12 and 3.23 (Student Markov Network)

We here implement the Markov Network on the hypergraph of Example 3.12, with tensors having independent random coordinates drawn from the uniform distribution on $[0, 1]$. We test in a final `assert` statement, whether the messages resulting from Algorithm 1 in a tree implementation contract to the marginal distribution, which we directly compute for comparison.

```
from tnreason.engine import create_random_core, contract

studentTensorNetwork = {
    "t0": create_random_core(name="t0", colors=["G", "D", "I"], shape=[6, 3, 2]),
    "t1": create_random_core(name="t1", colors=["L", "G"], shape=[2, 6]),
    "t2": create_random_core(name="t2", colors=["I", "S"], shape=[2, 10]),
}

## Execute the contraction propagation algorithm in the tree-based implementation

from demonstrations.comp_act_nets.algorithms import propagation as cp

propagator = cp.ContractionPropagation(studentTensorNetwork)
```

```
propagator.tree_propagation()

## Test on the marginals of the variables "L","G" (core "t1")

testContraction = contract(studentTensorNetwork, openColors=["L", "G"])
propContraction = contract({"mes_t0_t1": propagator.messages["t1"]["t0"],
                                    "t1": studentTensorNetwork["t1"]}, openColors=["L",
                                    ↪  "G"])

tolerance = 1e-6
for posDict in [{"L": 0, "G": 1}, {"L": 1, "G": 5}]:
    assert abs(testContraction[posDict] - propContraction[posDict]) < tolerance
```

B.1.3 Example 5.8, 5.12 and 5.16 (Sudoku Game)

We implement the $n^2 \times n^2$ Sudoku with the start assignment given in Example 5.8 and apply the Constraint Propagation Algorithm 3 to deduce the full assignment. We then test whether the correct board assignment (given in Example 5.16) has been found.

```
from tnreason.engine import contract
from tnreason.engine import create_from_slice_iterator as create
import numpy as np


def create_sudoku_rule_tensor_network(n):
    """
    Creates a tensor network of n^2 \tau^k matrices to each Sudoku constraint
    """
    rulesSpecDict = {
        ## Column Constraints
        **{f"I_:_:_{c0}_{c1}_{i}": [f"X_{r0}_{r1}_{c0}_{c1}_{i}" for r0 in range(n) for
        ↪  r1 in
                                    range(n)] for c0 in range(n) for c1 in range(n)
           for i in range(n ** 2)},
        ## Row Constraints
        **{f"I_{r0}_{r1}_:_:_{i}": [f"X_{r0}_{r1}_{c0}_{c1}_{i}" for c0 in range(n) for
        ↪  c1 in
                                    range(n)] for r0 in range(n) for r1 in range(n)
           for i in range(n ** 2)},
        ## Squares Constraints
        **{f"I_{r0}_:_{c0}_:_{i}": [f"X_{r0}_{r1}_{c0}_{c1}_{i}" for r1 in range(n) for
        ↪  c1 in
                                    range(n)] for r0 in range(n) for c0 in range(n)
           for i in range(n ** 2)},
        ## Position Constraints
        **{f"I_{r0}_{r1}_{c0}_{c1}_:": [f"X_{r0}_{r1}_{c0}_{c1}_{i}" for i in range(n **
        ↪  2)]
           for r0 in range(n) for r1 in range(n) for c0 in range(n) for c1 in range(n)}
    }
    cores = {}
    for decomKey in rulesSpecDict:
        cores.update({
            decomKey + "_" + atomVar: create(
                shape=[2, len(rulesSpecDict[decomKey])],
                colors=[atomVar, decomKey],
                sliceIterator=[(1, {atomVar: 0}),
                               (-1, {atomVar: 0, decomKey: i}),
                               (1, {atomVar: 1, decomKey: i})])
            for i, atomVar in enumerate(rulesSpecDict[decomKey])
        })
    return cores


def encode_trivial_extended_evidence(E, n):
    """
    Prepares e_1 basis vectors for known variables and trivial vectors for others
    """
    return {**{f"{r0}_{r1}_{c0}_{c1}_{i}_eC":
                   create(shape=[2], colors=[f"X_{r0}_{r1}_{c0}_{c1}_{i}"],
                          sliceIterator=[(1, {f"X_{r0}_{r1}_{c0}_{c1}_{i}": 1})])
               for r0, r1, c0, c1, i in E},
            **{f"{r0}_{r1}_{c0}_{c1}_{i}_eC":
                   create(shape=[2], colors=[f"X_{r0}_{r1}_{c0}_{c1}_{i}"],
                          sliceIterator=[(1, {})])
```

```
                for r0 in range(n) for r1 in range(n) for c0 in range(n)
                for c1 in range(n) for i in range(n ** 2) if (r0, r1, c0, c1, i) not in
                ↪  E}}


def extract_resulting_evidence(propagator, n):
    """
    Returns the evidence given a ContractionPropagation instance
    """
    return [(r0, r1, c0, c1, i) for r0 in range(n) for r1 in range(n)
            for c0 in range(n) for c1 in range(n) for i in range(n ** 2)
            if contract({
            "eC": propagator.cores[f"{r0}_{r1}_{c0}_{c1}_{i}_eC"],
            **propagator.messages[f"{r0}_{r1}_{c0}_{c1}_{i}_eC"]},
            openColors=[f"X_{r0}_{r1}_{c0}_{c1}_{i}"])[{f"X_{r0}_{r1}_{c0}_{c1}_{i}": 0}]
            ↪  == 0]


def tuples_to_array(evidence, n=2):
    """
    Arranges the variables in an array
    """
    array = np.zeros(shape=(n ** 2, n ** 2))
    for (r0, r1, c0, c1, i) in evidence:
        array[r0 * n + r1, c0 * n + c1] = i + 1
    return array


from demonstrations.comp_act_nets.algorithms import propagation as cp

n = 2
evidence = [(0, 0, 0, 0, 0), (0, 0, 1, 0, 2), (0, 0, 1, 1, 1),
            (0, 1, 0, 1, 1), (1, 0, 1, 0, 3), (1, 1, 0, 0, 3),
            (1, 1, 0, 1, 2)]
propagator = cp.ContractionPropagation(
    cores={**create_sudoku_rule_tensor_network(n=n),
           **encode_trivial_extended_evidence(evidence, n=n)})
propagator.constraint_propagation([f"{r0}_{r1}_{c0}_{c1}_{i}_eC" for (r0, r1, c0, c1, i)
↪  in evidence])
solutionArray = tuples_to_array(extract_resulting_evidence(propagator, n=2))
assert np.all(solutionArray == np.array([[1, 4, 3, 2], [3, 2, 1, 4], [2, 1, 4, 3], [4, 3,
↪  2, 1]]))
```

B.2 Algorithm 4 (Alternating Moment Matching)

We implement the Alternating Moment Matching algorithm, which estimates the parameters of Hybrid Logic Networks, as a class `MomentMatcher`.

```
from tnreason.engine import contract
from tnreason.engine import create_from_slice_iterator as create
import math


class MomentMatcher:
    def __init__(self, cores, hCols, satRates):
        self.cores = cores
        self.hCols = hCols
        self.satRates = satRates

        self.hardParams = {hCol: int(satRates[hCol]) for hCol in self.hCols if
                           satRates[hCol] in [0, 1]}
        self.softParams = {hCol: 0 for hCol in self.hCols if hCol not in self.hardParams}

    def update_canonical_parameter(self, uCol):
        con = contract({**self.cores,
                        **{hCol: create(shape=[2], colors=[hCol],
                                        sliceIterator=[(1, {hCol:
                                        ↪  self.hardParams[hCol]})])
                           for hCol in self.hardParams},
                        **{hCol: create(shape=[2], colors=[hCol],
                                        sliceIterator=[(1, {hCol: 0}),
                                                       (math.exp(self.softParams[hCol]),
                                                       ↪  {hCol: 1})])
                           for hCol in self.softParams if hCol != uCol}
```

```
                        }, openColors=[uCol])
        self.softParams[uCol] = math.log(self.satRates[uCol] * con[{uCol: 0}] / (
                (1 - self.satRates[uCol]) * con[{uCol: 1}]))

    def alternate(self, iterations=1):
        for _ in range(iterations):
            for hCol in self.softParams:
                self.update_canonical_parameter(hCol)
```

Let us now show the usage of the algorithm on the toy accounting model presented in Example 6.2. To this end we train the parameters based on a the dataset described in Example 6.3, and assert that the learned parameters are close to the true parameters. Note that a single iterations suffices for convergence in this simple example.

```
import pandas as pd

samples = pd.DataFrame({
    "A1": [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    "A2": [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    "F": [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
})

formulaExpressions = {
    "(xor_A1_A2)": ["xor", "A1", "A2", 0],
    "(imp_F_A1)": ["imp", "F", "A1", 0],
}

from tnreason.engine import normalize
from tnreason.application import data_to_cores as dtc
from tnreason.application import create_cores_to_expressionsDict as cte
from demonstrations.comp_act_nets.algorithms import moment_matching as mm

satRates = {
    formulaKey + "_cV":
        normalize({**dtc.create_data_cores(samples),
                   **cte({formulaKey: formulaExpressions[formulaKey]})},
                  outColors=[formulaKey + "_cV"], inColors=[])[{formulaKey + "_cV": 1}]
    for formulaKey in formulaExpressions
}

matcher = mm.MomentMatcher(cores=cte(formulaExpressions),
                           satRates=satRates, hCols=["(xor_A1_A2)_cV", "(imp_F_A1)_cV"])
matcher.alternate(iterations=1)
assert abs(matcher.softParams["(imp_F_A1)_cV"] - 1.09861228866811) < 1e-8
```