A Tensor-Network formalism for Neuro-Symbolic AI

Alex Goessmann ALEX.GOESSMANN@WIAS-BERLIN.DE
Janina Schütte JANINA.SCHUETTE@WIAS-BERLIN.DE
Maximilian Fröhlich MAXIMILIAN.FROEHLICH@WIAS-BERLIN.DE
Martin Eigel MARTIN.EIGEL@WIAS-BERLIN.DE
Weierstrass Institute of Applied Analysis and Stochastic
Anton-Wilhelm-Amo-Straße 39
Berlin, 10117, Germany

Editor: My editor

Abstract

The unification of the neural and the symbolic paradigms of artificial intelligence is a long-standing challenge. We in this work introduce the tensor-network formalism tnreason, 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. This unified treatment identifies tensor-network contractions as a fundamental inference class and formulates efficiently scaling reasoning algorithms, originating from probability theory and propositional logics, as contraction message passing schemes. The framework enables the definition and training of hybrid logical and probabilistic models, which we call Hybrid Logic Networks. We further demonstrate the concepts in the accompanying python library tnreason.

Keywords: Tensor Networks, Neuro-Symbolic AI

1 Introduction

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, to which probabilistic and logical approaches have been developed. 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 McCarthy (1959), 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 Pearl (1988); Koller and Friedman (2009), provide insights based on encoded variable independences and causality Pearl (2009). While probabilistic models and Statistical Relational AI Nickel et al. (2016); Getoor and Taskar (2019) have improved uncertainty handling, bridging these paradigms remains the central goal of *Neuro-Symbolic AI* Hochreiter (2022); Sarker et al. (2022); Colelough and Regli (2024). Building on early connectionist approaches Towell and Shavlik (1994);

Avila Garcez and Zaverucha (1999) and aligning with statistical relational learning Marra et al. (2024), 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 Richardson and Domingos (2006), a fully unified substrate that treats logical and probabilistic inference as instances of the same operation, is still missing.

We in this work propose to fill the gap between the probabilistic, neural and logical paradigm with *tensor networks*, in a framework called tnreason. Tensor spaces capture both the semantics of logical formulas (by boolean tensors) and probability distributions (by normalized non-negative tensors). 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, existence of sufficient statistics and neural model decomposition, are equivalent to tensor network decompositions. Moreover, we identify tensor network contraction as fundamental inference instances, which correspond with computation of marginal distributions and the decision of entailment. 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.

Efficient schemes to perform inference are known as message passing schemes (appearing as sum-product, belief propagation, etc.).. These schemes underly distributed approaches to contract tensor networks based on efficient local contractions.

To capture both logical and probabilistic models, we introduce *Computation-Activation Networks (CompActNets)*, an architecture that organizes this reasoning into two complementary substructures. The *computation network* encodes the structural relations of a problem, such as logical dependencies or more generic statistics, while the *activation network* assigns semantic or numerical values to these structures. We generalize the standard tensor network diagrammatic notation to visually represent these components, defining the inference process as a graphical tensor contraction between structure and activation. Logical inference emerges when activations are boolean, probabilistic inference when they are real-valued, and hybrid reasoning when both coexist.

1.1 Related works

The unification of neural, symbolic and probabilistic approaches to interpretable model architectures has been a long-standing aim. A central goal is to achieve *intrinsic explainability.* Unlike post-hoc interpretations—which analyze input influence or fit surrogate models after training Lipton (2018); Barredo Arrieta et al. (2020)—the proposed framework encodes symbolic relations that remain directly readable, building explainability into the architecture itself.

Historically rooted in quantum many-body physics White (1993), 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 Affleck et al. (1987). 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 Sander et al. (2025b,a). To address the topological constraints of MPS, the landscape of architectures was subsequently expanded to include Projected Entangled Pair States

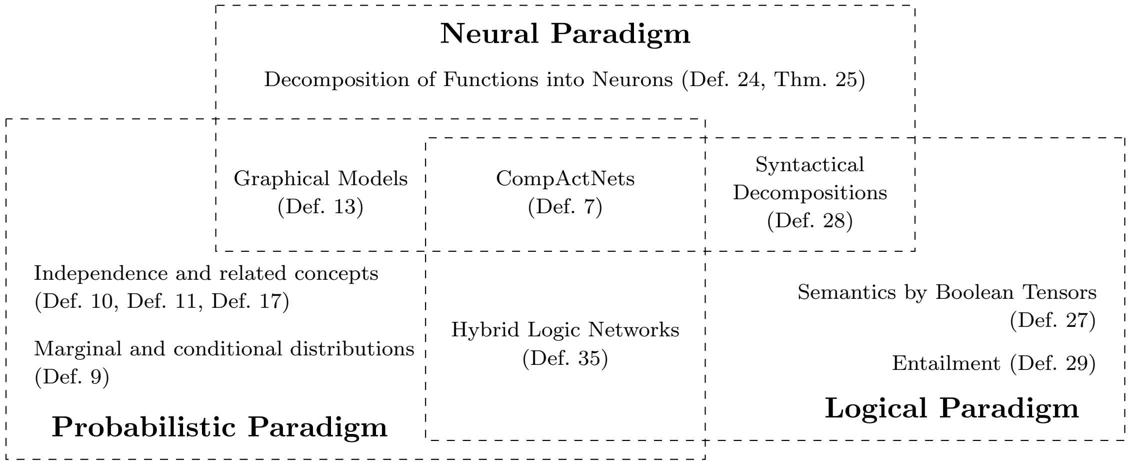


Figure 1: Sketch of the Concepts in the Neural, Probabilistic and Logical Paradigms, which we define based on tensor-network decompositions and contractions.

(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 Orús (2019); Berezutskii et al. (2025). 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 Hagemann et al. (2025); Eigel et al. (2017); Goessmann et al. (2020). Furthermore, they exhibit properties helpful for handling these high-dimensional spaces, such as restricted isometry properties Goessmann (2021). Recent advancements have demonstrated the efficacy of these methods in capturing multiscale phenomena in fluid dynamics and turbulence, proving that the tensor network formalism offers a robust alternative to classical numerical schemes Gourianov et al. (2025).

Tensor Networks have recently gained interest as a unifying language for AI, framed by Logical Tensor Networks Badreddine et al. (2022) and Tensor Logic Domingos (2025). Furthermore, the MeLoCoToN approach Ali (2025) 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 Hackbusch (2012). 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 Oseledets (2011); Hackbusch and Kühn (2009); Hitchcock (1927).

1.2 Structure of the paper

The paper is organized as follows. Section 2 introduces the basic notation for categorical variables, tensors, and tensor networks, establishing the formal framework on which all subsequent reasoning structures are defined. Section 3 develops the probabilistic representation of reasoning through soft activation, showing how exponential-family distributions can be expressed as tensor networks based on independence assumptions and sufficient statistics. Section 5 turns to hard activation, formulating propositional logic within the same tensor framework and demonstrating how logical inference, entailment, and knowledge bases can be represented by boolean tensors and contractions. Section 6 unifies these two perspectives in the concept of Hybrid Logic Networks, which integrate hard logical constraints with soft probabilistic activations, thereby forming the core of the computation–activation architecture. The paper concludes with coding examples in section 7 that illustrate the expressive power and interpretability of this unified tensor-based reasoning approach.

2 Foundations

We in this section introduce our 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 will first provide a formal definition as real maps from index sets enumerating the coordinates of vectors, matrices and larger order tensors.

Definition 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 here introduced tensors in a non-canonical way based on categorical variables assigned to its axis. While coming as syntactic sugar at this point, this will allow 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 1 (Delta Tensor) *Given a set of variables* $X_{[d]} = (X_0, \ldots, X_{d-1})$, *where* $d \geq 1$, *with identical dimension* m, *the delta tensor is the element*

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

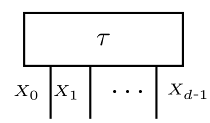
with coordinates

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

We will depict this tensor by black dots, which will sometimes appear auxiliarly in tensor network diagrams (see e.g. Figure 4). For $d = 1$, *the delta tensor is the trivial vector, which coordinates are constant* 1, *which we denote by* $\mathbb{I}[X]$.

2.2 Tensor Networks and Contractions

We will use a standard visualization of tensors (dating back to Penrose (1987)) by blocks with lines depicting the axes of the tensor. In addition we denote to each axis of the tensor the corresponding variable X_k:



Drawing on the association of variables with nodes and of tensors with hyperedges build by the assigned variables we continue with the definition of tensor networks.

Definition 2 (Tensor Network) *Let* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *be a hypergraph with nodes decorated by categorical variables* X_v *with dimensions* $m_v \in \mathbb{N}$ *and hyperedges* $e \in \mathcal{E}$ *decorated by core tensors*

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

where we denote by X_e *the set of categorical variables* X_v *with* $v \in e$. *Then we call the set*

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

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 (The CP format) *The Candecomp-Parafac (*CP *(Hitchcock (1927)) tensor format corresponds in our notation with a hypergraph (see Figure 2)*

- *Nodes by* $X_{[d]}$ *and a single hidden variable* I, *decorated by dimensions* $m_{[d]}$ *and* n.

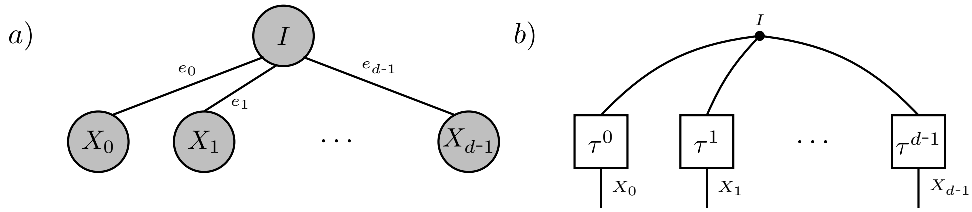


Figure 2: Hypergraph to a CP format. a) Node-centric design. b) Corresponding tensor-network on the edges of the hypergraph.

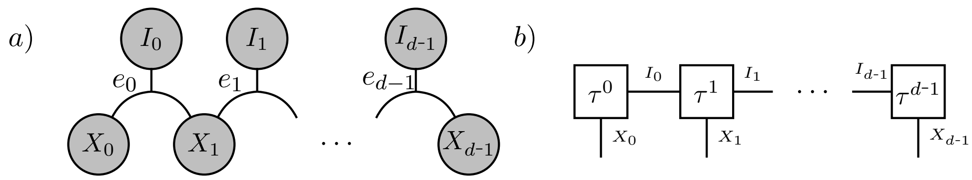


Figure 3: Hypergraph to a TT format. a) Node-centric design. b) Corresponding tensor-network on the edges of the hypergraph.

- *Edges by*

$$\left\{e_k = \{X_k, I\} \,:\, k \in [d]\right\}$$

  each decorated by a matrix $\tau^{e_k}[X_k, I]$.

Example 3 (The TT **format)** *The Tensor-Train (*TT*) (see Oseledets (2011)) format corresponds in our notation with a hypergraph (see Figure 3)*

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

- *Edges by*

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

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

2.3 Generic Contractions

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

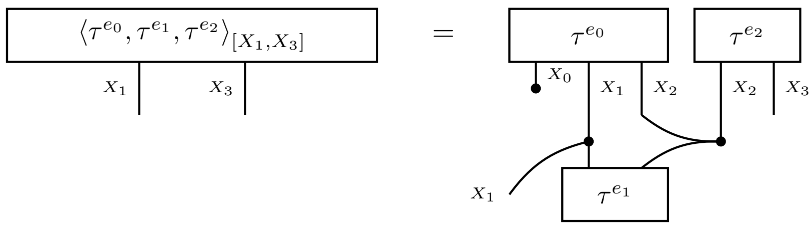


Figure 4: Graphical depiction of a tensor network contraction with the open variables X_1, X_3. Open variables are depicted by those without a dot at the end of the line.

Definition 3 *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)*

$$\left\langle \tau^{\mathcal{G}} \right\rangle_{[X_{\mathcal{U}}]} \in \bigotimes_{v \in \mathcal{U}} \mathbb{R}^{m_v} \tag{2}$$

defined coordinatewise by the sum

$$\left\langle \tau^{\mathcal{G}} \right\rangle_{[X_{\mathcal{U}} = x_{\mathcal{U}}]} = \sum_{x_{\mathcal{V}/\mathcal{U}} \in \times_{v \in \mathcal{V}/\mathcal{U}} [m_v]} \left(\prod_{e \in \mathcal{E}} \tau^e \left[X_e = x_e \right] \right) . \tag{3}$$

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

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

Example 4 (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 its axes. Then their tensor product is the map

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

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

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

2.4 Normalizations

Definition 4 *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*

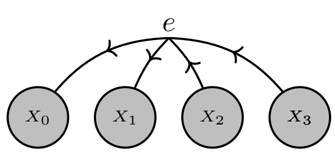
$$\left\langle \tau\left[X_{\mathcal{V}}\right]\right\rangle_{\left[X_{\mathcal{V}^{\text{out}}}|X_{\mathcal{V}^{\text{in}}}=x_{\mathcal{V}^{\text{in}}}\right]} = \begin{cases} \frac{\langle\tau\rangle\left[X_{\mathcal{V}^{\text{out}}},X_{\mathcal{V}^{\text{in}}}=x_{\mathcal{V}^{\text{in}}}\right]}{\langle\tau\rangle\left[X_{\mathcal{V}^{\text{in}}}=x_{\mathcal{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

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

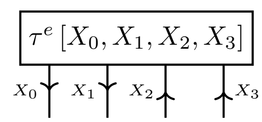
$$\tau\left[X_{\mathcal{V}}\right] = \left\langle \tau\left[X_{\mathcal{V}}\right]\right\rangle_{\left[X_{\mathcal{V}/\mathcal{V}^{\text{in}}}|X_{\mathcal{V}^{\text{in}}}\right]} .$$

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

a)



b)



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 5 (One-hot encodings to Factored Representations) *To any variable X taking values in $[m]$ the one-hot encoding of any state $x \in [m]$ is the vector with coordinates*

$$\epsilon_x\left[X = \tilde{x}\right] := \begin{cases} 1 & \text{if} \quad x = \tilde{x} \\ 0 & \text{else}. \end{cases} \tag{4}$$

To any tuple $(X_0, \ldots, X_{d-1})$ of variables taking values in $\bigtimes_{k\in[d]}[m_k]$ the one-hot encoding of a state tuple $x_{[d]} = (x_0, \ldots, x_{d-1})$ 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 6 (Basis encoding of maps between state sets) *Let there be two systems with factored representations by variables $X_{[d]}$ and $Y_{[p]}$, and a map*

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

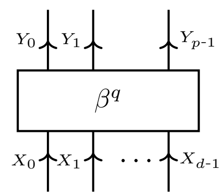
between these state sets. Then the basis encoding of q is a tensor

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

defined by

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

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 $q : \mathcal{U}^{\mathrm{in}} \to \mathcal{U}^{\mathrm{out}}$ between arbitrary sets, and enumerating maps I_{in} and I_{out} for both sets we define the basis encoding of q as

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

where X, Y are variables taking values in $[|\mathcal{U}^{\mathrm{in}}|]$ and $[|\mathcal{U}^{\mathrm{out}}|]$. We present in Example 13 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 7 (Computation-Activation Network (CompActNets)) *Given 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) as the* computation network *and by $\xi\left[Y_{\mathcal{V}}\right]$ as 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 $\Lambda^{t,\mathcal{G}} \subset \Lambda^{t,\mathrm{MAX}}$.

3 The Probabilistic Paradigm

We here 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 Clifford and Hammersley (1971)) to motivate graphical models. Further 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 8 (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 $\mathbb{P}\left[X_{[d]}\middle|Z\right]$ is a family of joint probability distributions, if for any $z \in \mathrm{val}(Z)$ the slice $\mathbb{P}\left[X_{[d]}\middle|Z = z\right]$ is a joint probability distribution.

Example 5 (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* $z \in [0,1]$ *by*

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

Notice, that for each slice with respect to $z \in [0,1]$ *we have by the binomial theorem* $\left\langle \mathbb{P}\left[X_{[d]}, Z = z\right]\right\rangle_{[\varnothing]} = 1$ *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* $z \in [0,1]$*:*

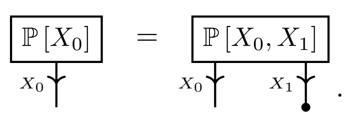
$$\mathbb{P}\left[X_{[2]}\middle|Z = z\right] = \begin{array}{cc} & \quad X_1:\ 0 \qquad\qquad 1 \\ X_0:\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} (1-z)^2 & z\cdot(1-z) \\ z\cdot(1-z) & z^2 \end{bmatrix} \end{array} .$$

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

Definition 9 *For any distribution* $\mathbb{P}[X,Y]$ *the marginal distribution is given by the contraction*

$$\mathbb{P}[X] := \langle \mathbb{P}[X,Y]\rangle_{[X]}$$

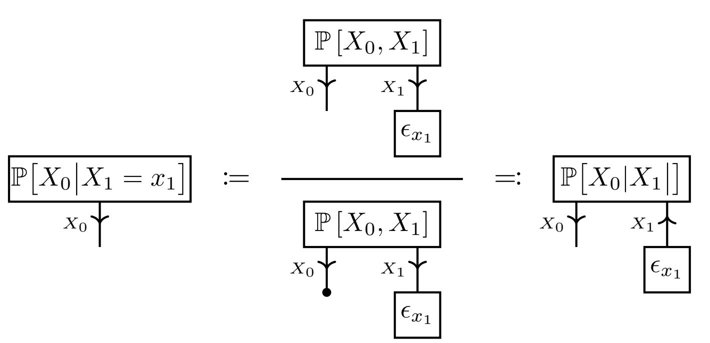
which is depicted by the diagram



The conditional distribution of X *on* Y *is a tensor* $\mathbb{P}\left[X\middle|Y\right]$ *defined for* $y \in [m_1]$

$$\mathbb{P}\left[X\middle|Y = y\right] := \begin{cases} \frac{1}{m}\cdot \mathbb{I}[X] & \textit{if} \quad \langle \mathbb{P}[X, Y = y]\rangle_{[\varnothing]} = 0 \\ \frac{1}{\langle \mathbb{P}[X,Y=y]\rangle_{[\varnothing]}} \cdot \mathbb{P}[X, Y = y] & \textit{else} \end{cases}$$

and in the second case depicted by the diagram



3.2 The Independence Mechanism: Graphical Model Factorization

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

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

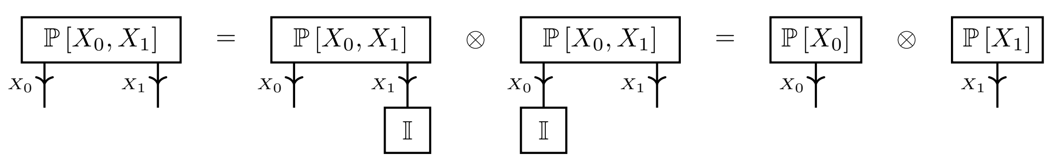
and therefore scales exponentially in the number of coordinates. To find efficient representation schemes of probability distributions by tensor networks, we need to exploit additional properties of the distribution. Independence leads to severe sparsifications of conditional probabilities and is therefore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

Definition 10 (Independence) *We say that X_0 is independent of X_1 with respect to a distribution $\mathbb{P}\left[X_0, X_1\right]$, if the distribution is the tensor product of the marginal distributions, that is*

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

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

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



Let us notice, 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, which 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.

Independence is, as we observed, a strong assumption, which is often too restrictive. Conditional independence instead is a less demanding assumption, which still implies efficient tensor network decompositions schemes. We introduce conditional independence as independence of variables with respect to conditional distributions.

Definition 11 (Conditional Independence) *Given a joint distribution of variables X_0, X_1 and X_2, such that $\mathbb{P}\left[X_2\right]$ is positive. We say that X_0 is independent of X_1 conditioned on X_2 if for any states $x_0 \in [m_0], x_1 \in [m_1]$ and $x_2 \in [m_2]$*

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

In this case we denote $(X_0 \perp X_1) \mid X_2$.

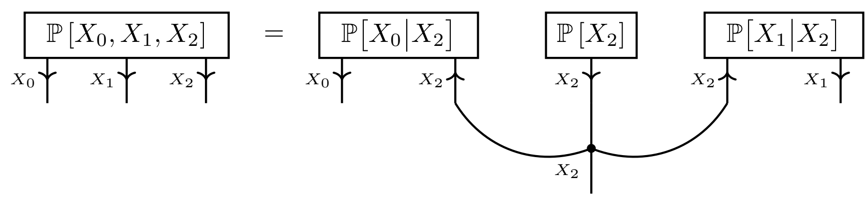
Conditional independence stated in Def. 11 has a close connection with independence stated in Def. 10. To be more precise, X_0 is independent of X_1 conditioned on X_2, if and only if X_0 is independent of X_1 with respect to any slice $\mathbb{P}\left[X_0, X_1 \middle| X_2 = x_2\right]$ of the conditional distribution $\mathbb{P}\left[X_0, X_1 \middle| X_2\right]$.

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

Corollary 12 *Let $\mathbb{P}\left[X_0, X_1, X_2\right]$ be a joint distribution. If and only if X_0 is independent of X_1 conditioned on X_2 the distribution satisfies*

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

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 13 (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*

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

We call the denominator

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

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

We define graphical models based on hypergraphs, to establish a direct connection with tensor network 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 Koller and Friedman (2009)). The probabilistic graphical models are along that alternative dual to tensor networks Robeva and Seigal (2019); Glasser et al. (2019).

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

Definition 14 (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 of clique-capturing established a correspondence of hyperedges with maximal cliques in the more canonical graph-based definition of Markov Networks Koller and Friedman (2009).

Definition 15 (Clique-Capturing Hypergraph) *We call a hypergraph $\mathcal{G}$* clique-capturing, *when each subset $\mathcal{U} \subset \mathcal{V}$ is contained in a hyperedge, if for any $a, b \in \mathcal{U}$ there is a hyperedge $e \in \mathcal{E}$ with $a, b \in \mathcal{U}$.*

Let us now show a characterization of Markov Networks in terms of conditional independencies.

Theorem 16 (Hammersley-Clifford Factorization Theorem) *Let there be a positive probability distribution $\mathbb{P}[X_{\mathcal{V}}]$ and a clique-capturing hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Then the following are equivalent:*

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

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

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

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

Proof Shown in Appendix Sect. A. ■

By Thm. 16 the conditional independence structure of $\mathbb{P}[X_{\mathcal{V}}]$ determines a global tensor network decomposition of $\mathbb{P}[X_{\mathcal{V}}]$. 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 was however not required in our characterization of independencies and conditional independencies by the existence of corresponding tensor decompositions (see Def. 10 and Def. 11).

Example 6 (Independent Boolean Variables: Coin toss interpretation) *Let there be d boolean variables $X_{[d]}$, which are i.i.d. drawn from a positive distribution $\mathbb{P}[X]$. From the pairwise independencies of X_k it follows with the Hammersley-Clifford Factorization Thm. 16 that the distribution is representable by an elementary tensor network, that is*

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

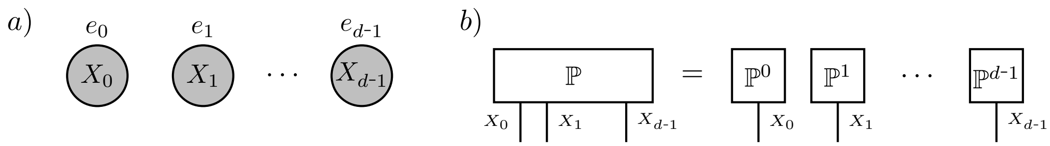


Figure 5: Decomposition of a probability distribution with independent variables (see Example 6). The independences are captured by the elementary hypergraph a), which edges contain single nodes. The corresponding tensor $\mathbb{P}\left[X_{[d]}\right]$ is then represented by a Markov Network on the elementary hypergraph, where each factor is the marginal distribution of the corresponding variable.

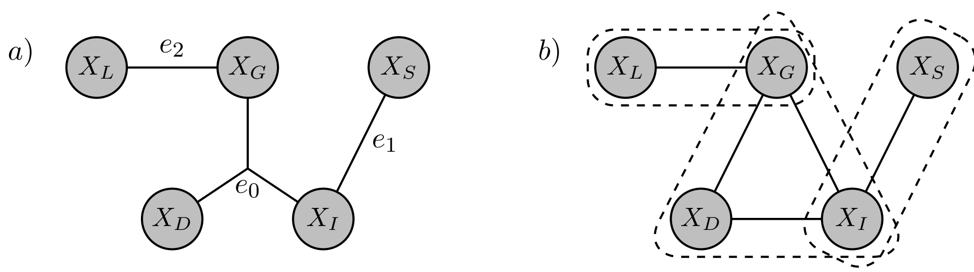


Figure 6: Hypergraph a) capturing the conditional independences of the student example. The cliques of the node adjacency graph are higlighted in b), and coincide with hyperedges of the hypergraph. The hypergraph is therefore clique-capturing (see Def. 15).

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

Example 7 *We consider a classical example of a graphical model (see (Koller and Friedman, 2009, 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 in presense of Sufficient Statistics

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

Definition 17 *Let $\mathbb{P}[X, Z]$ 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 $\mathbb{P}[X, Z, Y_t] = \left\langle \mathbb{P}[X, Z], \beta^t[Y_t, X] \right\rangle_{[X,Z,Y_t]}$. We say that t is a sufficient statistic for Z if and only if X is independent of Z conditioned on Y_t.

Note that the independence in Def. 17 is true if and only if

$$\mathbb{P}\left[X \middle| Z, Y_t\right] = \mathbb{P}\left[X \middle| Y_t\right] \otimes \mathbb{I}[Z] \ .$$

Example 8 (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*

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

When knowing the value $tx_{[d]}$ 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

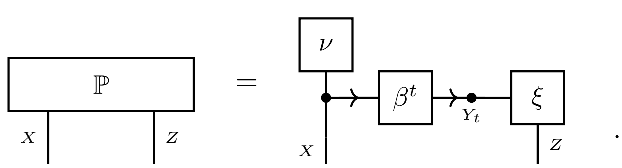
$$\mathbb{P}\left[X_{[d]}\right] = \left\langle \beta^t\left[Y_{[p]}, X_{[d]}\right], \xi\left[Y_{[p]}\right] \right\rangle_{\left[X_{[d]}\right]} .$$

Example 8 hints at a connection between sufficient statistics and decompositions into CompActNets. More generally, such decompositions are provided by the Fisher-Neyman Factorization Theorem.

Theorem 18 (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 $\nu[X]$ and $\xi[Y_t, Z]$ such that*

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

We depict this equation diagrammatically by



Proof Shown in more generality in the appendix, see Thm. 39. ■

Notice, that the definition of sufficient statistic does not make use of the marginal distribution $\mathbb{P}[Z]$. 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 distribution $\mathbb{P}[Z]$. We then use the Thm. 18 to embed such families in CompActNets.

Corollary 19 *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 a activation tensors $\xi\left[Y_{[p]}, Z\right]$ such that for any $z \in \mathrm{val}(Z)$*

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

Example 9 (Order Statistic for Boolean Variables: Coin toss interpretation) *Let there be d boolean variables $X_{[d]}$ and a family $\{\mathbb{P}^\theta\left[X_{[d]}\right] : \theta \in \Theta\}$ 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*

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

When the order statistic is sufficient, the detailed order of the outcomes is uninformative about the member $\theta \in \Theta$ from which the random variables have been drawn. Let us now investigate those families for which t^+ is a sufficient statistic. By the Fisher-Neyman Factorization Theorem Thm. 18 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 $\theta \in \Theta$

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

The family of distributions, such that the variables $X_{[d]}$ are i.i.d. with respect to each other (see Example 6) are the special case, where $\nu\left[X_{[d]}\right] = \mathbb{I}\left[X_{[d]}\right]$ and the family is labeled by $\theta \in [0,1]$ such that for $\theta \in (0,1)$ and $k \in [d+1]$

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

and for $\theta \in \{0,1\}$

$$\xi^\theta\left[Y_+\right] = \begin{cases} \epsilon_0\left[Y_+\right] & \text{if} \quad \theta = 0 \\ \epsilon_d\left[Y_+\right] & \text{if} \quad \theta = 1 \end{cases} .$$

The marginal distribution $\mathbb{P}^\theta\left[Y_+\right]$ is then the binominal distribution $B(d,\theta)$.

The Fisher-Neyman Theorem is the fundamental motivation for the CompActNets Architecture:

- The decomposition of $\xi\left[Y_{[p]}\right]$ is called *activation network*.

- The decomposition of $\beta^t\left[Y_{[p]}, X_{[d]}\right]$ is called *computation network.*

Example 10 (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

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

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

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

In other words, 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 in case of elementary activation tensors

A classical theorem by Pitman-Koopman-Darmois (see Pitman (1936)) states, that whenever a family with constant support and a finite sufficient statistic for arbitrary large data sets is in an exponential family. We now restrict the activation cores to specific elementary tensors, which correspond with further assumptions on the dependence of $\mathbb{P}$ and t made by exponential families. For a discussion of further universal properties of exponential families, such that the existence of priors and entropy maximizers, see Murphy (2022).

Definition 20 (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 $l \in [p]$ the image $\mathrm{im}\,(t_l)$ by an interpretation map*

$$I_l \,:\, [|\mathrm{im}\,(t_l)|] \to \mathrm{im}\,(t_l)\,.$$

For any canonical parameter vector $\theta\,[L] \in \mathbb{R}^p$ we build the activation cores $\alpha^{l,\theta}\,[Y_l]$ for each coordinate $y_l \in [|\mathrm{im}\,(t_l)|]$ by

$$\alpha^{l,\theta}\,[Y_l = y_l] = \exp\left[\theta\,[L = l] \cdot I_l(y_l)\right]$$

and define the distribution

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

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

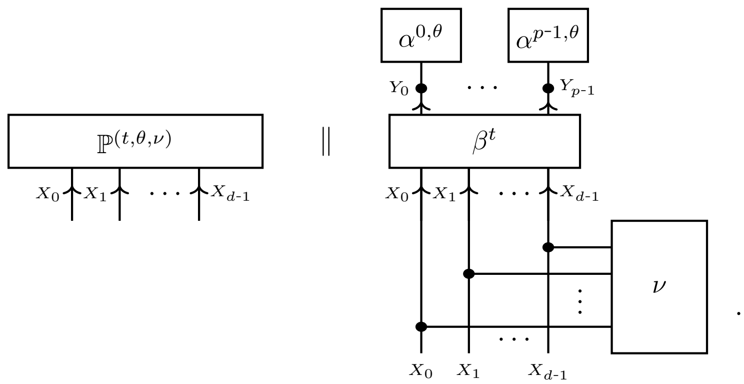


Figure 7: Tensor Network diagram of a member of an exponential family $\mathbb{P}^{t,\nu}\left[X_{[d]}|\Theta=\theta\right]$ before normalization as an CompActNet with elementary activation, that is an element in $\Lambda^{t,\mathrm{EL},\nu}$.

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 11 (Joint distributions of two booleans) *In general, joint distribution of two Boolean variables X_0, X_1 are 2x2 matrices of non-negative coordinates summing to 1:*

$$\mathbb{P}\left[X_{[2]}\right] = \begin{bmatrix} p_{0,0} & p_{0,1} \\ p_{1,0} & p_{1,1} \end{bmatrix}$$

In the following example, we will assume at different points that X_0, X_1 have a sufficient statistic, are independent and they have positive distributions. By the normalization constraint, $p_{1,1}$ is determined from $p_{0,0}, p_{0,1}$ and $p_{1,0}$, which leaves us with three free parameters.

$$\mathbb{P}\left[X_{[2]}\right] = \begin{bmatrix} p_{0,0} & p_{0,1} \\ p_{1,0} & 1-(p_{0,0}+p_{0,1}+p_{1,0}) \end{bmatrix}$$

Let us now restrict to those distributions, which have the sum $X_0 + X_1$ as a sufficient statistic. They need to satisfy $p_{0,1} = p_{1,0}$ (since in that cases the statistic is 1 and the definition of sufficiency is that the distribution conditioned on the statistic is uniform), leaving us with two free parameters.

$$\mathbb{P}\left[X_{[2]}\right] = \begin{bmatrix} p_{0,0} & p_{0,1} \\ p_{0,1} & 1-p_{0,0}-2p_{0,1} \end{bmatrix}$$

This symmetry also implies, that the distributions are identically distributed, i.e. for any $x \in \{0,1\}$ we have

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

Restricting further to those, where X_0 and X_1 are independent and the distribution is everywhere supported, brings us to the rank one formulation of the distribution

$$\mathbb{P}[X_0, X_1] = \begin{bmatrix} \mathbb{P}[X_0 = 0]\,\mathbb{P}[X_1 = 0] & \mathbb{P}[X_0 = 0]\,\mathbb{P}[X_1 = 1] \\ \mathbb{P}[X_0 = 1]\,\mathbb{P}[X_1 = 0] & \mathbb{P}[X_0 = 1]\,\mathbb{P}[X_1 = 1] \end{bmatrix} = \mathbb{P}[X_0] \otimes \mathbb{P}[X_1]$$

In terms of an exponential family with the head count as a sufficient statistic, we parametrize the distribution by the canonical parameter $\theta \in \mathbb{R}$ as

$$\mathbb{P}[X_0] = \frac{1}{1 + \exp[\theta]} \begin{bmatrix} 1 \\ \exp[\theta] \end{bmatrix}$$

Note, that with this parametrization the probabilities for head and tail automatically have the form $p, (1-p)$.

$$\mathbb{P}[X_0, X_1] = \frac{1}{(1 + \exp[\theta])^2} \begin{bmatrix} 1 \\ \exp[\theta] \end{bmatrix} \begin{bmatrix} 1 & \exp[\theta] \end{bmatrix}$$

We can interpret this distribution as two independent coin tosses with outcome X_0 and X_1 and head probability

$$\mathbb{P}[X_0 = 1] = \mathbb{P}[X_1 = 1] = \frac{\exp[\theta]}{1 + \exp[\theta]}$$

which is the sigmoid of θ and inverted by the logit

$$\theta = \ln\left[\frac{\mathbb{P}[X_0 = 1]}{1 - \mathbb{P}[X_0 = 1]}\right].$$

Consistent with the above parametrization, we have a uniform distribution of X_0 and X_1 in the fair coin toss case $\mathbb{P}[X_0 = 1] = 0.5$, where $\theta = 0$.

As a Computation-Activation Network we can represent any distribution $\mathbb{P}[X_0, X_1]$ with the head count $+$ as sufficient statistic by

$$\mathbb{P}[X_0, X_1] = \left\langle \beta^+[Y_+, X_0, X_1], \xi[Y_+] \right\rangle_{[X_0, X_1 | \varnothing]},$$

such that

$$\begin{aligned} \mathbb{P}[X_0 = x_0, X_1 = x_1] &= \frac{1}{Z} \left\langle \beta^+[Y_+, X_0, X_1], \xi[Y_+] \right\rangle_{[X_0 = x_0, X_1 = x_1]} \\ &= \frac{1}{Z} \sum_{y_+ \in [2]} \beta^+[Y_+ = y_+, X_0 = x_0, X_1 = x_1] \cdot \xi[Y_+ = y_+] \\ &= \frac{1}{Z} \xi[Y_+ = x_0 + x_1], \end{aligned}$$

where the normalization constant Z cancels out any multiplicative constant $\lambda \in \mathbb{R}\backslash\{0\}$ in ξ and the equation above implies

$$\xi[Y] = \lambda \cdot \begin{bmatrix} p_{0,0} \\ p_{0,1} \\ p_{1,1} \end{bmatrix}.$$

We choose $\lambda = 1/p_{0,0} = (1+\exp[\theta])^2$ *in the following. Among these distribution, the exponential family with the head count statistic is then parametrized by activation tensors*

$$\xi[Y] = \begin{bmatrix} 1 \\ p_{0,1}/p_{0,0} \\ p_{1,1}/p_{0,0} \end{bmatrix} = \begin{bmatrix} 1 \\ \exp[\theta] \\ \exp[2\theta] \end{bmatrix},$$

since $p_{0,1} = \mathbb{P}[X_0 = 0] \cdot \mathbb{P}[X_0 = 1] = (1+\exp[\theta])^{-1} \cdot \exp[\theta](1+\exp[\theta])^{-1}$ *and* $p_{1,1} = (\exp[\theta](1+\exp[\theta])^{-1})^2$.

3.5 Efficient Contractions by Message Passing

Contractions of tensor networks are generally hard to solve. We here investigate message passing algorithms, which decompose contractions into a sequence of subcontractions, which are passed as messages through the tensor network. The resulting algorithm is called Belief Propagation Algorithm 1. 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 $e_0 \cap e_1 \neq \varnothing$.

Algorithm 1 Tree Belief Propagation

Require: Tensor network $\tau^{\mathcal{G}}$ on a hypergraph $\mathcal{G}$
Ensure: Messages $\{\chi_{e_0 \rightarrow e_1}[X_{e_0 \cap e_1}] : (e_2, e_0) \in \mathcal{E}^{\rightarrow}\}$

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

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

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

Theorem 21 *The messages in the tree belief propagation are contracted to local marginals, that is for each* $e_0 \in \mathcal{E}$ *we have*

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

We show Thm. 21 based on the following lemma. We denote for each pair (e_0, e_1) the subset $\mathcal{E}^{\rightarrow (e_0, e_1)} \subset \mathcal{E}$ as the subset of edges $e \in \mathcal{E}$, for which each path to e_1 passes through e_0. Note, that by construction $e_0 \in \mathcal{E}^{\rightarrow (e_0, e_1)}$.

Lemma 22 *For any tensor network on a tree hypergraph, Algorithm* **??** *terminates in the tree-based implementation and returns final messages*

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

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

$n = 1$: 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.

$n \to n+1$: Let us 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

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

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

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

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 [Proof of Thm. 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

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

Here we used Lem. 22 in the second equation. ■

We examplify the usage of Algorithm 1 on the Markov Network of Example 7.

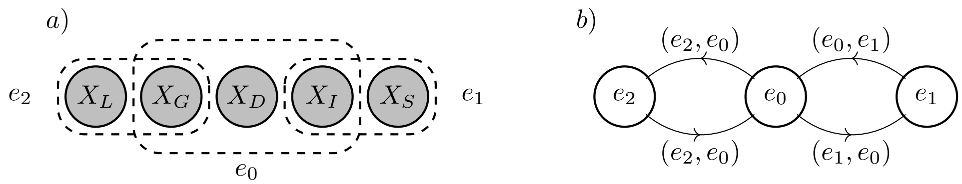


Figure 8: a) Sketch of the overlap of the edges, resulting in the message directions b) $\mathcal{E}^{\rightarrow} = \{(e_2, e_0), (e_0, e_2), (e_0, e_1), (e_1, e_0)\}$.

Example 12 (Continuation of Example 7) *Let us now exemplify the Belief Propagation Algorithm 1 on the Markov Network in the student example (see Example 7). The directions of the messages result from the hyperedge overlaps (see Figure 8 a) and the resulting directions $\mathcal{E}^{\rightarrow}$ 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 exectution of (e_2, e_0) the message (e_0, e_1) is placed on S and after exectution of (e_1, e_0) the message (e_0, e_2).*

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

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

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 decomposeable into neurons can be represented by tensor networks. To this end we formalize discrete neural models in decomposition graphs and formally proof the corresponding decomposition of their basis encodings. Particular examples will be presented in the next section by propositional formulas with sparse syntactical descriptions.

4.1 Function Decomposition

As a main principle of tensor decompositions we now show that basis encodings of composition functions is the contraction of basis encodings to the components.

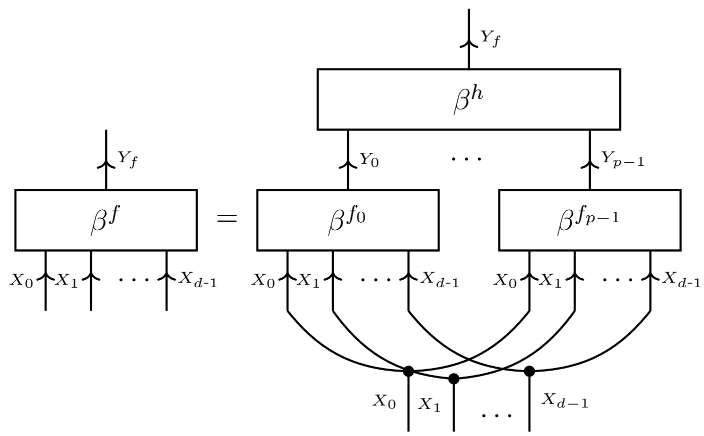


Figure 9: Tensor network decomposition of a the basis encoding of a function f, which is the composition of the functions $f_0, \ldots, f_{p-1}$ with a function h.

Lemma 23 *Let $f\left[X_{[d]}\right]$ be a composition of a p-ary connective function h and functions $f_l\left[X_{[d]}\right]$, where $l \in [p]$, 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)

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

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

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

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

Let us now define a more generic decomposition of discrete functions.

Definition 24 *A* decomposition hypergraph *is a directed acyclic hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that*

- *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*

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

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

$$g^e \,:\, \bigtimes_{v\in e^{\mathrm{in}}} \mathcal{U}^v \rightarrow \bigtimes_{v\in e^{\mathrm{out}}} \mathcal{U}^v\,.$$

- *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 $v^{\mathrm{in}}_{[d]}$. We abbreviate the corresponding variables by $X_{v^{\mathrm{in}}_{[d]}} = X_{[d]}$.*
- *The nodes not appearing as an incoming node are enumerated by $v^{\mathrm{out}}_{[p]}$. We abbreviate the variables by $X_{v^{\mathrm{out}}_{[l]}} = Y_{[p]}$.*

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

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

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

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

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

$$q_{\mathcal{G}} \,:\, \bigtimes_{k\in[d]} \mathcal{U}^{v^{\mathrm{in}}_k} \rightarrow \bigtimes_{l\in[p]} \mathcal{U}^{v^{\mathrm{out}}_l} \quad , \quad q_{\mathcal{G}} = \bigtimes_{l\in[p]} q_{v^{\mathrm{out}}_l}$$

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

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

Theorem 25 *For any decomposition hypergraph $\mathcal{G}$ with composition formula $q_{\mathcal{G}}$ we have*

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

Proof The claim follows by induction from the leafs to the root and iteratively applying Lem. 23. ■

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

Example 13 (Sum of integers in m-adic representation) *Let us develop a tensor network representation of integer summations on the set $[m^d] = \{0, \ldots, m^d - 1\}$, 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

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

We notice 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. 24) consistent in $4 \cdot d$ 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

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

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. 25 we have a decomposition of the basis encoding to $q_{\mathcal{G}}$ (see Figure 10b) as

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

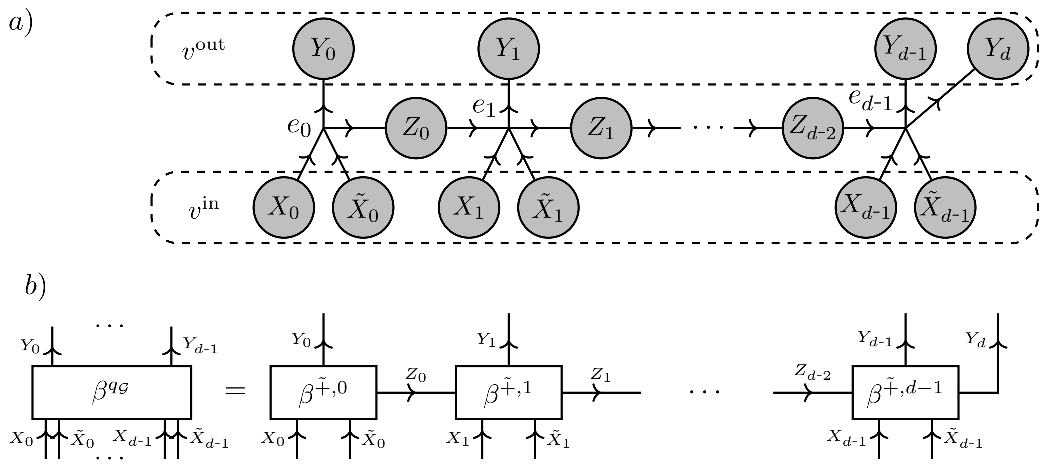


Figure 10: Example of a decomposition hypergraph to the sum of integers (see Example 13). a) Hypergraph of directed edges e_k for $k \in [d]$, each decorated by a integer summation + preparing an index Y_k of the resulting sum. b) Corresponding tensor network decomposition of the basis encoded composition function, which is the sum of integers in m-adic representation.

4.2 Directed Message Passing

We now present an efficient inference algorithm based on tensor network contractions.

Algorithm 2 Directed Belief Propagation

Require: Tensor network $\tau^{\mathcal{G}}$ on a directed hypergraph $\mathcal{G}$
Ensure: Messages $\{\chi_{e_0 \to e_1}[X_{e_0 \cap e_1}] : (e_2, e_0) \in \mathcal{E}^{\to}\}$

Prepare directed message directions

$$\mathcal{E}^{\to} = \left\{ \left((\mathcal{V}_0^{\text{in}}, \mathcal{V}_0^{\text{out}}), (\mathcal{V}_1^{\text{in}}, \mathcal{V}_1^{\text{out}}) \right) : \mathcal{V}_0^{\text{in}} \cap (\mathcal{V}_1^{\text{in}}, \mathcal{V}_1^{\text{out}}) = \varnothing, \mathcal{V}_1^{\text{out}} \cap (\mathcal{V}_0^{\text{in}}, \mathcal{V}_0^{\text{out}}) = \varnothing, \mathcal{V}_0^{\text{out}} \cap \mathcal{V}_1^{\text{in}} \neq \varnothing \right\}$$

Initialize a message queue $S = \{(e_2, e_0) : e_2 \quad$ 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 $\{\chi_{e_0 \to e_1}[X_{e_0 \cap e_1}] : (e_2, e_0) \in \mathcal{E}^{\to}\}$

Let us now apply the Directed Belief Propagation Algorithm 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.

Theorem 26 *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*

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

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 $\epsilon_{x_k}\left[X_k\right]$.

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

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

Thus also the new message is 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.

Example 14 (Continuation of Example 13) *Let us now show how Algorithm 2 can be exploited to compute by an efficient message passing algorithm 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*

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

to the hypergraph of Example 13 and decorate them by the digit one-hot encodings $\epsilon_{x_k}\left[X_k\right]$ 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 consists then in the messages from the digit encoding. As sketched in Figure 11 to each digit there are three messages (with the exception of the last

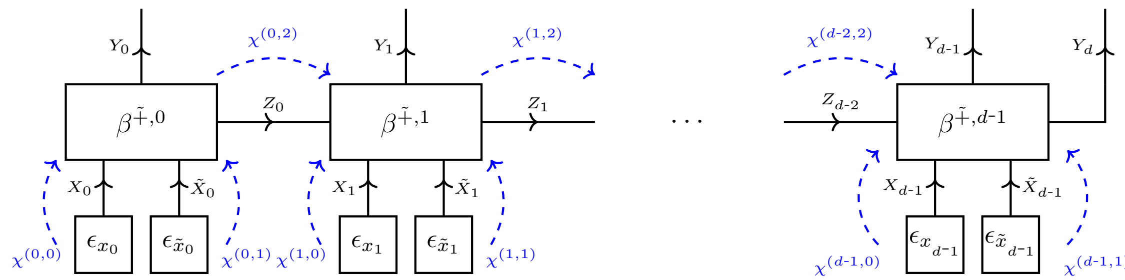


Figure 11: Computation of the integer sum in m-adic representation by the Directed Belief Propagation Algorithm 2 (see Example 14). The summands are represented by one-hot encodings of the digits $x_{[d]}$ and $\tilde{x}_{[d]}$, from which the messages start. The k-th digit (for $k \in \{0, \ldots, d-1\}$) 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.

being two), which can be scheduled in consecutive epochs $\chi^{(k,[3])}$. We then have by Thm. 26 for $k \in [d-1]$ that

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

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

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

We notice, that the hypergraph representing this instance is a tree and by Thm. 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 strengthen 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

Definition 27 *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*

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

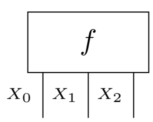
where we associate True $\leftrightarrow 1$ *and* False $\leftrightarrow 0$*. If there is a model to a propositional formula, we say the formula is* satisfiable*.*

Example 15 *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

$f\left[X_{[3]}\right] =$



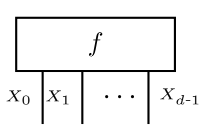
$$= \begin{array}{c|cc} & X_1{=}0 & X_1{=}1 \\ \hline X_0{=}0 & 0 & 1 \\ X_0{=}1 & 1 & 1 \end{array}\ (X_2 = 0) \qquad \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\ (X_2 = 1)\,.$$

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

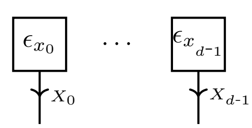
$$x_{[3]} \in \{(1,0,0),(0,1,0),(1,1,0)\}\,.$$

The formula f is therefore satisfiable.

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, i.e. fulfills the formula, a propositional formula can be written as the sum over the one-hot encodings of its models.



$$= \sum_{\substack{x_0,\dots,x_{d-1}\in\bigtimes_{k\in[d]}[2] \\ f(x_0,\dots,x_{d-1})=1}}$$



This decomposition corresponds to the CP decomposition of a tensor.

Example 16 *For the formula described in Example 15, we have*

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

where we denote the vectors $\epsilon_1[Y] = [0,1]^T$ *and* $\epsilon_0[Y] = [1,0]^T$. *Then for the model* $x_{[3]} = (1,1,0)$ *it holds*

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

Model counts by contraction Each coordiante of the propositional formula is either a 1 or 0 encoding if 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 checking if $\langle f \rangle_{[\varnothing]} > 0$.

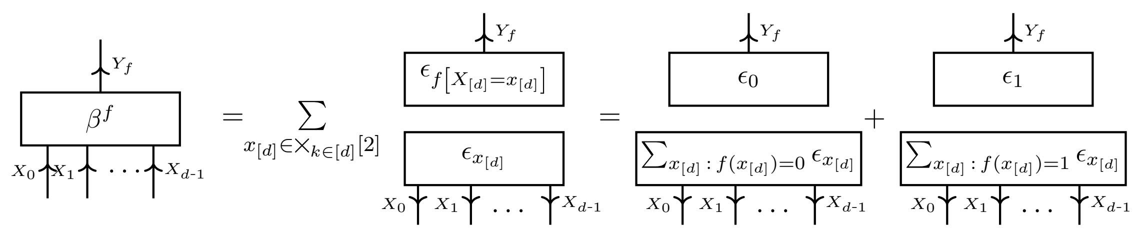
Basis encoding Representing booleans by elements in $\{0,1\}$ leads to the problem, that negation is an affine transformation and can not be represented by multilinear tensors. Therefore, instead of using this *coordinate calculus* an approach based on *basis calculus* is employed, which is explained in this section. To be able to express different kinds of connectives and finally any propositional formula by multi-linear tensors, booleans are encoded by one-hot encodings as defined in Def. 5. Propositional formulas f can be expressed in terms of a tensor describing the mapping and its negation by

$$\beta^f\left[Y_f = y_f, X_{[d]} = x_{[d]}\right] = \begin{cases} 1 & \text{if} \quad f\left[X_{[d]} = x_{[d]}\right] = y_f \\ 0 & \text{else} \end{cases} . \tag{5}$$

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

$$\beta^f\left[Y_f, X_{[d]}\right] = \epsilon_1[Y_f] \otimes f\left[X_{[d]}\right] + \epsilon_0[Y_f] \otimes \neg f\left[X_{[d]}\right] . \tag{6}$$

In our graphical notation this property is visualized by



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

Example 17 (Logical Negation and Conjunction) *The basis encodings of the negation* $\neg : [2] \to [2]$ *is the matrix*

$$\beta^{\neg}[Y_{\neg}, X] = \begin{array}{cc} & Y_{\neg}: \; 0 \quad 1 \\ X_0: \begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{array}$$

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

$$\beta^{\land}\left[Y_{\land}, X_0, X_1\right] = \begin{array}{c|c} & \\ Y_{\land} \begin{smallmatrix}0\\1\end{smallmatrix} & \begin{bmatrix}1\\0\end{bmatrix}\end{array} \otimes \begin{array}{cc} & \overset{X_1}{\begin{smallmatrix}0 & 1\end{smallmatrix}} \\ X_0 \begin{smallmatrix}0\\1\end{smallmatrix} & \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}\end{array} + \begin{array}{c|c} & \\ Y_{\land} \begin{smallmatrix}0\\1\end{smallmatrix} & \begin{bmatrix}0\\1\end{bmatrix}\end{array} \otimes \begin{array}{cc} & \overset{X_1}{\begin{smallmatrix}0 & 1\end{smallmatrix}} \\ X_0 \begin{smallmatrix}0\\1\end{smallmatrix} & \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}\end{array} = \begin{array}{ccc} & \overset{X_1}{\begin{smallmatrix}0 & 1\end{smallmatrix}} & \\ X_0 \begin{smallmatrix}0\\1\end{smallmatrix} & \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix} & \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix} \\ & \underset{Y_{\land}}{0} & 1 \end{array}$$

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

$$\beta^{\lor}\left[Y_{\lor}, X_0, X_1\right] = \begin{array}{ccc} & \overset{X_1}{\begin{smallmatrix}0 & 1\end{smallmatrix}} & \\ X_0 \begin{smallmatrix}0\\1\end{smallmatrix} & \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix} & \begin{bmatrix}0 & 1\\1 & 1\end{bmatrix} \\ & \underset{Y_{\lor}}{0} & 1 \end{array}$$

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

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

Both f and $\neg f$ are thus Computation-Activation Networks to the statistic $\{f\}$ and the hard activation tensor $\epsilon_1[Y_f]$, respectively $\epsilon_0[Y_f]$. This representation of propositional formulas with respect to basis encoding thus leads to Computation-Activation Networks, which were also used to describe probability distributions in the last section. In this way the soft and hard logic can be combined in one framework.

5.2 Decomposition of Propositional Formulas

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

Definition 28 (Syntactical Decompositions) *A syntactical decomposition of a propositional formula f is a decomposition hypergraph (see Def. 24) such that all nodes are decorated with the dimension $m_v = 2$ and composition function f.*

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

5.3 Contractions to decide entailment

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.

Definition 29 (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. Russell and Norvig (2021)) entailment in propositional logics is defined as the the unsatisfiability of $\mathcal{KB} \wedge \neg f$. This is equivalent to Def. 29, 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 18 ($n^2 \times n^2$ 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 $(r_0, r_1, c_0, c_1, i) \notin E^{\text{start}}$ 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

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

or false

$$\mathcal{KB} \wedge \left(\bigwedge_{(r_0,r_1,c_0,c_1,i)\in E^{\text{start}}} X_{r0,r1,c0,c1,i}\right) \models \neg X_{r0,r1,c0,c1,i}\,. \tag{7}$$

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 $(r_0, r_1, c_0, c_1, i) \notin E^{\text{start}}$. 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 18 a solution algorithm to solve this instance, after we have derived sparse tensor network representations in Example 20.

5.4 Efficient Representation of Knowledge Bases

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

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

To show efficient repesentations we will use the following identities.

Lemma 30 (Computation Network Symmetries) *We have for the d-ary $\wedge$-connective (where $d \in \mathbb{N}$) and the unary $\neg$-connective that*

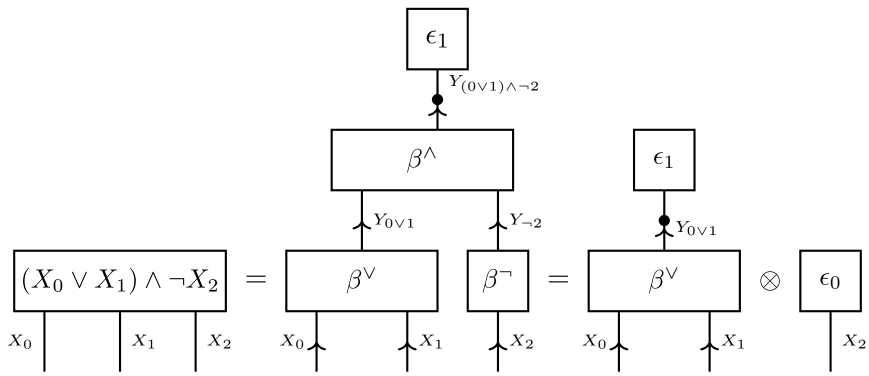
$$\left\langle \epsilon_1 \left[Y\right], \beta^{\wedge}\left[Y, X_{[d]}\right]\right\rangle_{\left[X_{[d]}\right]} = \bigotimes_{k \in [d]} \epsilon_1 \left[X_k\right] \quad \text{and} \quad \left\langle \epsilon_1 \left[Y\right], \beta^{\neg}\left[Y, X\right]\right\rangle_{[X]} = \epsilon_0 \left[X\right]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

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

Example 19 (Computation Network Symmetries (Continuation of Example 15)) *For the propositional formula*

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

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 31 *For any knowledge base* $\mathcal{KB}\left[X_{[d]}\right] = \bigwedge_{l\in[p]} f_l\left[X_{[d]}\right]$ *it holds that*

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

Proof With Lem. 30 we have

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

■

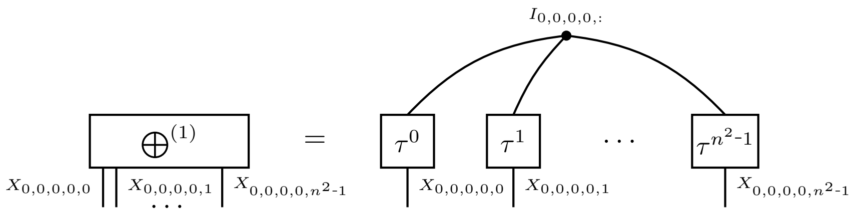
Example 20 (Sparse representation of Sudoku rule Knowledge Base) *We now exploit Thm. 31 to find efficient tensor network representation of the Sudoku knowledge base from Example 18. We directly get, that the knowledge base* $\mathcal{KB}^n$ *of Sudoku rules is a tensor network of the* $4\cdot n^4$ *constraint 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

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

which is a CP *decomposition (for the example of the position constraint at* $(r0, r1, c0, c1) = (0,0,0,0)$*) depicted as:*



Given evidence E^{start} we denote the Sudoku Knowledge Base $\mathcal{KB}^{n,E^{\text{start}}}$. We model the Sudoku Knowledge Base $\mathcal{KB}^{n,E^{\text{start}}}$ as a tensor network on a hypergraph $\mathcal{G}^{\text{Sudoku},n}$ 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 an 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 $\{X_{(r0,r1,c0,c1,i)}\}$ a 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 Message-passing for Entailment

Since contracting the whole tensor is often infeasible and for instance for the Sudoku example would correspond to solving the whole problem, local contractions can be considered to decide in some cases. Here a local contraction describes the calculation of contractions along few closely connected legs in the tensor network. Now, if the local contraction of any legs leads to a zero-tensor in the network decomposition, the whole contraction amounts to zero, and the knowledge base entails f.

Theorem 32 (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 futhermore add entailed formulas to the knowledge base without the latter, as we show next.

Theorem 33 (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

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

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. 33 provides us with a mean 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 neighbored 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 $\{\chi_{e_0\to e_1}[X_{e_0\cap e_1}] : (e_2, e_0) \in \mathcal{E}^{\to}\}$ containing entailment statements

Initialize a queue $S = \mathcal{E}^{\to}$ of message directions
Initialize messages $\chi_{e_0\to e_1}[X_{e_0\cap e_1}] = \mathbb{I}[X_{e_0\cap e_1}]$ 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

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

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

Theorem 34 *All messages during constraint propagation are sound, i.e*

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

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

$$\tau^{\mathcal{G}} = \left\langle \{\tau^{\mathcal{G}}\} \cup \{\chi_{e_0\to e_1}[X_{e_0\cap e_1}] : (e_0, e_1) \in \mathcal{E}^{\to}\} \right\rangle_{[X_{\mathcal{V}}]} \, . \tag{8}$$

By Thm. 32 we then have for the first message send along the pair (e_0, e_1) that

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

Let us now assume that at an arbitrary state of the algorithm the inequality holds for all previous sent messages. By Thm. 33 we can add contract the messages on the tensor network without changing it, and (8) thus still holds. We then conclude with Thm. 32 that the claimed property also holds for the new message. ■

Example 21 (Constraint Propagation for the Sudoku of Example 18) *We iteratively solve a Sudoku puzzle by determining a possible value based on neighboring cells, rows and squares (using Thm. 32) and adding to our knowledge (using Thm. 33). 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:*

$$
\begin{array}{|cc|cc|}\hline 1 & & 3 & 2 \\ & 2 & & \\ \hline & & 4 & \\ 4 & 3 & & \\ \hline \end{array}
=
\begin{array}{|cc|cc|}\hline 1 & & 3 & 2 \\ 3 & 2 & & \\ \hline & & 4 & \\ 4 & 3 & & \\ \hline \end{array}
= \dots =
\begin{array}{|cc|cc|}\hline 1 & 4 & 3 & 2 \\ 3 & 2 & 1 & 4 \\ \hline 2 & 1 & 4 & 3 \\ 4 & 3 & 2 & 1 \\ \hline \end{array}
$$

To illustrate the first reasoning step of assigning $X_{0,1,0,0,2}$ *we make the following entailment steps applying Thm. 32. We also depict in Figure 12 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 12) along the directions*

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

  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 12) 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. 33) 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 12) 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*

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

6 Hybrid Logic Networks

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

6.1 Parametrization

Definition 35 (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 to* t *is the set*

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

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

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

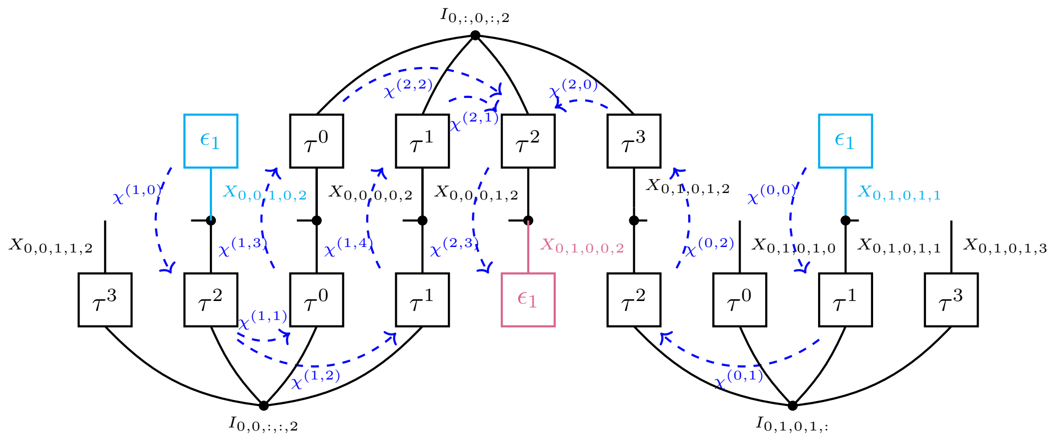


Figure 12: The tensor network decomposition of 3 out of $4 \cdot 2^2 = 64$ rules in the $2^2 \times 2^2$ Sudoku knowledge base (see Example 20), namely to the number 3 appearing once in the $(0,0)$-square (above), the number 3 appearing once in the $(0,0)$-row (below left) and a unique number appearing at the $(0,1,0,1)$-position (below right). The evidence of the number 3 assigned at the position $(0,0,1,0)$ is sketched on 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 21), 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 21.

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 $\bigotimes_{l\in[p]} \xi^l\left[Y_l\right]$, we can always find a corresponding tuple in $\mathcal{P}_p$ by choosing[1]

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

further for all $l \in A$

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

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

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

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

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

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

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

Example 22 (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 A1 is typically booked.*

We formalize this with the statistic

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

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

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

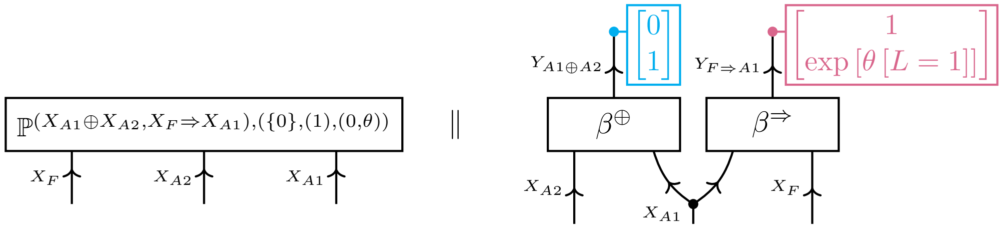
Since the first feature is hard, the "soft logic activation tensor" should be invariant under the first coordinate of the canonical parameter and we set $\theta\left[L=0\right] = 0$. We choose the soft parameters as $\theta[L] = [0, \theta[L=1]]^\intercal$ to achieve

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

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

$$\mathbb{P}^{(X_{A1}\oplus X_{A2}, X_F\Rightarrow X_{A1}),\left(\{0\},(1),(0,\theta)\right)}[X_{A1}, X_{A2}, X_F] = \frac{1}{1+3\cdot\exp[\theta]} \; X_F=0: \begin{array}{c|cc} X_{A1}\backslash X_{A2} & 0 & 1 \\ \hline 0 & 0 & \exp[\theta] \\ 1 & \exp[\theta] & 0 \end{array} \qquad X_F=1: \begin{array}{c|cc} X_{A1}\backslash X_{A2} & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & \exp[\theta] & 0 \end{array}$$

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 $\mathbb{P}^{t,(A,y_A,\theta)}\left[X_{[d]}\right]$ 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 notice that this is ∞ if and only if there is a data point $j \in [m]$ with

$$f^{t,(A,y_A)}\left[X_{[d]} = x_{[d]}^j\right] = 0\,.$$

If this is not the case, we can rewrite the loss using the empirical mean vector $\mu_D[L] \in \mathbb{R}^p$, which is defined for $l \in [p]$ as

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

by

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

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

$$A = \{l \,:\, \mu_D[L = l] \in \{0, 1\}\} \quad \text{and} \quad y_l = \mu_D[L = l] \quad \text{for} \quad l \in A\,.$$

We further optimize the coordinates $l \in [p]/A$ of $\theta[L] \in \mathbb{R}^p$ alternatingly by the coordinate descent steps

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

where

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

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 the computation of 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 Ganapathi et al. (2008)).

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

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

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

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

    Set

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

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

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 (Goessmann, 2025, Chapter 9).

Example 23 (Continuation of Example 22) *Let us recall the statistic of Example 22 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 $\theta [L]$*)*

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

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

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

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* $\tau [Y_1]$ *will not change in any further iteration of the* While *and the parameter* $\theta [L = 1]$ *will therefore stay constant until the termination of the algorithm.*

6.3 Entailment by Hybrid Logic Networks

Let us now demonstrate a further usage 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 36 *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 we have the probabilistic entailment of* h *by* $\mathbb{P}^{t,(A,y_A,\theta)}$*, 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

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

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 therfore 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. 29 this vanishes if and only if $f^{t,(A,y_A)} \models h$. ■

Example 24 (Continuation of Example 23) *Let us consider again the Hybrid Logic Network* $\mathbb{P}^{(X_{A1}\oplus X_{A2}, X_F \Rightarrow X_{A1}),\left(\{0\},(1),(0,\ln[3])\right)}$ *from Example 23 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. 36 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. 29 this entailment holds, since by the De-Morgan rule*

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

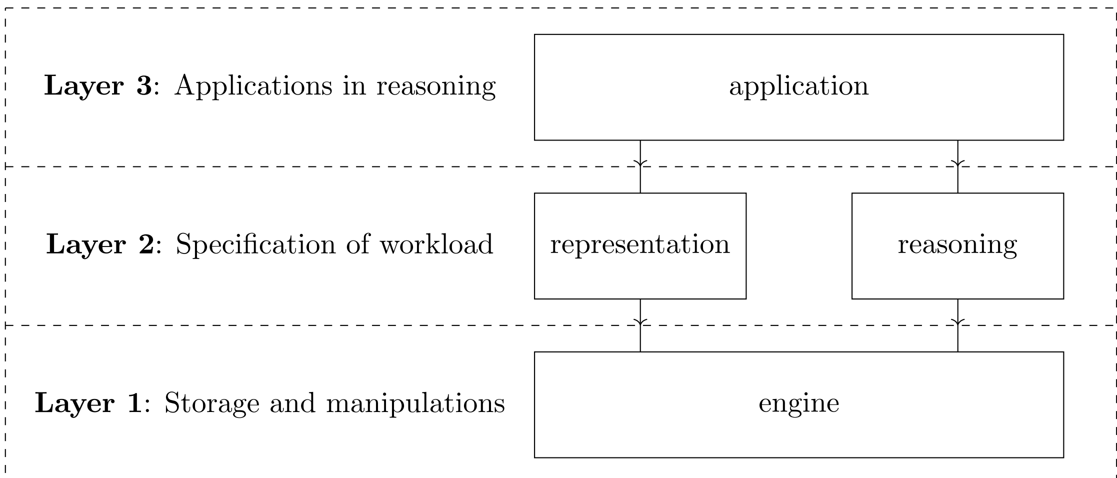
We thus conclude, that h *is probabilistically entailed by* $\mathbb{P}^{(X_{A1}\oplus X_{A2}, X_F \Rightarrow X_{A1}),\left(\{0\},(1),(0,\ln[3])\right)}$.

7 Implementation in the python library tnreason

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

7.1 Architecture

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



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 generalization 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 data sets.

7.2 Basic usage

We demonstrate the basic usage of the tnreason package with the implementation of Example 15. 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. 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 15 is a sum of the one-hot encodings of its three models (see Example 16) 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})])
```

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

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 coordinates of tensors can be retrieved by subscribing them with a posDict. We can for example check whether {"X_0":0,"X_1":1,"X_2"} 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 (Goessmann, 2025, Appendix A).

Following Def. 2, 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 19):

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

Here we apply a nested list description of syntactical hypergraphs (see Def. **??**) with a specification of the logical connectives at 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"]})
```

A depiction of the underlying hypergraph as a factor graph, which highlights edges by blue blocks and nodes by red blocks is can be created by engine.draw_factor_graph(fDecomp) (see Figure 13). Single tensors can be retrieved by contractions depending on a tensor network and a specification of the open variables (for explanation of the suffices see Figure 13), 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 (Goessmann, 2025, Appendix A).

8 Conclusion & Outlook

This work has treated the representation of several models in tensor networks. Especially, probability distributions over discrete sets and propositional formulas can be represented in the introduced structure. Multiple properties, such as independence of variables for the distributions and connections of subsets for the propositional formulas, can be directly encoded into the architecture leading to sparse, memory-efficient representations. The exact representation also encourages the carrying over of analysis of the mathematical concepts to the CompActNets.

Model inference such as the calculating marginal distributions and deciding entailment are formulated by tensor network contractions. The computation of these contractions is

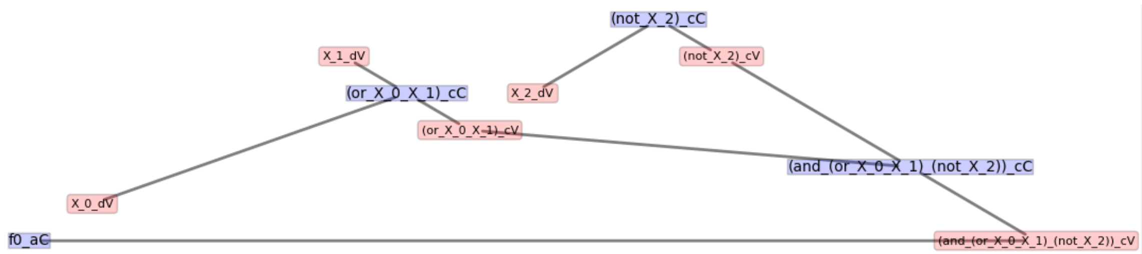


Figure 13: Factor graph highlighting a tensor network decomposition of the syntactical decomposition of the propositional formula of Example 19. Blue blocks highlight hyperedges carrying tensors and red blocks variables. The tensor label suffices `"_cC"` and `"_aC"` indicate whether the tensor is part 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.

however often a bottleneck. This bottleneck is related to the NP-hardness of probabilistic inferences in graphical models (see Koller and Friedman (2009)) and of logical reasoning (see Russell and Norvig (2021)). Approximation schemes to overcome this bottleneck can be summarized under the umbrella of variational inference (see Wainwright and Jordan (2008)), such as loopy message passing schemes and mean field methods. We in this work presented message-passing schemes as belief propagation in probability theory and syntactical inference algorithms in logics. We can understand them as approximation of (potentially intractible) contractions and will dedicate future work to study them in the tnreason formalism. While these schemes are developed either for graphical models (sum-product algorithms, belief-propagation) or more general exponential families (expectation-propagation), we anticipate to derive them in future work for generic CompActNets. Further frequently applied schemes are particle-based inference schemes such as Gibbs sampling.

Beyond the theoretical integration of differentiable architectures into the transformer architecture of Large Language Models Vaswani et al. (2017), 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 LLM is grounded by the explainable, instance-adaptive topology of the Computation-Activation Network.

Acknowledgments and Disclosure of Funding

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

References

Ian Affleck, Tom Kennedy, Elliott H. Lieb, and Hal Tasaki. Rigorous results on valence-bond ground states in antiferromagnets. *Physical Review Letters*, 59(7):799–802, August 1987. doi: 10.1103/PhysRevLett.59.799. URL https://link.aps.org/doi/10.1103/PhysRevLett.59.799. Publisher: American Physical Society.

Alejandro Mata Ali. Explicit Solution Equation for Every Combinatorial Problem via Tensor Networks: MeLoCoToN, February 2025. URL http://arxiv.org/abs/2502.05981.

Artur S. Avila Garcez and Gerson Zaverucha. The Connectionist Inductive Learning and Logic Programming System. *Applied Intelligence*, 11(1):59–77, July 1999. ISSN 1573-7497. doi: 10.1023/A:1008328630915. URL https://doi.org/10.1023/A:1008328630915.

Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic Tensor Networks. *Artificial Intelligence*, 303:103649, February 2022. ISSN 0004-3702. doi: 10.1016/j.artint.2021.103649. URL https://www.sciencedirect.com/science/article/pii/S0004370221002009.

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, June 2020. ISSN 1566-2535. doi: 10.1016/j.inffus.2019.12.012. URL https://www.sciencedirect.com/science/article/pii/S1566253519308103.

A. V. Berezutskii, I. A. Luchnikov, and A. K. Fedorov. Simulating quantum circuits using the multi-scale entanglement renormalization ansatz. *Physical Review Research*, 7(1):013063, January 2025. doi: 10.1103/PhysRevResearch.7.013063. URL https://link.aps.org/doi/10.1103/PhysRevResearch.7.013063. Publisher: American Physical Society.

George Casella and Roger Berger. *Statistical Inference*. Cengage Learning, Pacific Grove, Calif, June 2001. ISBN 978-0-534-24312-8.

P. Clifford and J. M. Hammersley. Markov fields on finite graphs and lattices. *Unpublished*, 1971. URL https://ora.ox.ac.uk/objects/uuid:4ea849da-1511-4578-bb88-6a8d02f457a6.

Brandon C Colelough and William Regli. Neuro-Symbolic AI in 2024: A Systematic Review. Jeju, South Korea, 2024.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, Hoboken, N.J, 2nd edition edition, September 2006. ISBN 978-0-471-24195-9.

Pedro Domingos. Tensor Logic: The Language of AI, October 2025. URL http://arxiv.org/abs/2510.12269. arXiv:2510.12269 [cs].

Martin Eigel, Max Pfeffer, and Reinhold Schneider. Adaptive stochastic Galerkin FEM with hierarchical tensor representations. *Numerische Mathematik*, 136(3):765–803, July 2017. ISSN 0945-3245. doi: 10.1007/s00211-016-0850-x. URL https://doi.org/10.1007/s00211-016-0850-x.

Varun Ganapathi, David Vickrey, John Duchi, and Daphne Koller. Constrained approximate maximum entropy learning of Markov random fields. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, UAI'08, pages 196–203, Arlington, Virginia, USA, July 2008. AUAI Press. ISBN 978-0-9749039-4-1.

Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning*. MIT Press, September 2019. ISBN 978-0-262-53868-8.

Ivan Glasser, Ryan Sweke, Nicola Pancotti, Jens Eisert, and Ignacio Cirac. Expressive power of tensor-network factorizations for probabilistic modeling. *Advances in Neural Information Processing Systems*, 32, 2019.

Alex Goessmann. *Uniform Concentration of Tensor and Neural Networks: An Approach towards Recovery Guarantees*. PhD Thesis, Technische Universität Berlin, Berlin, 2021. URL https://depositonce.tu-berlin.de/handle/11303/15990.

Alex Goessmann. The Tensor-Network Approach to Efficient and Explainable AI, 2025. URL https://github.com/EnexaProject/enexa-tensor-reasoning-documentation/.

Alex Goessmann, Ingo Roth, Gitta Kutyniok, Michael Götte, Ryan Sweke, and Jens Eisert. Tensor network approaches for data-driven identification of non-linear dynamical laws. In *Advances in Neural Information Processing Systems - First Workshop on Quantum Tensor Networks in Machine Learning*, page 21, 2020.

Nikita Gourianov, Peyman Givi, Dieter Jaksch, and Stephen B. Pope. Tensor networks enable the calculation of turbulence probability distributions. *Science Advances*, 11(5): eads5990, January 2025. doi: 10.1126/sciadv.ads5990. URL https://www.science.org/doi/10.1126/sciadv.ads5990. Publisher: American Association for the Advancement of Science.

W. Hackbusch and S. Kühn. A New Scheme for the Tensor Representation. *Journal of Fourier Analysis and Applications*, 15(5):706–722, October 2009. ISSN 1531-5851.

Wolfgang Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*. Springer Series in Computational Mathematics. Springer-Verlag, Berlin Heidelberg, 2012. ISBN 978-3-642-28026-9. doi: 10.1007/978-3-642-28027-6.

Paul Hagemann, Janina Schütte, David Sommer, Martin Eigel, and Gabriele Steidl. Sampling from Boltzmann Densities with Physics Informed Low-Rank Formats. In Tatiana A. Bubba, Romina Gaburro, Silvia Gazzola, Kostas Papafitsoros, Marcelo Pereyra, and

Carola-Bibiane Schönlieb, editors, *Scale Space and Variational Methods in Computer Vision*, pages 374–386. Springer Nature Switzerland, Cham, 2025. ISBN 978-3-031-92366-1.

Frank L. Hitchcock. The Expression of a Tensor or a Polyadic as a Sum of Products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927. ISSN 1467-9590. doi: https://doi.org/10.1002/sapm192761164.

Sepp Hochreiter. Toward a broad AI. *Communications of the ACM*, 65(4):56–57, January 2022. ISSN 0001-0782, 1557-7317. doi: 10.1145/3512715. URL https://dl.acm.org/doi/10.1145/3512715.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, Cambridge, Mass., 1. edition edition, July 2009. ISBN 978-0-262-01319-2.

Zachary C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, September 2018. ISSN 0001-0782. doi: 10.1145/3233231. URL https://dl.acm.org/doi/10.1145/3233231.

Giuseppe Marra, Sebastijan Dumančić, Robin Manhaeve, and Luc De Raedt. From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence*, 328:104062, March 2024. ISSN 0004-3702. doi: 10.1016/j.artint.2023.104062. URL https://www.sciencedirect.com/science/article/pii/S0004370223002084.

John McCarthy. Programs with Common Sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91. Her Majesty's Stationary Office, London, 1959. URL http://www-formal.stanford.edu/jmc/mcc59.html.

Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts London, England, March 2022. ISBN 978-0-262-04682-4.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1): 11–33, January 2016. ISSN 0018-9219, 1558-2256. doi: 10.1109/JPROC.2015.2483592. URL https://ieeexplore.ieee.org/document/7358050/.

Román Orús. Tensor networks for complex quantum systems. *Nature Reviews Physics*, 1 (9):538–550, September 2019. ISSN 2522-5820. doi: 10.1038/s42254-019-0086-7.

I. V. Oseledets. Tensor-Train Decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, January 2011. ISSN 1064-8275. doi: 10.1137/090752286. URL https://epubs.siam.org/doi/10.1137/090752286. Publisher: Society for Industrial and Applied Mathematics.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, s.l., September 1988. ISBN 978-1-55860-479-7.

Judea Pearl. *Causality: Models, Reasoning and Inference. Ausgezeichnet: ACM Turing Award for Transforming Artificial Intelligence 2011*. Cambridge University Press, Cambridge New York, NY Port Melbourne New Delhi Singapore, 2 edition, November 2009. ISBN 978-0-521-89560-6.

Roger Penrose. *Spinors and Space-Time: Volume 1, Two-Spinor Calculus and Relativistic Fields*. Cambridge University Press, Cambridge, February 1987. ISBN 978-0-521-33707-6.

E. J. G. Pitman. Sufficient statistics and intrinsic accuracy. *Mathematical Proceedings of the Cambridge Philosophical Society*, 32(4):567–579, December 1936. ISSN 1469-8064, 0305-0041. doi: 10.1017/S0305004100019307. URL https://www.cambridge.org/core/journals/mathematical-proceedings-of-the-cambridge-philosophical-society/article/abs/sufficient-statistics-and-intrinsic-accuracy/6A3E45FB1C423F3F684308F8910D6919.

Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, February 2006. ISSN 0885-6125, 1573-0565. doi: 10.1007/s10994-006-5833-1. URL http://link.springer.com/10.1007/s10994-006-5833-1.

Elina Robeva and Anna Seigal. Duality of graphical models and tensor networks. *Information and Inference: A Journal of the IMA*, 8(2):273–288, June 2019. ISSN 2049-8772. doi: 10.1093/imaiai/iay009.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach, Global Edition: A Modern Approach, Global Edition*. Pearson, Boston, 4 edition, May 2021. ISBN 978-1-292-40113-3.

Aaron Sander, Maximilian Fröhlich, Mazen Ali, Martin Eigel, Jens Eisert, Michael Hintermüller, Christian B. Mendl, Richard M. Milbradt, and Robert Wille. Quantum circuit simulation with a local time-dependent variational principle, August 2025a. URL http://arxiv.org/abs/2508.10096. arXiv:2508.10096 [quant-ph].

Aaron Sander, Maximilian Fröhlich, Martin Eigel, Jens Eisert, Patrick Gelß, Michael Hintermüller, Richard M. Milbradt, Robert Wille, and Christian B. Mendl. Large-scale stochastic simulation of open quantum systems. *Nature Communications*, 16(1): 11074, December 2025b. ISSN 2041-1723. doi: 10.1038/s41467-025-66846-x. URL http://arxiv.org/abs/2501.17913. arXiv:2501.17913 [quant-ph].

Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. Neuro-symbolic artificial intelligence: Current trends. *AI Communications*, 34(3):197–209, March 2022. ISSN 18758452, 09217126. doi: 10.3233/AIC-210084. URL https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/AIC-210084.

Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1):119–165, October 1994. ISSN 0004-3702. doi: 10.1016/0004-3702(94)90105-8. URL https://www.sciencedirect.com/science/article/pii/0004370294901058.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Martin J. Wainwright and Michael Irwin Jordan. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc, 2008. ISBN 978-1-60198-184-4.

Steven R. White. Density-matrix algorithms for quantum renormalization groups. *Physical Review B*, 48(14):10345–10356, October 1993. doi: 10.1103/PhysRevB.48.10345. URL `https://link.aps.org/doi/10.1103/PhysRevB.48.10345`. Publisher: American Physical Society.

Appendix 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. Koller and Friedman (2009) for Hammersley-Clifford and Casella and Berger (2001) 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 Clifford and Hammersley (1971)), we here proof the analogous statement for hypergraphs, where we have to demand the property of clique-capturing defined in Def. 15. We start with showing the following Lemmata to be exploited in the proof.

Lemma 37 *Let $\tau[X_{\mathcal{V}}]$ be a positive tensor and $y_{\mathcal{V}}$ an arbitrary index. Then we have*

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

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

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

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

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

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 38 *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

$$\prod_{\mathcal{W} \subset \mathcal{U}} (Z_{\mathcal{W}})^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}} = \prod_{\mathcal{W} \subset \mathcal{U}/\{a,b\}} \left(\frac{Z_{\mathcal{W}} \cdot Z_{\mathcal{W} \cup \{a,b\}}}{Z_{\mathcal{W} \cup \{a\}} \cdot Z_{\mathcal{W} \cup \{b\}}} \right)^{(-1)^{|\mathcal{U}|-|\mathcal{W}|}} . \tag{9}$$

From the independence assumption it follows that for any index x

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

Applying this in each squares bracket term of (9) 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 (9) is trivial, which establishes the claim. ■

We are finally ready to prove the Hammersley-Clifford Thm. 16 based on the Lemmata above.

Proof [Proof of Thm. 16] *ii)* ⇒ *i)* By Lem. 37 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. 16 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. 38 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

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

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

i) ⇒ *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

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

By Def. 11, 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. Cover and Thomas (2006)), we also show that also that definition is equivalent to the factorization of the family.

Theorem 39 (Factorization Theorem of Fisher and Neyman) *Let $\mathbb{P}$ be a joint distribution of variables Z, X with values $\mathrm{val}(Z), \mathrm{val}(X)$ and let $t(x)$ be a statistic. The following are equivalent:*

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

$$I(Z;X) = I(Z;Y_t)$$

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

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

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

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

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

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

and thus if and only if $i)$ holds

$$I(Z;X|Y_t) = 0 .$$

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

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

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

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

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

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 $x \in \mathrm{val}(X)$

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

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

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

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

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

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

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

and with the above at thus at almost all such pairs

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

This is equivalent to *ii)*. ■

Thm. 18 follows from Thm. 39 by the equivalence of *ii)* and *iii)*.

Appendix 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


```

```python
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)
```

```python
            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.2 Algorithm 4 (Alternating Moment Matching)

```python
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)
```

B.3 Example 13 and 14 (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
```

```python

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.4 Example 7 and 12 (Student Markov Network)

We here implement the Markov Network on the hypergraph of Example 7, 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.

```python
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.5 Example 18, 20 and 21 (Sudoku Game)

We implement the $n^2 \times n^2$ Sudoku with the start assignment given in Example 18 and apply the Constraint Propagation Algorithm 3 to deduce the full assignment. We then test whether the correct board assignment (given in Example 21) has been found.

```python
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.6 Example 22 and 23 (Toy Accounting Model)

```
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, 1],
    "A2": [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 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, 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
```