Università degli Studi di Bari
Dipartimento di Informatica

LACAM
Machine Learning

Learning Sum-Product Networks

Nicola Di Mauro **Antonio Vergari**

International Conference on Probabilistic Graphical Models
Lugano, Switzerland, 6-9 September 2016

Outline

→ **Representation**

→ **Inference**

→ **Interpretation**

→ **Learning**

→ **Applications**

→ **Representation Learning**

→ ***References***

The need for Tractable Inference

Probabilistic modeling of data aims at

- representing probability distributions *compactly*
- computing their marginals and modes *efficiently* (inference)
- learning them *accurately*

A solution is to use Probabilistic Graphical Models (PGMs)
However, PGMs are limited in

- representing compact distributions
- having intractable (exponential in their treewidth) exact inference in the worst case
  - falling back on approximate inference
- requiring and exponential sample size (wrt the number of variables)
- learning the structure since it requires inference

Exact inference in a tractable model may be better than performing approximate inference in an intractable model

The need for SPN

Why should you work on SPNs?

Sum-Product Networks (SPNs) are a type of probabilistic model[1]

- a class of deep probabilistic models that consist of many layers of hidden variables and can have unbounded treewidth

  $\rightarrow$ **probabilistic semantics and NN interpretation**

- inference in SPNs is guaranteed to be tractable
  - structure and parameters can be effectively and accurately learned

SPNs represent probability distributions and a corresponding exact inference machine for the represented distribution at the same time

Simple and effective algorithms to learn them

Successfully employed in several applications

---

[1] **Poon and Domingos, "Sum-Product Networks: a New Deep Architecture", 2011**

Representation

Density estimation

Given a set of i.i.d. samples $\{\mathbf{x}_i\}_{i=1}^N$ over RVs $\mathbf{X}$, the aim is to learn an estimator for the joint probability distribution $p_{\mathbf{X}}$

Unsupervised learning density estimators

- Bayesian Networks
- Markov Networks
- Kernel Density Estimators
- Autoregressive Neural Networks
- Sum-Product Networks
- ...

Once a density estimator is learned, one uses it to answer *queries*, i.e. to do ***inference***

(Different kinds of) Inference

Different types of models make different operations tractable

Operations that may be required to be efficient are

- $p(\mathbf{X} = \mathbf{x})$ (evidence)

  $\rightarrow$ tractable for SPNs, BNs

- $p(\mathbf{E}), \mathbf{E} \subset \mathbf{X}$ (marginals)

  $\rightarrow$ tractable for SPNs, hard in BNs (even approximate)

- $p(\mathbf{Q}|\mathbf{E}), \mathbf{Q}, \mathbf{E} \subset \mathbf{X}, \mathbf{Q} \cap \mathbf{E} = \emptyset$ (conditionals)

  $\rightarrow$ tractable for SPNs, hard in BNs (even approximate)

- $\arg\max_{\mathbf{q} \sim \mathbf{Q}} p(\mathbf{q}|\mathbf{E})$ (MPE assignment)

  $\rightarrow$ hard for both SPNs and BNs

- $Z = \sum_{\mathbf{x} \sim \mathbf{X}} \phi(\mathbf{x})$ (partition function)

  $\rightarrow$ tractable for SPNs, hard for MNs

- sampling: generate independent samples from the posterior distribution

Tractable Probabilistic Models

Due to the importance of efficient inference a lot of work has been devoted to learning probabilistic models for which inference is guaranteed to be tractable

- Graphical models
  - graphical models with low treewidth and their mixtures
  - thin junction trees
- Computational graphs from Knowledge Compilation
  - Arithmetic Circuits
  - Sentential Decision Diagrams[2]
- Neural Networks
  - Restricted Boltzmann Forest
  - Neural Autoregressive Distribution Estimator (NADE)[3]
  - Masked Autoencoder Distribution Estimator (MADE)[4]

---

[2] Darwiche, *Modeling and Reasoning with Bayesian Networks*, 2009

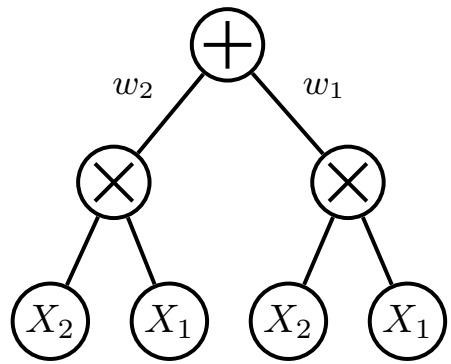
[3] Larochelle and Murray, "The Neural Autoregressive Distribution Estimator", 2011

[4] Germain et al., "MADE: Masked Autoencoder for Distribution Estimation", 2015

Sum-Product Networks

A *Sum-Product Network* S over RVs $\mathbf{X}$ is a rooted weighted DAG consisting of distribution ***leaves*** (network inputs), ***sum*** and ***product*** nodes (inner nodes).

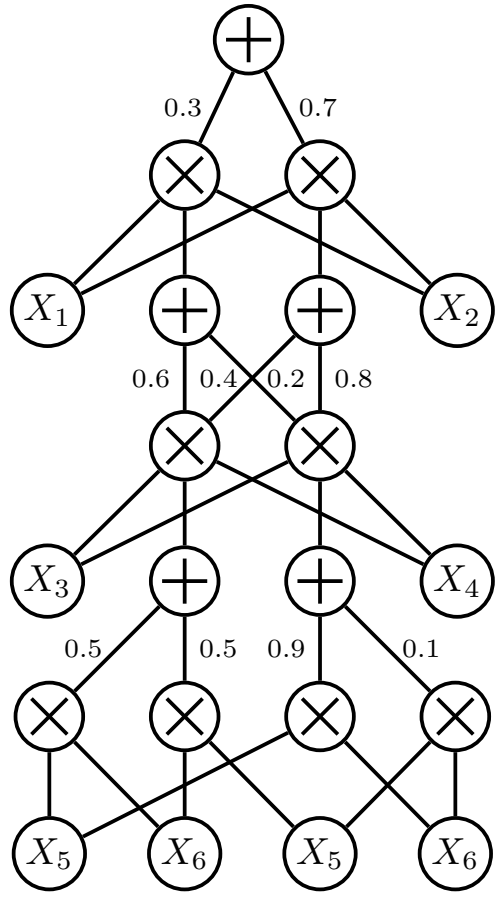
- A leaf n defines a tractable, possibly unnormalized, distribution ϕ_n over some RVs in $\mathbf{X}$.
- A nonnegative weight w_{nc} is associated to each edge linking a sum node n to $c \in \mathsf{ch}(n)$
  - $\mathsf{ch}(n)$: child (input) nodes of a node n
  - $\mathsf{pa}(n)$: parent (output) nodes of a node n
  - S_n: sub-network rooted at node n



Scopes

The ***scope*** of a node n in S is denoted as

$$\mathsf{sc}(n) \subseteq \mathbf{X}$$

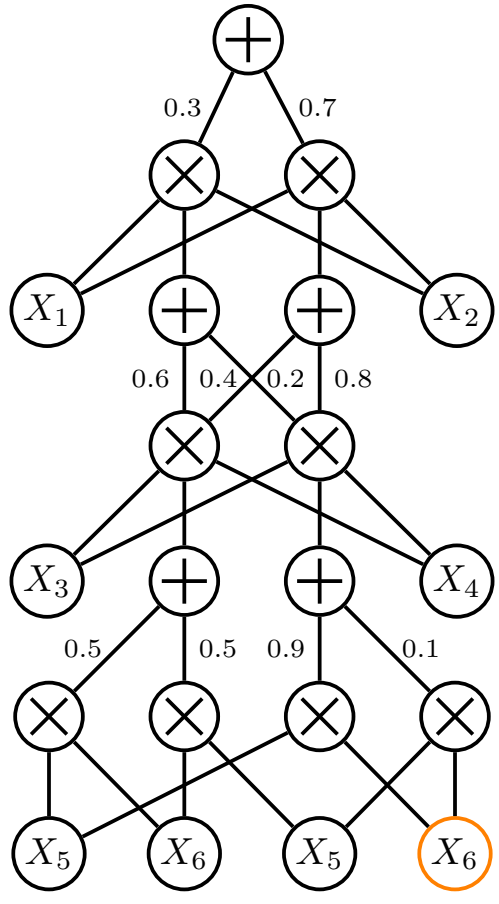


Scopes

The ***scope*** of a node n in S is denoted as

$$\text{sc}(n) \subseteq \mathbf{X}$$

- the scope of a leaf node n is defined as the set of RVs over which ϕ_n is defined
  E.g. sc(n) $= \{X_6\}$

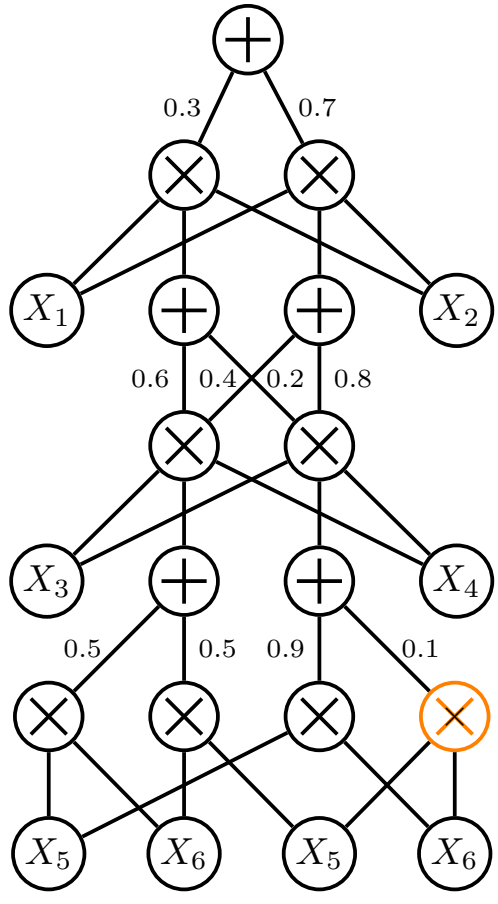


Scopes

The ***scope*** of a node n in S is denoted as

$$\mathsf{sc}(n) \subseteq \mathbf{X}$$

- the scope of a leaf node n is the set of RVs over which ϕ_n is defined
- the scope of an inner node n is defined as $\mathsf{sc}(n) = \bigcup_{c \in \mathsf{ch}(n)} \mathsf{sc}(c)$
  E.g. $\mathsf{sc}(\boxed{\text{n}}) = \{X_5, X_6\}$

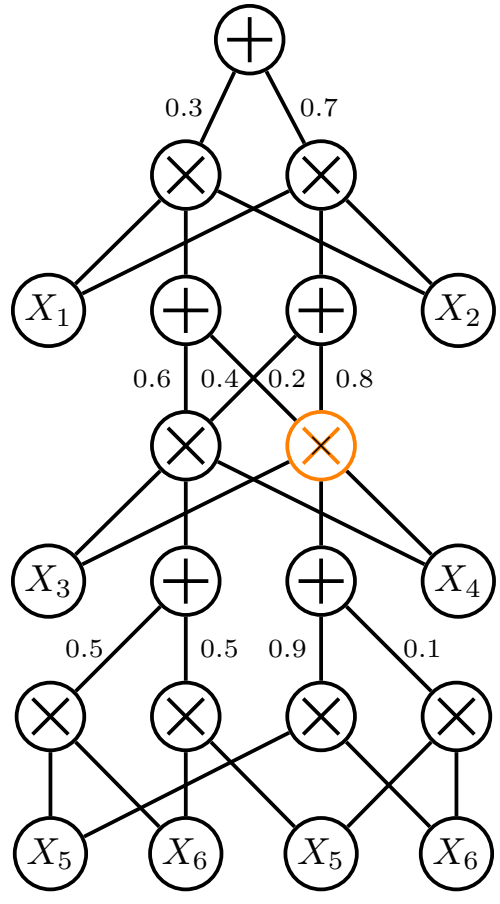


Scopes

The ***scope*** of a node n in S is denoted as

$$\text{sc}(n) \subseteq \mathbf{X}$$

- the scope of a leaf node n is the set of RVs over which ϕ_n is defined
- the scope of an inner node n is defined as $\text{sc}(n) = \bigcup_{c \in \text{ch}(n)} \text{sc}(c)$
  E.g. sc(n) $= \{X_3, X_4, X_5, X_6\}$

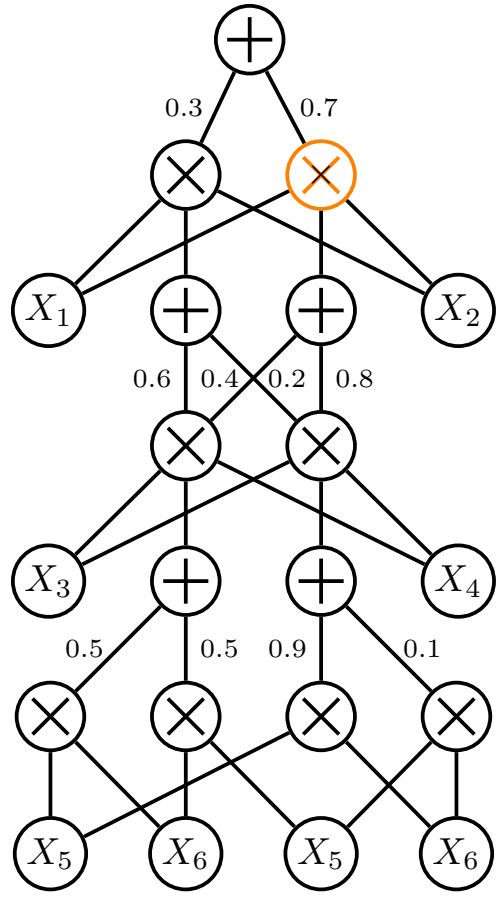


Scopes

The ***scope*** of a node n in S is denoted as

$$\mathsf{sc}(n) \subseteq \mathbf{X}$$

- the scope of a leaf node n is the set of RVs over which ϕ_n is defined
- the scope of an inner node n is defined as $\mathsf{sc}(n) = \bigcup_{c \in \mathsf{ch}(n)} \mathsf{sc}(c)$
  E.g.
  $\mathsf{sc}(\boxed{\mathsf{n}}) = \{X_1, X_2, X_3, X_4, X_5, X_6\}$



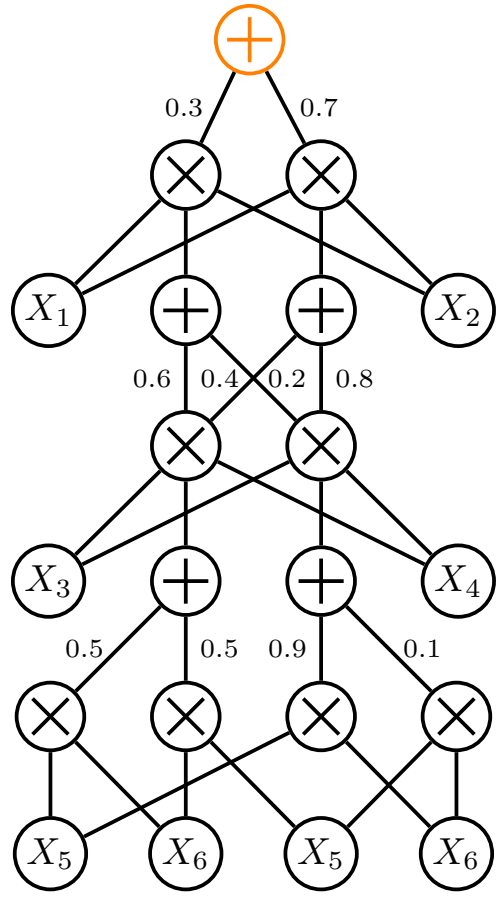
Scopes

The ***scope*** of a node n in S is denoted as

$$\mathsf{sc}(n) \subseteq \mathbf{X}$$

- the scope of a leaf node n is the set of RVs over which ϕ_n is defined
- the scope of an inner node n is defined as $\mathsf{sc}(n) = \bigcup_{c \in \mathsf{ch}(n)} \mathsf{sc}(c)$
- the scope of S is the scope of its root, i.e. $\mathbf{X}$

  E.g.
  $\mathsf{sc}(S) = \mathsf{sc}(\boxed{\mathsf{n}}) =$
  $\{X_1, X_2, X_3, X_4, X_5, X_6\}$



Structural Properties

Let S be an SPN and let $\mathbf{S}_{\oplus}$ (resp. $\mathbf{S}_{\otimes}$) be the set of all sum (resp. product) nodes in S

1. S is ***complete*** iff $\forall n \in \mathbf{S}_{\oplus}, \forall c_1, c_2 \in \mathsf{ch}(n) : \mathsf{sc}(c_1) = \mathsf{sc}(c_2)$
2. S is ***decomposable*** iff
   $\forall n \in \mathbf{S}_{\otimes}, \forall c_1, c_2 \in \mathsf{ch}(n), c_1 \neq c_2 : \mathsf{sc}(c_1) \cap \mathsf{sc}(c_2) = \emptyset$
3. If S is complete and decomposable, then it is ***valid*** [5][6]

Evaluating a valid network corresponds to evaluate a joint unnormalized probability distribution $p_{\mathbf{X}}$: $\forall \mathbf{x}, S(\mathbf{x})/Z = p(\mathbf{X} = \mathbf{x})$

- Z being the normalizing *partition* function $Z = \sum_{\mathbf{x} \sim \mathbf{X}} S(\mathbf{x})$

Valid SPN correctly compiles the *extended network polynomial* encoding the distribution $p_{\mathbf{X}}$ [7].

---

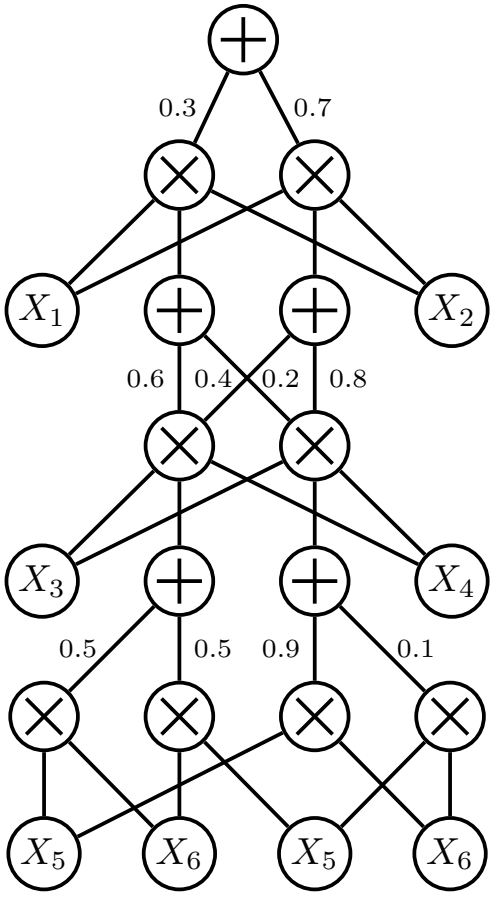
[5] Darwiche, ***Modeling and Reasoning with Bayesian Networks***, 2009

[6] Poon and Domingos, "Sum-Product Networks: a New Deep Architecture", 2011

[7] Peharz et al., "On Theoretical Properties of Sum-Product Networks", 2015

Inference

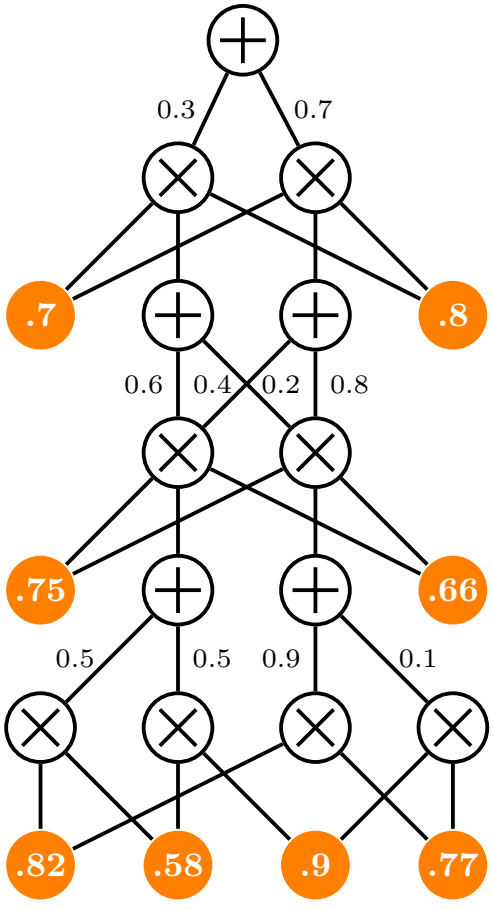
Complete evidence inference



To compute $p(\mathbf{X} = \mathbf{x})$, evaluate S in a bottom-up (feedforward) fashion.

Each node n, compute $S_n(\mathbf{x}_{|\mathsf{sc}(n)}) = S_n(\mathbf{x})$:

Complete evidence inference

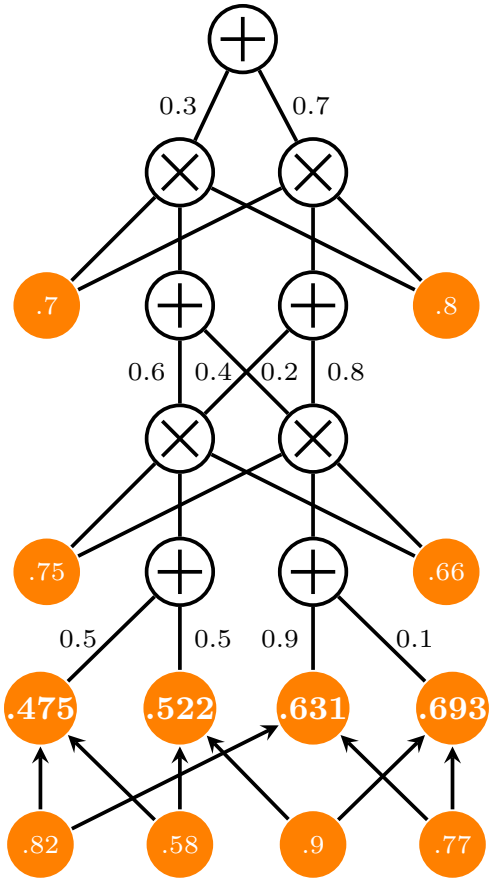


To compute $p(\mathbf{X} = \mathbf{x})$, evaluate S in a bottom-up (feedforward) fashion.

Each node n, compute $S_n(\mathbf{x}_{|\mathsf{sc}(n)}) = S_n(\mathbf{x})$:

- $S_n(\mathbf{x}) = \phi_n(\mathsf{sc}(n) = \mathbf{x}_{|\mathsf{sc}(n)})$
  if n is a leaf node

Complete evidence inference

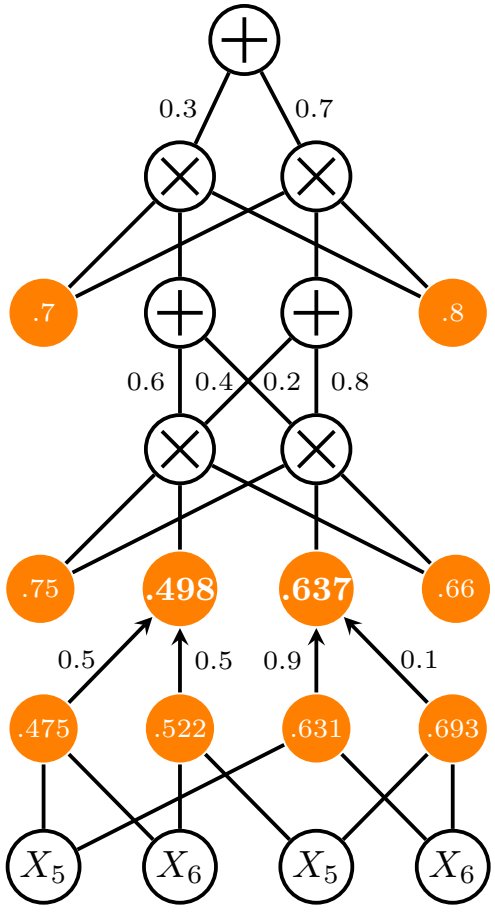


To compute $p(\mathbf{X} = \mathbf{x})$, evaluate S in a bottom-up (feedforward) fashion.

Each node n, compute $S_n(\mathbf{x}_{|\mathsf{sc}(n)}) = S_n(\mathbf{x})$:

- $S_n(\mathbf{x}) = \phi_n(\mathsf{sc}(n) = \mathbf{x}_{|\mathsf{sc}(n)})$
  if n is a leaf node
- $S_n(\mathbf{x}) = \prod_{c \in \mathsf{ch}(n)} S_c(\mathbf{x})$
  if n is a product node

Complete evidence inference

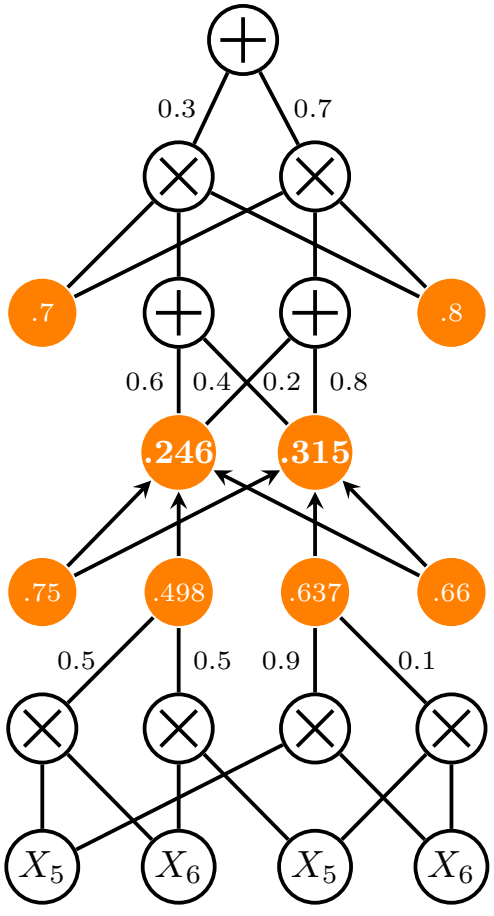


To compute $p(\mathbf{X} = \mathbf{x})$, evaluate S in a bottom-up (feedforward) fashion.

Each node n, compute $S_n(\mathbf{x}_{|\mathsf{sc}(n)}) = S_n(\mathbf{x})$:

- $S_n(\mathbf{x}) = \phi_n(\mathsf{sc}(n) = \mathbf{x}_{|\mathsf{sc}(n)})$
  if n is a leaf node
- $S_n(\mathbf{x}) = \prod_{c \in \mathsf{ch}(n)} S_c(\mathbf{x})$
  if n is a product node
- $S_n(\mathbf{x}) = \sum_{c \in \mathsf{ch}(n)} w_{nc} S_c(\mathbf{x})$
  if n is a sum node

Complete evidence inference

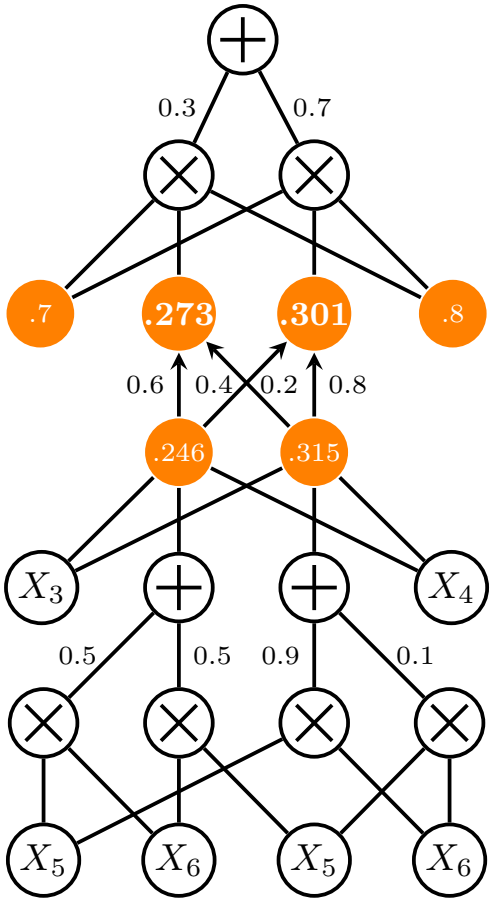


To compute $p(\mathbf{X} = \mathbf{x})$, evaluate S in a bottom-up (feedforward) fashion.

Each node n, compute $S_n(\mathbf{x}_{|\mathsf{sc}(n)}) = S_n(\mathbf{x})$:

- $S_n(\mathbf{x}) = \phi_n(\mathsf{sc}(n) = \mathbf{x}_{|\mathsf{sc}(n)})$
  if n is a leaf node
- $S_n(\mathbf{x}) = \prod_{c \in \mathsf{ch}(n)} S_c(\mathbf{x})$
  if n is a product node
- $S_n(\mathbf{x}) = \sum_{c \in \mathsf{ch}(n)} w_{nc} S_c(\mathbf{x})$
  if n is a sum node

Complete evidence inference

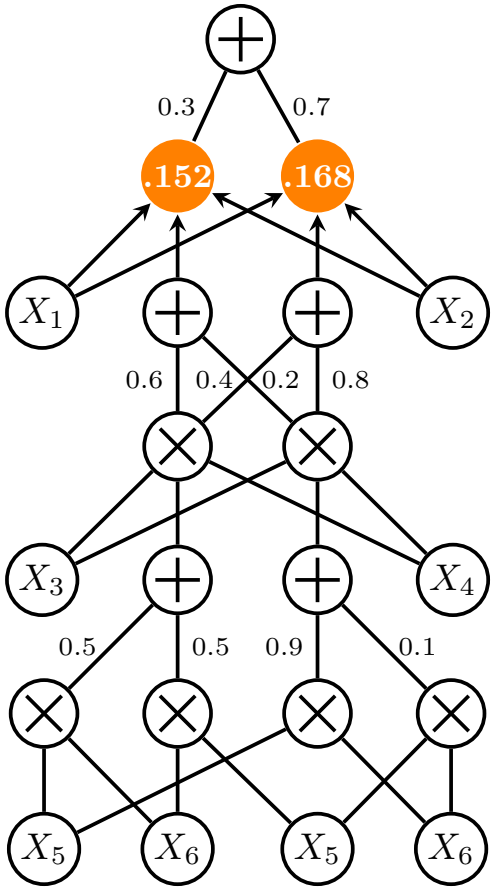


To compute $p(\mathbf{X} = \mathbf{x})$, evaluate S in a bottom-up (feedforward) fashion.

Each node n, compute $S_n(\mathbf{x}_{|\mathsf{sc}(n)}) = S_n(\mathbf{x})$:

- $S_n(\mathbf{x}) = \phi_n(\mathsf{sc}(n) = \mathbf{x}_{|\mathsf{sc}(n)})$
  if n is a leaf node
- $S_n(\mathbf{x}) = \prod_{c \in \mathsf{ch}(n)} S_c(\mathbf{x})$
  if n is a product node
- $S_n(\mathbf{x}) = \sum_{c \in \mathsf{ch}(n)} w_{nc} S_c(\mathbf{x})$
  if n is a sum node

Complete evidence inference

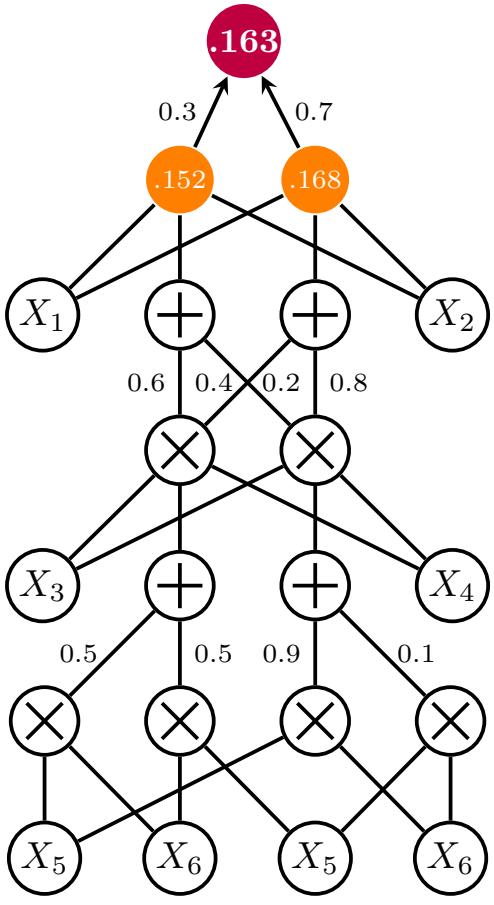


To compute $p(\mathbf{X} = \mathbf{x})$, evaluate S in a bottom-up (feedforward) fashion.

Each node n, compute $S_n(\mathbf{x}_{|\mathsf{sc}(n)}) = S_n(\mathbf{x})$:

- $S_n(\mathbf{x}) = \phi_n(\mathsf{sc}(n) = \mathbf{x}_{|\mathsf{sc}(n)})$
  if n is a leaf node
- $S_n(\mathbf{x}) = \prod_{c \in \mathsf{ch}(n)} S_c(\mathbf{x})$
  if n is a product node
- $S_n(\mathbf{x}) = \sum_{c \in \mathsf{ch}(n)} w_{nc} S_c(\mathbf{x})$
  if n is a sum node

Complete evidence inference



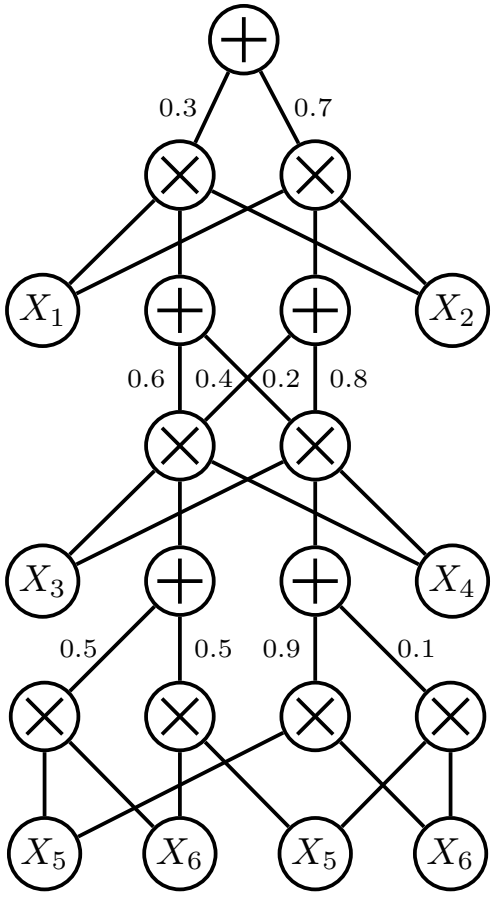
To compute $p(\mathbf{X} = \mathbf{x})$, evaluate S in a bottom-up (feedforward) fashion.

Each node n, compute $S_n(\mathbf{x}_{|\mathsf{sc}(n)}) = S_n(\mathbf{x})$:

- $S_n(\mathbf{x}) = \phi_n(\mathsf{sc}(n) = \mathbf{x}_{|\mathsf{sc}(n)})$
  if n is a leaf node
- $S_n(\mathbf{x}) = \prod_{c \in \mathsf{ch}(n)} S_c(\mathbf{x})$
  if n is a product node
- $S_n(\mathbf{x}) = \sum_{c \in \mathsf{ch}(n)} w_{nc} S_c(\mathbf{x})$
  if n is a sum node

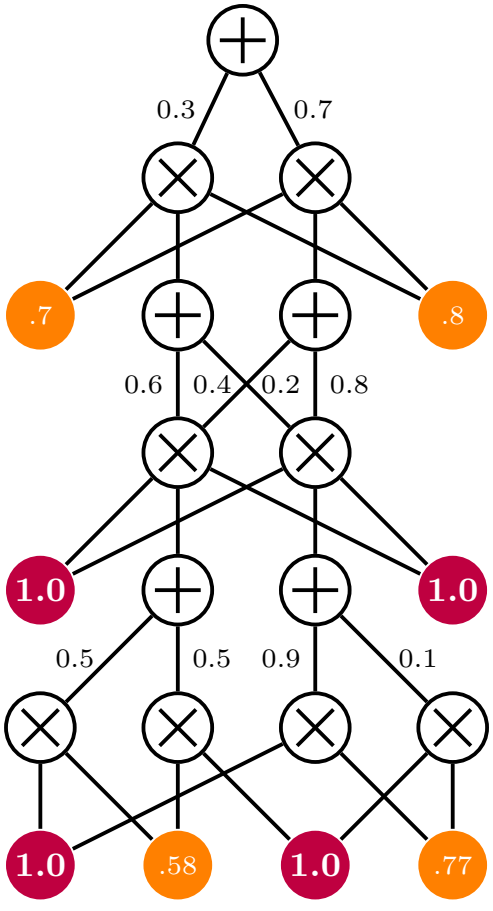
the root output is $S(\mathbf{x}) = p(\mathbf{X} = \mathbf{x})$

Marginal inference



To compute a marginal query like $p(\mathbf{Q} = \mathbf{q}), \mathbf{Q} \subset \mathbf{X}$ evaluate S as before (feedforward)

Marginal inference

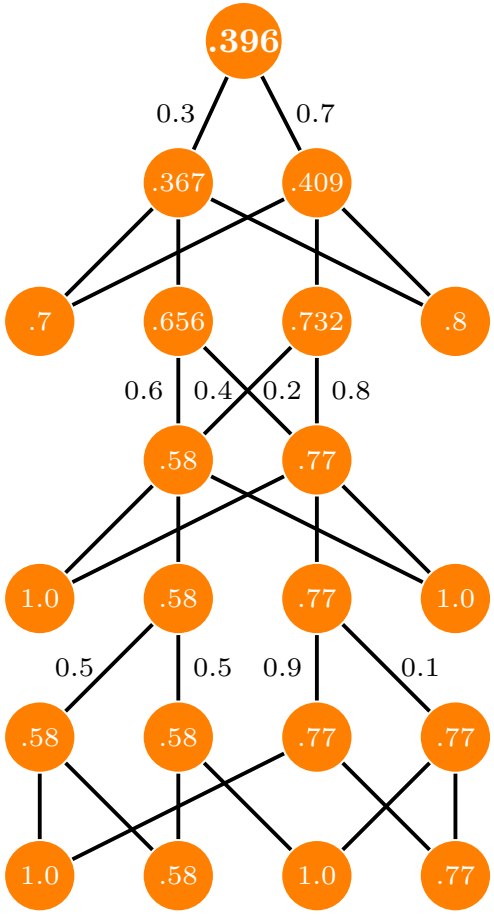


To compute a marginal query like $p(\mathbf{Q} = \mathbf{q}), \mathbf{Q} \subset \mathbf{X}$ evaluate S as before (feedforward)

but evaluate a leaf n as:

$$S_n(\mathbf{q}) = \begin{cases} p(\text{sc}(n) = \mathbf{q}_{|\text{sc}(n)}) & \text{if sc}(n) \subseteq \mathbf{Q} \\ 1.0 & \text{otherwise} \end{cases}$$

Marginal inference



To compute a marginal query like $p(\mathbf{Q} = \mathbf{q}), \mathbf{Q} \subset \mathbf{X}$ evaluate S as before (feedforward)

but evaluate a leaf n as:

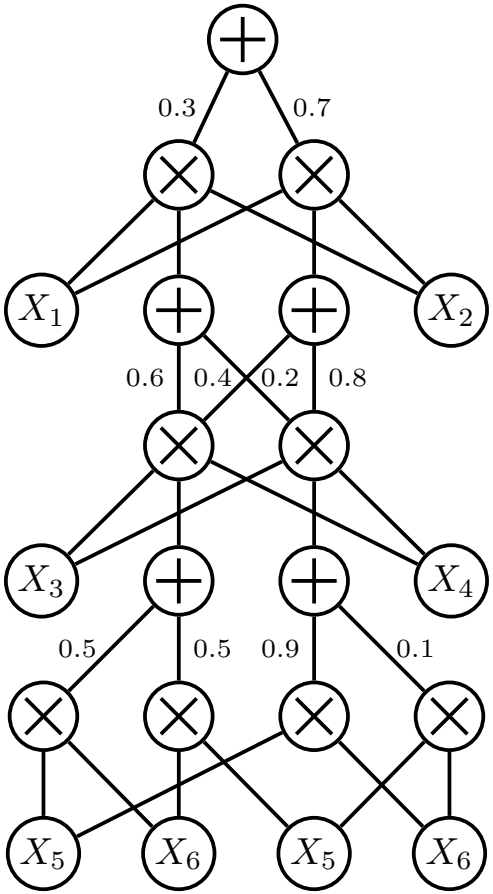
$$S_n(\mathbf{q}) = \begin{cases} p(\text{sc}(n) = \mathbf{q}_{|\text{sc}(n)}) & \text{if sc}(n) \subseteq \mathbf{Q} \\ 1.0 & \text{otherwise} \end{cases}$$

then propagate as before

- each sub-network shall output 1 as the probability of marginalizing over all the RVs out of its scope

Conditionals are tractable as well:
$p(\mathbf{Q}|\mathbf{E}) = p(\mathbf{Q}, \mathbf{E})/p(\mathbf{E})$

MPE inference[8]



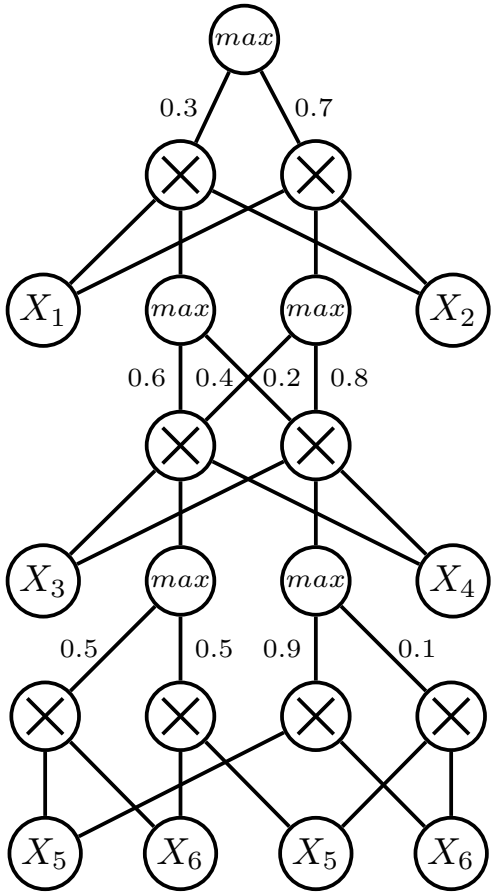
An approximation of MPE inference can be answered in linear time as well

$$\mathbf{q}^* = \underset{\mathbf{q} \sim \mathbf{Q}}{\operatorname{argmax}}\, p(\mathbf{E}, \mathbf{q})$$

for some RVs $\mathbf{E}, \mathbf{Q} \subset \mathbf{X}, \mathbf{E} \cap \mathbf{Q} = \emptyset$, $\mathbf{E} \cup \mathbf{Q} = \mathbf{X}$

[8] Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", 2016

MPE inference[8]



An approximation of MPE inference can be answered in linear time as well.

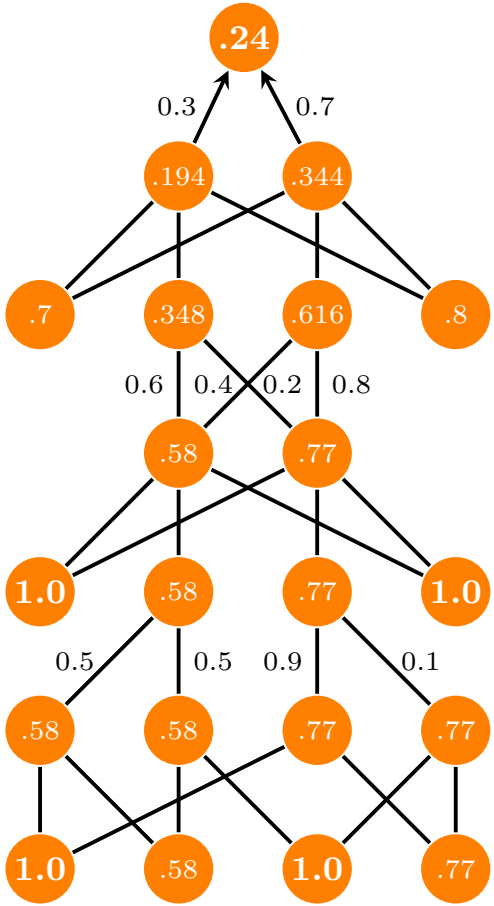
eg: $\mathbf{Q} = \{X_3, X_4, X_5\}, \mathbf{E} = \{X_1, X_2, X_6\}$

- build a ***Max-Product Network*** M substituting each $n \in \mathbf{S}_{\oplus}$ for a **max** node computing

$$\max_{c \in \mathsf{ch}(n)} w_{nc} M_n$$

[8] Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", 2016

MPE inference[8]



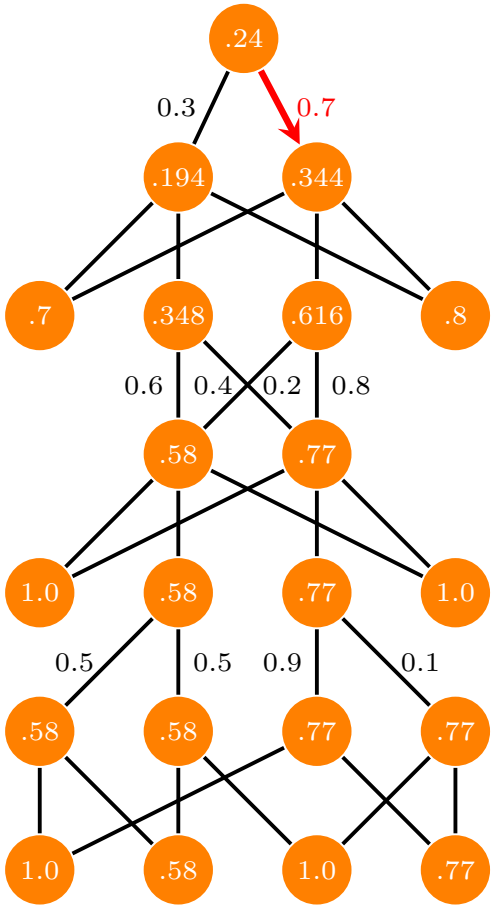
An approximation of MPE inference can be answered in linear time as well.

eg: $\mathbf{Q} = \{X_3, X_4, X_5\}, \mathbf{E} = \{X_1, X_2, X_6\}$

- build a ***Max-Product Network*** M
- evaluate M bottom-up after setting all leaves n, $\text{sc}(n) \subseteq \mathbf{Q}$ to 1

[8] Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", 2016

MPE inference[8]



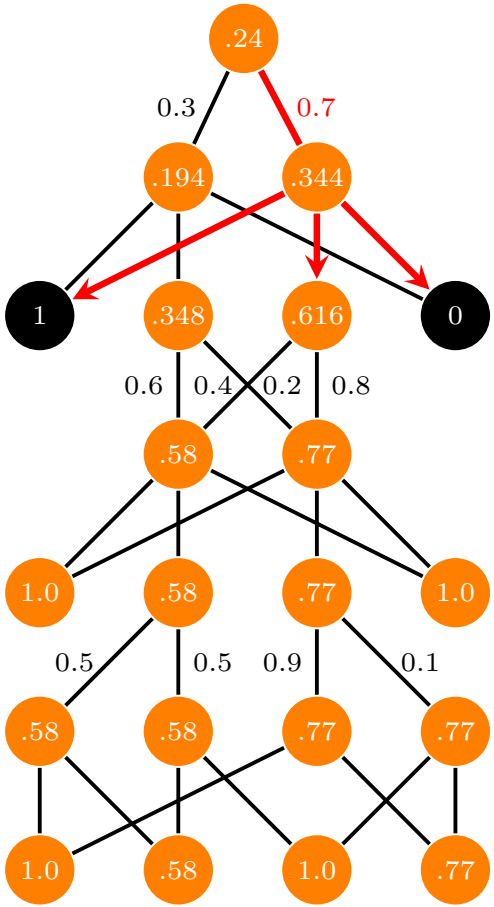
An approximation of MPE inference can be answered in linear time as well.

eg: $\mathbf{Q} = \{X_3, X_4, X_5\}, \mathbf{E} = \{X_1, X_2, X_6\}$

- build a ***Max-Product Network*** M
- evaluate M bottom-up after setting all leaves n, $\text{sc}(n) \subseteq \mathbf{Q}$ to 1
- a top-down traversal traces back the MPE assignment for each RV in $\mathbf{Q}$, following:
  - only the max output child branch of a max node

[8] Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", 2016

MPE inference[8]



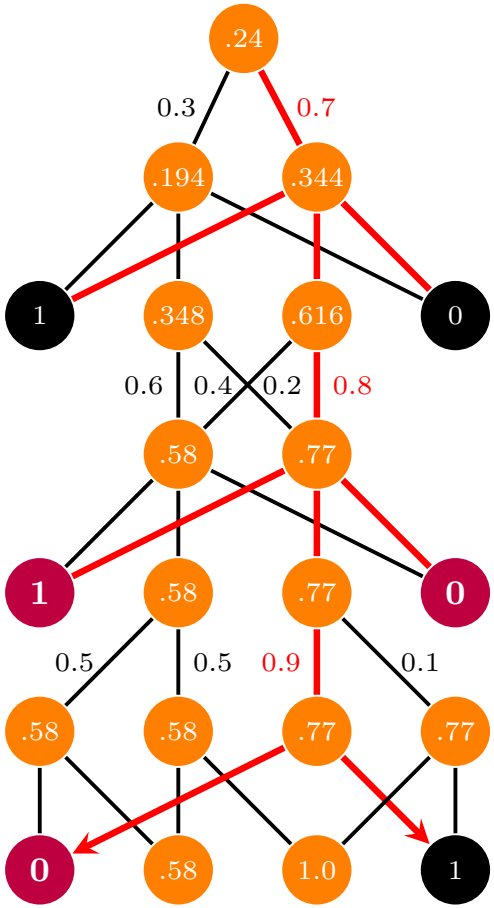
An approximation of MPE inference can be answered in linear time as well.

eg: $\mathbf{Q} = \{X_3, X_4, X_5\}, \mathbf{E} = \{X_1, X_2\}, X_6$

- build a ***Max-Product Network*** M
- evaluate M bottom-up after setting all leaves n, $\text{sc}(n) \subseteq \mathbf{Q}$ to 1
- a top-down traversal traces back the MPE assignment for each RV in $\mathbf{Q}$, following:
  - only the max output child branch of a max node
  - all child branches of product nodes

[8] Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", 2016

MPE inference[8]



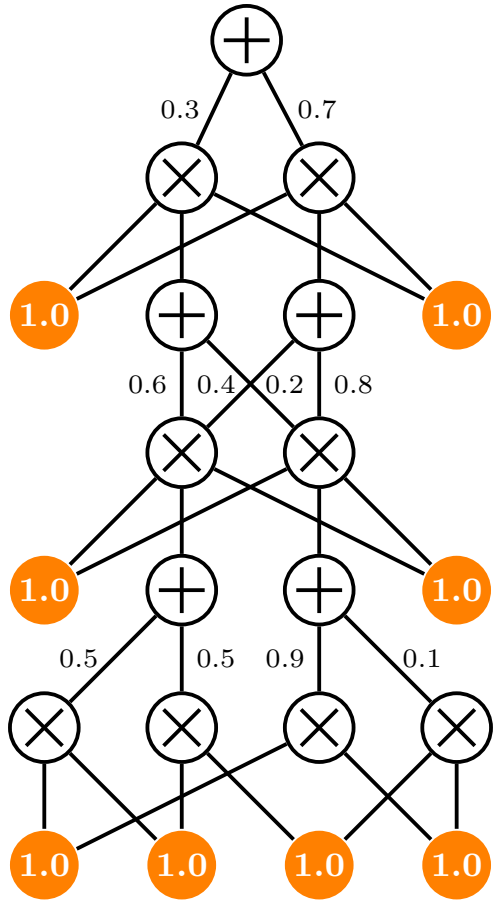
An approximation of MPE inference can be answered in linear time as well.

eg: $\mathbf{Q} = \{ X_3, X_4, X_5 \}, \mathbf{E} = \{ X_1, X_2, X_6 \}$

- build a ***Max-Product Network*** M
- evaluate M bottom-up after setting all leaves n, $\text{sc}(n) \subseteq \mathbf{Q}$ to 1
- a top-down traversal traces back the MPE assignment for each RV in $\mathbf{Q}$, following:
  - only the max output child branch of a max node
  - all child branches of product nodes
- determining a **path** whose leaves union forms the MPE assignment

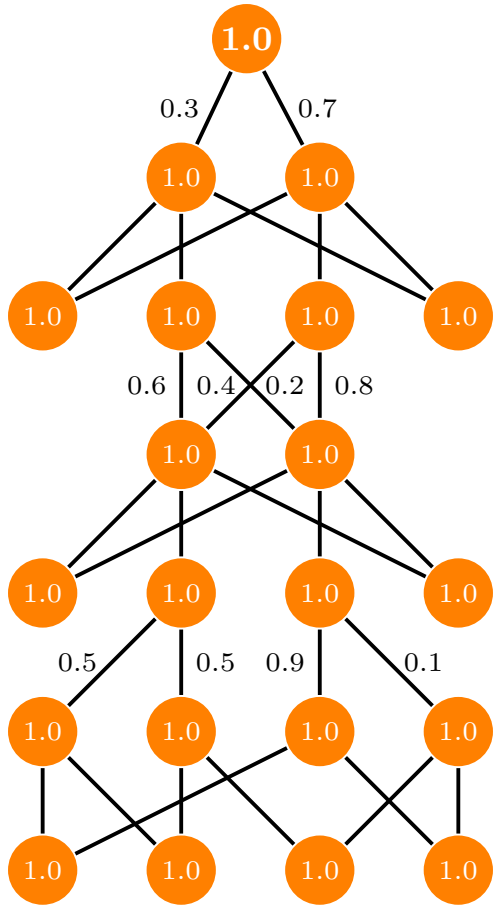
[8] Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", 2016

Partition function computation



As for ACs, setting all leaf outputs to 1 equals to compute the *partition function*

Partition function computation



As for ACs, setting all leaf outputs to 1 equals to compute the *partition function*

Interpretation

SPN interpretations

Probabilistic model

- sum nodes in a valid network are probabilistic mixtures over their children distributions whose coefficients are the children weights
  - a categorical latent RV H_n can be associated to each sum node n, having values in $\{1, \ldots, |\mathsf{ch}(n)|\}$
  - the weights of a sum node n can also be interpreted as the probabilities of choosing the corresponding child branch from node n, having already taken the path from the root up to n
- since product nodes are evaluated as product of probability values, they identify factorizations over independent distributions

Deep feedforward neural network

- SPNs can also be interpreted as a particular kind of feedforward deep Neural Networks (NNs) with nonnegative parameters, where the leaf distributions are input neurons whereas sum and product nodes are the hidden neurons

SPNs and other models

- SPNs are more general than both hierarchical mixture models and thin junction trees
  - SPNs can be exponentially more compact (distribution over states of variables with an even number of 1's, for instance)
- SPNs are not classical PGMs
  - they are computational graphs, inference machines,...
- SPNs are not "probabilistic, general-purpose convolutional networks, with average-pooling corresponding to marginal inference and max-pooling corresponding to MPE inference"

Network Polynomials

Let $\Phi(x) \geq 0$ an unnormalized probability distribution on Boolean variables. x (resp. $\bar{x}$) denotes the indicator function $[x]$ (resp. $[\bar{x}]$) for the variable X. The network polinomial[9] of $\Phi(x)$ is $\sum_x \Phi(x)\Pi(x)$

- $\Pi(x)$ is the product of the indicators that have value 1 in state x

Example (Bernoulli distribution over X with parameter p)

$$px + (1-p)\bar{x}$$

Example (Bayes Network $X_1 \to X_2$)

$$\theta_{x_1}\theta_{x_2|x_1}x_1x_2 + \theta_{x_1}\theta_{\bar{x}_2|x_1}x_1\bar{x}_2 + \theta_{\bar{x}_1}\theta_{x_2|\bar{x}_1}\bar{x}_1x_2 + \theta_{\bar{x}_1}\theta_{\bar{x}_2|\bar{x}_1}\bar{x}_1\bar{x}_2$$

with $\theta_\cdot = P(\cdot)$

[9] Darwiche, "A Differential Approach to Inference in Bayesian Networks", 2003

Network Polynomials (II)

- The network polynomial is a multilinear function of the indicator variables
- The unnormalized probability of evidence e (partial instantiation of X) is the value of the network polynomial when all indicators compatible with e are set to 1 and the remainder are set to 0

Example

$\Phi(X_1 = 1, X_3 = 0)$ is the value of the network polynomial when $\bar{x}_1$ and x_3 are set to 0 and the remaining indicators are set to 1 throughout.

- The partition function is the value of the network polynomial when all indicators are set to 1
- For any evidence e, the cost of computing $P(e) = \Phi(e)/Z$ is linear in the size of the network polynomial
- The network polynomial has size exponential in the number of variables
- it is possible to represent and evaluate it in polynomial space and time using an AC or an SPN

Arithmetic Circuits

Arithmetic Circuits (ACs)[10]: inference representation closely related to SPNs

- a rooted DAG with sums and products as interior nodes
- indicator nodes and parameters as leaves

Properties

- ***decomposable***: children of a product node have disjoint scopes
- ***smooth***: children of a sum node have identical scopes
- ***deterministic***: children of a sum node are mutually exclusive
  - at most one is non-zero for any complete configuration

An AC represents a valid probability distribution if it is decomposable and smooth

ACs generated by compiling graphical models are typically deterministic as well

- while for SPNs sum nodes represent mixtures of distributions and are not deterministic in general

---

[10] **Darwiche, "A Differential Approach to Inference in Bayesian Networks", 2003**

Arithmetic Circuits (II)

ACs and SPNs representations are equivalent for discrete domains[11]

- every decomposable and smooth AC can be represented as an equivalent SPN with fewer or equal nodes and edges
- every SPN can be represented as an AC with at most a linear increase in the number of edges

Learning ACs has been made by

- a standard BN structure learner with the complexity of the resulting circuit as the regularizer[12]
- learning MNs representable by ACs, but does not reusing sub-ACs[13]

SPN learning algorithms emphasize mixtures

- results in models that use implicit latent variables to capture all of the interactions among the observable variables

---

[11] **Rooshenas and Lowd, "Learning Sum-Product Networks with Direct and Indirect Variable Interactions", 2014**

[12] **Lowd and Domingos, "Learning Arithmetic Circuits", 2012**

[13] **Lowd and Rooshenas, "Learning Markov Networks With Arithmetic Circuits", 2013**

Arithmetic Circuits (II)

Differences with SPNs

probabilistic semantics of SPNs

- allows for direct structure learning schemes where the compilation process is implicit
- allows sampling from their encoded distribution (generative model)

no shared weights

- differently from ACs, it is not possible to have the same tied parameter for many nodes in SPNs

generalized SPNs

- instead of using IVs to represent the states of discrete RVs, SPNs have been generalized to continuous RVs and discrete RVs with infinitely many states[14]
  - IVs $\lambda_{X=x}$ are replaced by distributions

[14] Peharz et al., "On Theoretical Properties of Sum-Product Networks", 2015

SPNs as NNs (I)

SPNs are a particular kind of ***labelled***, ***constrained*** and ***fully probabilistic*** neural networks.

Labelled: each neuron is associated a *scope*
Constrained: completeness and decomposability determine network topology.
Fully probabilistic: each valid sub-SPN is still a valid-SPN.

SPNs provide a ***direct encoding*** of the input space into a deep architecture $\rightarrow$ *visualizing representations* (back) into the *input space*.

Vergari et al., "Visualizing and Understanding Sum-Product Networks", 2016

SPNs as NNs (II)

A classic MLP hidden layer computes the function:

$$h(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

SPNs can be reframed as *DAGs* of MLPs, each sum layer computing:

$$\mathbf{S}(\mathbf{x}) = \log(\mathbf{W}\mathbf{x})$$

and product layers computing:

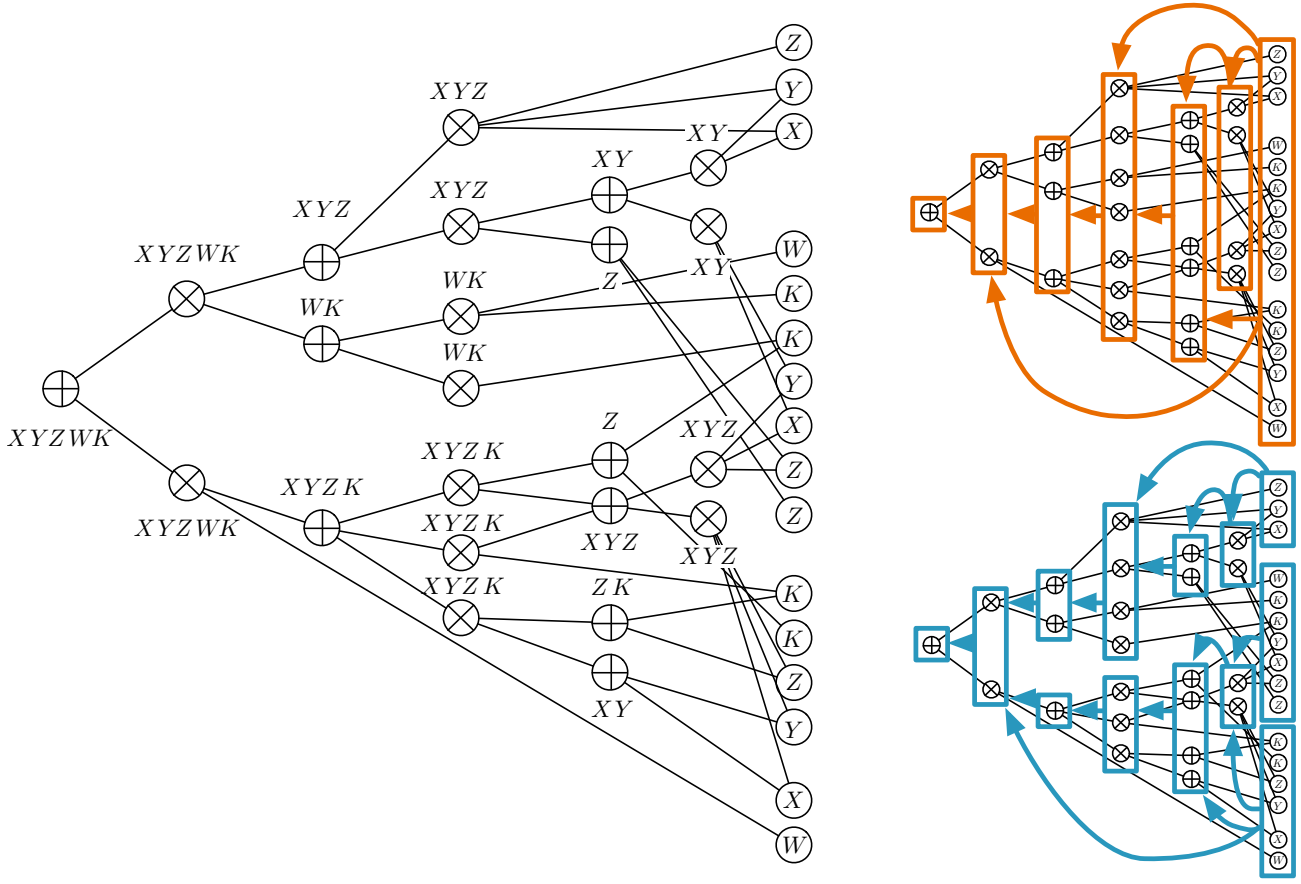
$$\mathbf{S}(\mathbf{x}) = \exp(\mathbf{P}\mathbf{x})$$

where $\mathbf{W} \in \mathbb{R}_{+}^{s \times r}$ and $\mathbf{P} \in \{0, 1\}^{s \times r}$ are the weight matrices:

$$\mathbf{W}_{(ij)} = \begin{cases} w_{ij} & \text{if } i \to j \\ 0 & \text{otherwise} \end{cases} \qquad \mathbf{P}_{(ij)} = \begin{cases} 1 & \text{if } i \to j \\ 0 & \text{otherwise} \end{cases}$$

Vergari et al., "Visualizing and Understanding Sum-Product Networks", 2016

SPNs as NNs (III)



SPNs as NNs (IV): filters

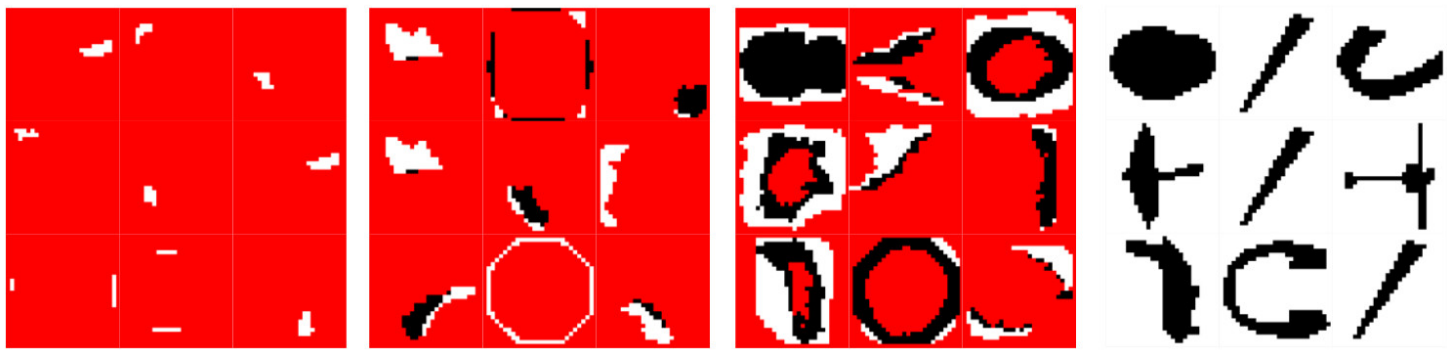
Learned features as images maximizing neuron activations[15]:

$$\mathbf{x}^* = \underset{\mathbf{x}, ||\mathbf{x}||=\gamma}{\operatorname{argmax}} \, h_{ij}(\mathbf{x}; \boldsymbol{\theta}).$$

With SPNs, joint solution as an MPE assignment for all nodes (linear time):

$$\mathbf{x}^*_{|\mathsf{sc}(n)} = \underset{\mathbf{x}}{\operatorname{argmax}} \, S_n(\mathbf{x}_{|\mathsf{sc}(n)}; \mathbf{w})$$

.



$\rightarrow$ *scope length* ($|\mathsf{sc}(n)|$) correlates with feature abstraction level

---

[15] Erhan et al., "Visualizing Higher-Layer Features of a Deep Network", 2009
Vergari et al., "Visualizing and Understanding Sum-Product Networks", 2016

SPNs as BNs

Adopting Algebraic Decision Diagrams (ADDs) for CPDs, every SPN can be converted into a BN in linear time and space complexity in the size of the SPN

- the generated BN has a simple bipartite structure
- applying the VE algorithm to the generated BN with ADD representation of its CPDs, the original SPN can be recovered in linear time and space with respect to the size of the SPN
  - the SPN can be viewed as a caching of the VE inference process

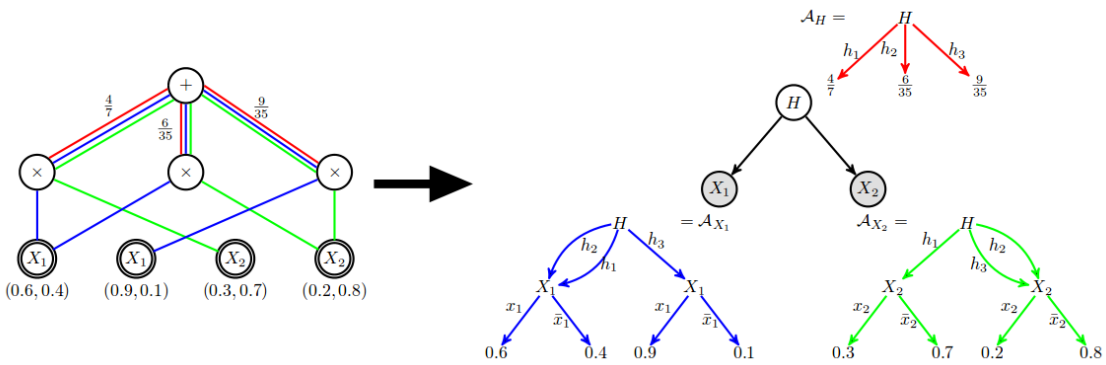
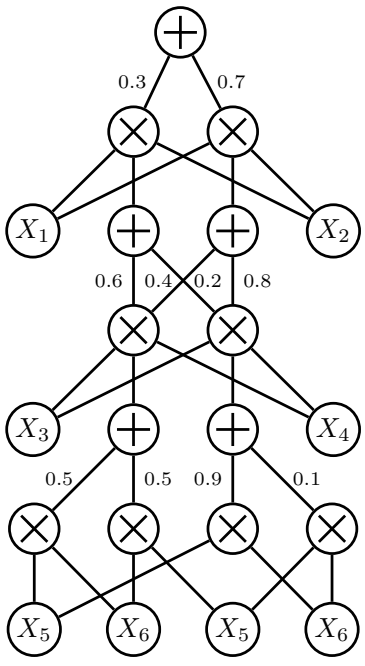
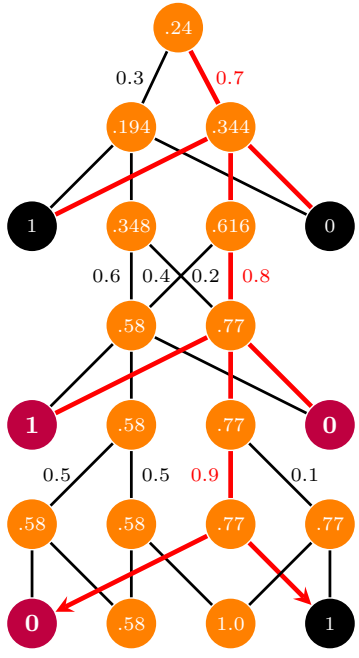


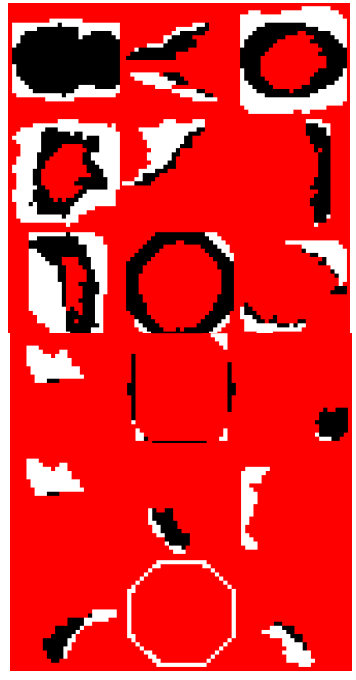
Figure from [16]. Construct a BN with CPDs represented by ADDs from an SPN.

[16] Zhao et al., "On the Relationship between Sum-Product Networks and Bayesian Networks", 2015

Discuss


0.3
0.7
X_1
X_2
0.6
0.4
0.2
0.8
X_3
X_4
0.5
0.5
0.9
0.1
X_5
X_6
X_5
X_6


.24
0.3
0.7
.194
.344
1
.348
.616
0
0.6
0.4
0.2
0.8
.58
.77
1
.58
.77
0
0.5
0.5
0.9
0.1
.58
.58
.77
.77
0
.58
1.0
1



Learning

Learning SPNs

Parameter learning: estimate $\mathbf{w}$ from data considering an SPN as a latent RV model, or as a NN.

Structure learning: build the network from data by assigning scores to tentative structures or by exploiting constraints

How to learn a "full" SPN:

- handcrafted structure, then parameter learning **[Poon and Domingos 2011] [Gens and Domingos 2012]**
- random structures, then parameter learning **[Rashwan et al. 2016]**
- structure learning, then parameter learning (fine tuning) **[Zhao et al. 2016]**
- learn both weight and structure at the same time **[Adel et al. 2015; Gens and Domingos 2013; Rooshenas and Lowd 2014; Vergari et al. 2015]** ...

Structure Learning

Score vs constraint based search. No closed form for likelihood scores, need heuristics **[Rooshenas and Lowd 2014].**
No need for it by exploiting the inner nodes probabilistic semantics

Learning graph vs tree structures: Easier to learn a tree SPN (sometimes SPT) with greedy approaches. Graph SPNs may be more compact and expressive efficient.

Top-down vs bottom-up approaches: iteratively cluster data matrix (top-down) or start by the marginal RVs (bottom-up)

LearnSPN is a ***greedy***, ***top down***, ***constraint based*** learner for ***tree*** SPNs **[Gens and Domingos 2013]**
$\rightarrow$ *First principled top-down learner, inspired many algorithms and variations*
$\rightarrow$ *Surprisingly simple and accurate*

LearnSPN (I)

Build a tree SPN by recursively split the data matrix:

- splitting columns into pairs by a greedy ***G Test*** with threshold ρ:

$$G(X_i, X_j) = 2 \sum_{x_i \sim X_i} \sum_{x_j \sim X_j} c(x_i, x_j) \cdot \log \frac{c(x_i, x_j) \cdot |T|}{c(x_i) c(x_j)}$$

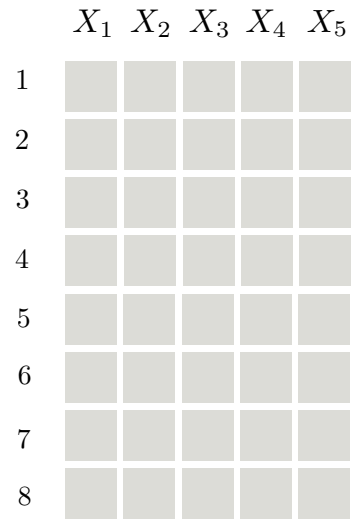
- clustering instances into $|C|$ sets with ***online Hard-EM***, estimating weights as cluster proportions with cluster penalty λ

$$p(\mathbf{X}) = \sum_{C_i \in \mathbf{C}} \prod_{X_j \in \mathbf{X}} p(X_j | C_i) p(C_i)$$

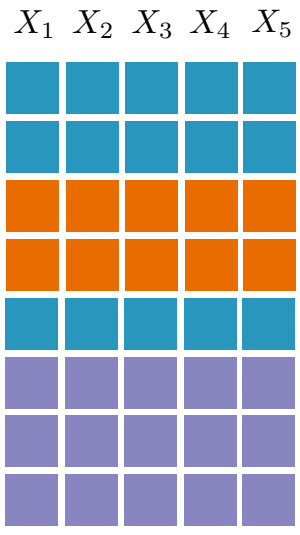
- if there are less than m instances, put a ***naive factorization*** over leaves
- each univariate distribution get ***ML estimation*** smoothed by α

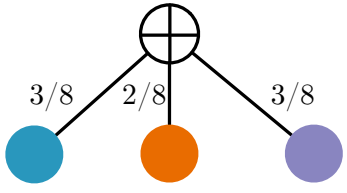
Hyperparameter space: $\{\rho, \lambda, m, \alpha\}$.

LearnSPN (II)

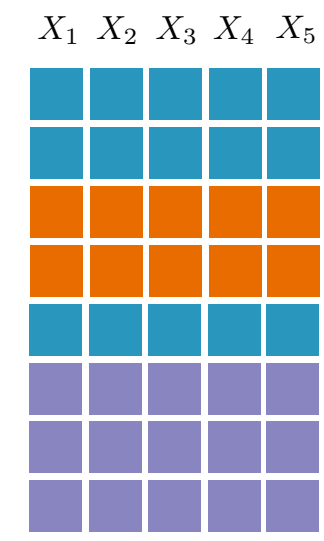


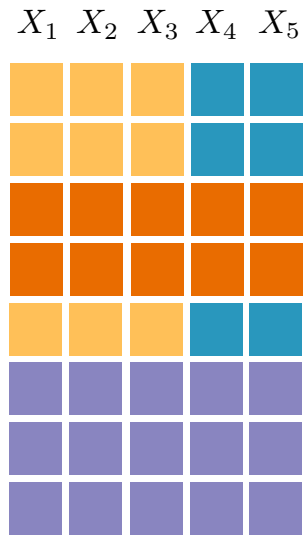
LearnSPN (II)

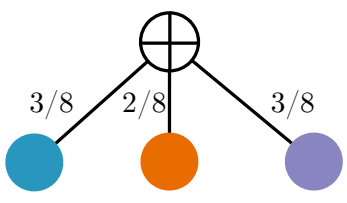


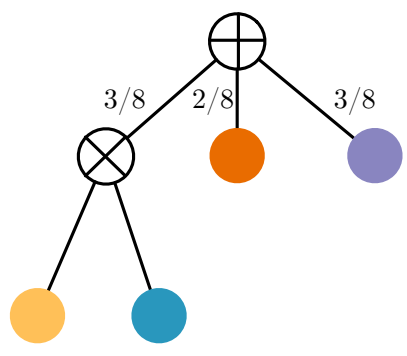


LearnSPN (II)

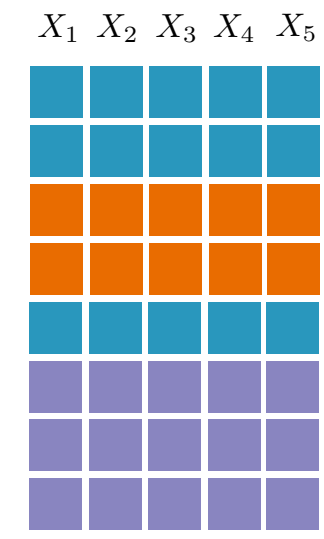


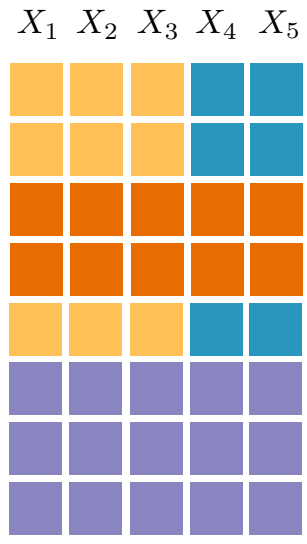


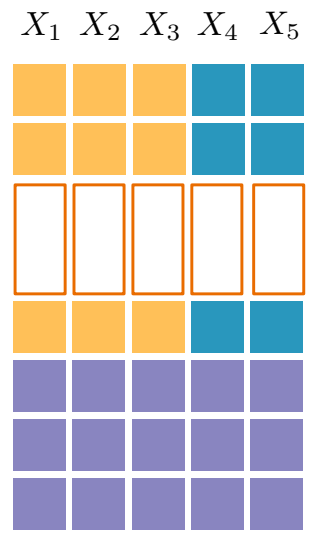


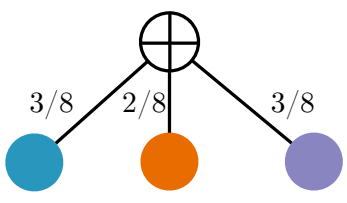


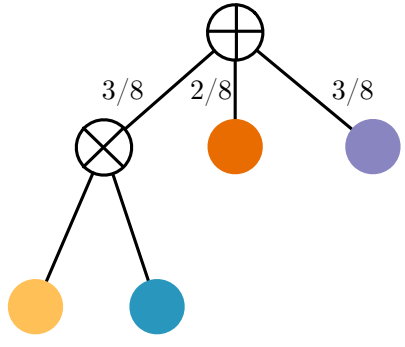
LearnSPN (II)

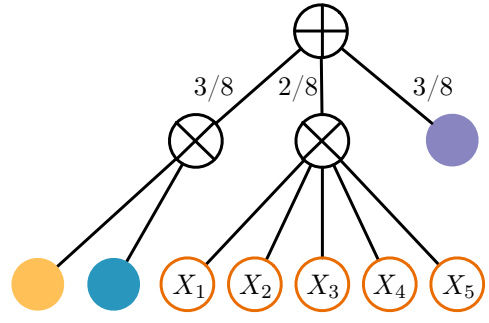

X_1 X_2 X_3 X_4 X_5


X_1 X_2 X_3 X_4 X_5


X_1 X_2 X_3 X_4 X_5


3/8
2/8
3/8


3/8
2/8
3/8


3/8
2/8
3/8
X_1
X_2
X_3
X_4
X_5

Tweaking LearnSPN

LearnSPN performs two interleaved greedy ***hierarchical divisive clustering*** processes (co-clutering on the data matrix).

Fast and simple. But both processes never look back and are committed to the choices they take $\rightarrow$ *slow down the two processes*

Online EM does not need to specify the number of clusters k in advance. But overcomplex structures are learned by exploding the number of sum node children $\rightarrow$ *look for deeper networks*

Tractable leaf estimation. But too strong naive factorization independence assumptions, hard to regularize $\rightarrow$ *learn tree distributions as leaves*

ML estimations are effective. But they are not robust to noise, they can overfit the training set easily $\rightarrow$ *learn bagged sum nodes*

Why Structure Quality Matters

Tractable inference is guaranteed *if the network size is polynomial* in $|\mathbf{X}|$.

Network ***size*** influences inference complexity: smaller networks, faster inference!
$\rightarrow$ Comparing network sizes is better than comparing inference times

Network ***depth*** influences *expressive efficiency* **[Martens and Medabalimi 2014] [Zhao et al. 2015]**

Structural simplicity as a bias: overcomplex networks may not generalize well.

Structure quality desiderata: **smaller** but **accurate**, **deeper** but not wider, SPNs.

LearnSPN-b

Observation: each clustering process benefits from the other one improvements/highly suffers from other's mistakes.

Idea: slow them down the processes by limiting the number of nodes to split to the minimum. LearnSPN-b, binary splitting $k = 2$.

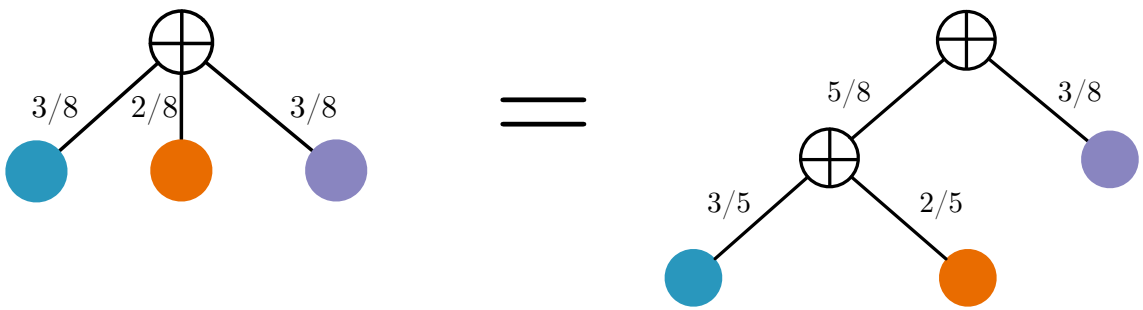
$\rightarrow$ *one hyperparameter less,* λ.

$\rightarrow$ *not committing to complex structures too early*

$\rightarrow$ *reducing node out fan increases the depth*

$\rightarrow$ *same expressive power as LearnSPN structures*

$\rightarrow$ *statistically same (or better) accuracy, smaller networks*



LearnSPN-b: depth VS size

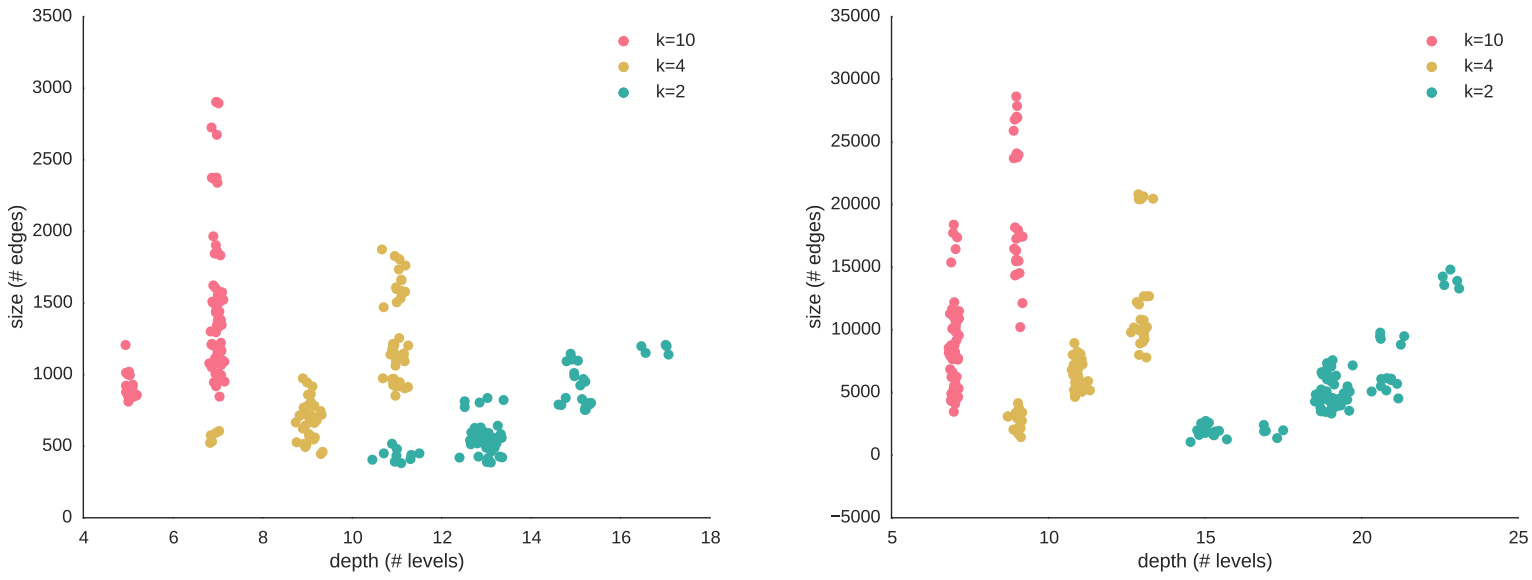


Figure : Network sizes VS depths while varying the max number of sum node children splits ($k \in \{10, 4, 2\}$). Each dot is an experiment in the grid search hyperparameter space performed by LearnSPN-b on NLTCS (left) and Plants (right).

LearnSPN-b: best ll VS size

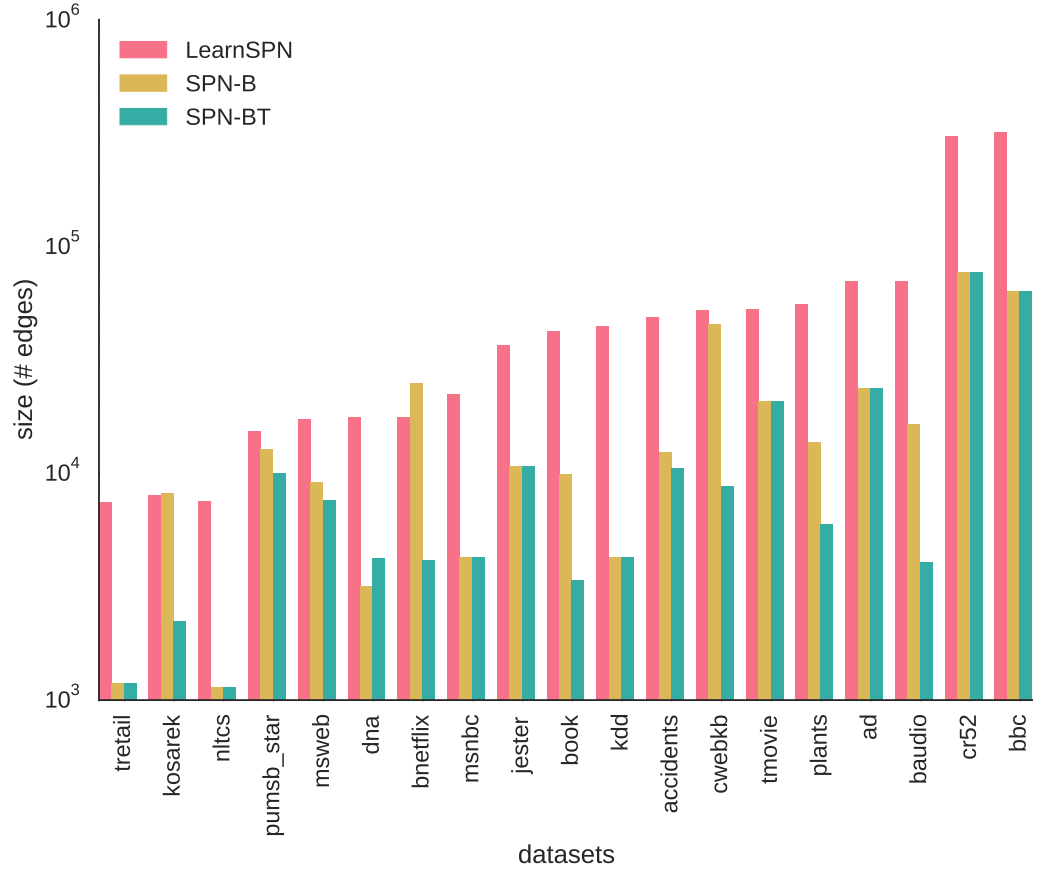


Figure : Comparing network sizes for the networks scoring the best log-likelihoods in the grid search as obtained by LearnSPN, LearnSPN-b and LearnSPN-bT for each dataset.

Other variations on LearnSPN

ACs modeling leaves by performing a greedy score search. ***ID-SPN*** best log-likelihood learner (but lots of hyperparameters).
Freely available in the Libra[17] toolkit **[Rooshenas and Lowd 2014]**

Looking for **correlations instead of independencies** via matrix factorizations.
Splitting matrix rows and columns at the same time: SPN-SVD.
It can cope with continuous data **[Adel et al. 2015]**

Post-learning **mergining sub-SPNs** that model "similar" distributions.
Reducing network sizes **[Rahman and Gogate 2016].**

Learning Relational SPNs on **first order data** represented in Tractable Markov Logic (TML), *LearnRSPN* **[Nath and Domingos 2015].**

[17] http://libra.cs.uoregon.edu/

Other Tendencies in Structure Learning

Learning deterministic structures which enable closed form log-likelihood and weight estimation.
Selective SPNs, enabling efficient Stochastic Local Search **[Peharz et al. 2014a].**
Mixing latent and deterministic mixtures as sum nodes (a Cutset Network is an SPN!) **[Rahman and Gogate 2016]**

Learning DAGs structures instead of trees.
Substituting sub-structures with more complex ones by cloning mixtures **[Dennis and Ventura 2015]**

Template learning for sequence models. Stochastic local search over well defined constrained structures. Dynamic SPNs **[Melibari et al. 2016]**→ *PGM'16!*

Parameter Learning

Non convex optimization, solvable with (online) iterative methods (e.g. SGD)

Classical approach: compute the gradient $\nabla_{\mathbf{w}} S(\mathbf{x})$
$\rightarrow$ use backpropagation (differential approach[18])

1. $\nabla_{S(\mathbf{x})} S(\mathbf{x}) \leftarrow 1$ start from the root
2. if n is a sum node, $\forall_{c \in \mathsf{ch}(n)}$:
   $\nabla_{S_c(\mathbf{x})} S(\mathbf{x}) \leftarrow \nabla_{S_c(\mathbf{x})} S(\mathbf{x}) + w_{nc} \nabla_{S_n(\mathbf{x})} S(\mathbf{x})$
3. if n is a product node, $\forall_{c \in \mathsf{ch}(n)}$:
   $\nabla_{S_c(\mathbf{x})} S(\mathbf{x}) \leftarrow \nabla_{S_c(\mathbf{x})} S(\mathbf{x}) + \nabla_{S_n(\mathbf{x})} S(\mathbf{x}) \prod_{k \in \mathsf{ch}(n) \setminus \{c\}} S_k(\mathbf{x})$

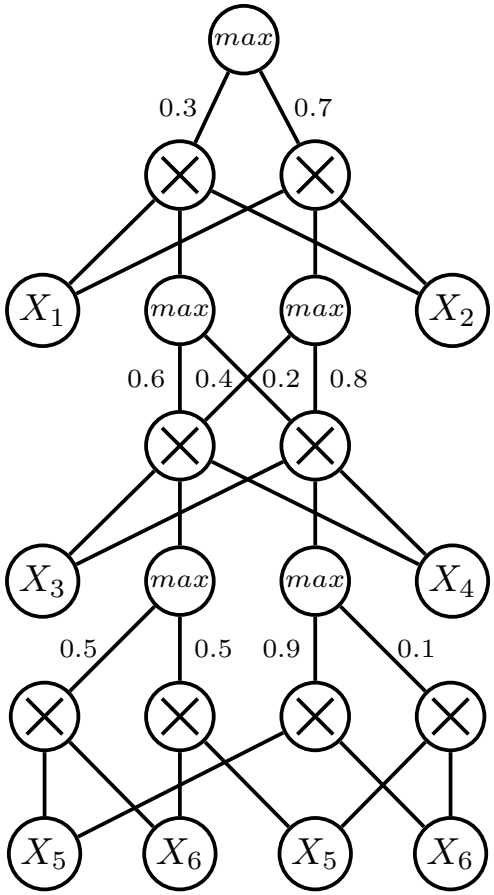
Issues:

- vanishing gradients: depth is a major problem for *soft* gradients
- hyperparameter choices
  - adaptive learning rate scheduling algos not employed yet!

[18] Darwiche, "A Differential Approach to Inference in Bayesian Networks", 2003

“Hard” gradients

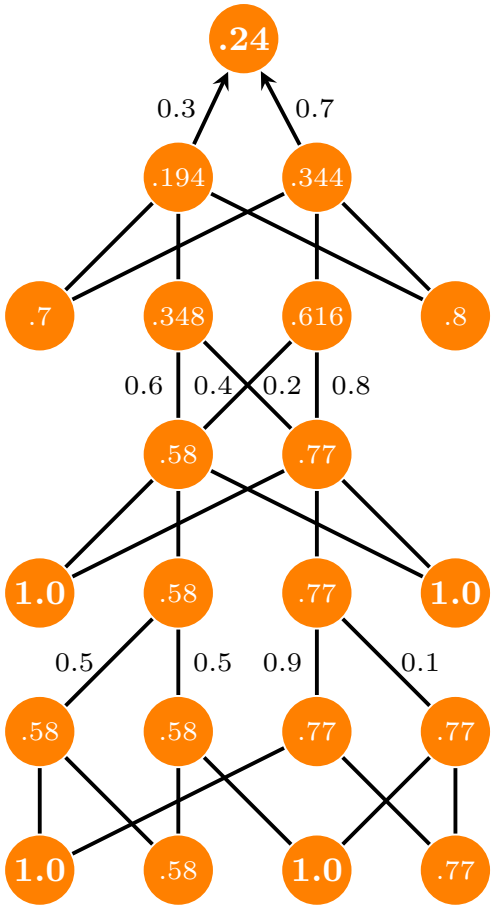
From SPN S to MPN M



Poon and Domingos, “Sum-Product Networks: a New Deep Architecture”, 2011

Gens and Domingos, “Discriminative Learning of Sum-Product Networks”, 2012

“Hard” gradients



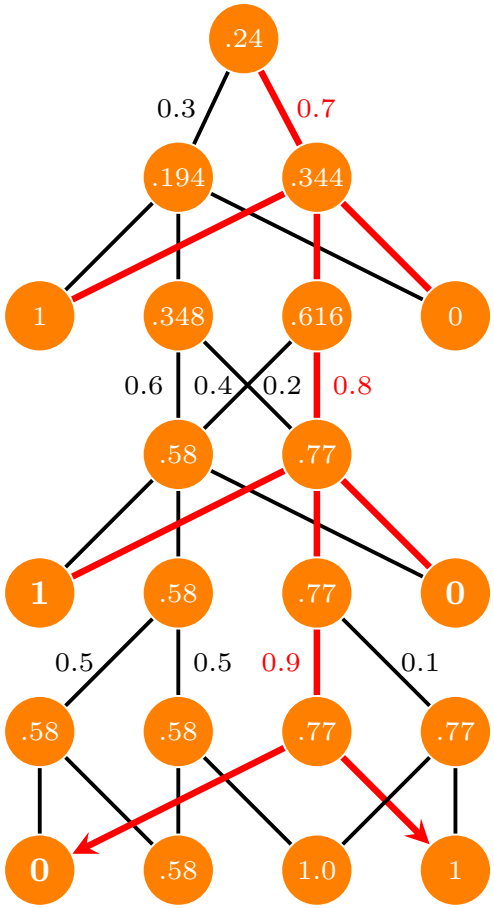
From SPN S to MPN M

- forward (bottom-up) prop $\mathbf{x}^i$

Poon and Domingos, “Sum-Product Networks: a New Deep Architecture”, 2011

Gens and Domingos, “Discriminative Learning of Sum-Product Networks”, 2012

“Hard” gradients



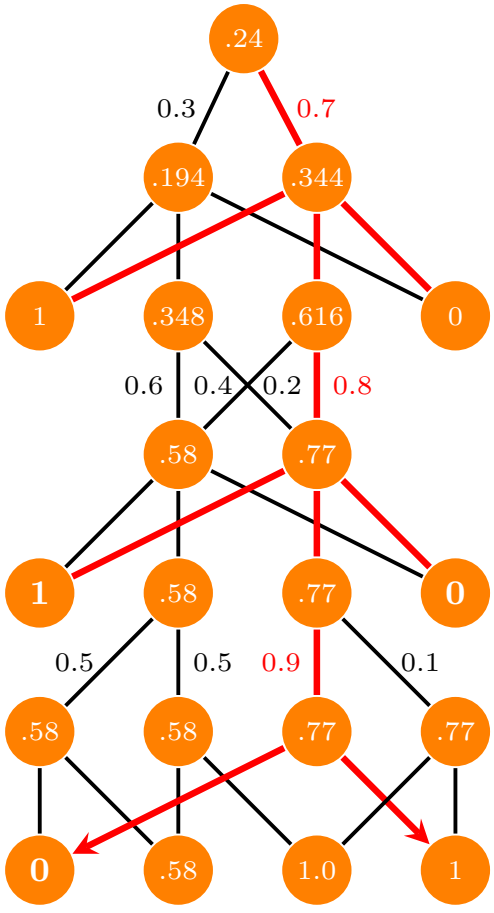
From SPN S to MPN M

- forward (bottom-up) prop $\mathbf{x}^i$
- backprop as MPE descent

Poon and Domingos, “Sum-Product Networks: a New Deep Architecture”, 2011

Gens and Domingos, “Discriminative Learning of Sum-Product Networks”, 2012

“Hard” gradients



From SPN S to MPN M

- forward (bottom-up) prop $\mathbf{x}^i$
- backprop as MPE descent
- “count” the weights occurrencies in the path $W_{\mathbf{x}^i}$

$$\nabla_{w_{pc}} \log M(\mathbf{x}) = \frac{\sharp\{w_{pc} \in W_{\mathbf{x}}\}}{w_{pc}}$$

$\rightarrow$ not vanishing (regardless depth)

$\rightarrow$ slower convergence
(less updates/instance)

Poon and Domingos, “Sum-Product Networks: a New Deep Architecture”, 2011

Gens and Domingos, “Discriminative Learning of Sum-Product Networks”, 2012

Hard/Soft Parameter Updating

	Δw_{pc}
Soft Gradient	
Generative ($\nabla_{w_{pc}} S(\mathbf{x})$)	$S_c(\mathbf{x}) \nabla_{S_p(\mathbf{x})} S(\mathbf{x})$
Discriminative ($\nabla_{w_{pc}} \log S(\mathbf{y}\|\mathbf{x})$)	$\frac{\nabla_{w_{pc}} S(\mathbf{y}\|\mathbf{x})}{S(\mathbf{y}\|\mathbf{x})} - \frac{\nabla_{w_{pc}} S(*\|\mathbf{x})}{S(*\|\mathbf{x})}$
Hard Gradient	
Generative ($\nabla_{w_{pc}} \log M(\mathbf{x})$)	$\frac{\sharp\{w_{pc} \in W_{\mathbf{x}}\}}{w_{pc}}$
Discriminative ($\nabla_{w_{pc}} \log M(\mathbf{y}\|\mathbf{x})$)	$\frac{\sharp\{w_{pc} \in W_{(\mathbf{y}\|\mathbf{x})}\} - \sharp\{w_{pc} \in W_{(\mathbf{1}\|\mathbf{x})}\}}{w_{pc}}$
Soft Posterior[19] ($p(H_p = c\|\mathbf{x})$)	$\propto \frac{1}{S(\mathbf{x})} \frac{\partial S(\mathbf{x})}{\partial S_p(\mathbf{x})} S_c(\mathbf{x}) w_{pc}$
Hard Posterior ($p(H_p = c\|\mathbf{x})$)	$= \begin{cases} 1 \text{if} w_{pc} \in W_{\mathbf{x}} \\ 0 \text{otherwise} \end{cases}$

[19] Peharz et al., "On the Latent Variable Interpretation in Sum-Product Networks", 2016

Gens and Domingos, "Discriminative Learning of Sum-Product Networks", 2012

Bayesian Parameter Learning

Learning in a Bayesian setting is computing the posterior $p(\mathbf{w}|\{\mathbf{x}^i\}_{i=1}^m)$ having a prior $p(\mathbf{w})$:

$$p(\mathbf{w}|\{\mathbf{x}^i\}_{i=1}^{t+1}) \propto p(\mathbf{w}|\{\mathbf{x}^i\}_{i=1}^{t})p(\mathbf{x}^{t+1}|\mathbf{w})$$

$p(\mathbf{w})$ modeled as a product of Dirichlet, $p(\mathbf{x}^{t+1}|\mathbf{w})$ is an exponential sum of monomials, $\rightarrow$ the posterior becomes a mixture of products of Dirichlets growing exponentially in the data and sum nodes!

Online Bayesian Moment Matching (OBMM): computing first two moments to approximate the intractable posterior, efficiently for tree SPNs **[Rashwan et al. 2016].**

Collapse Variational Inference (CVB-SPN) to optimize a logarithmic lower bound (better than ELBO) efficiently (linear in $|S|$) **[Zhao et al. 2016].**

Parameter learning

	CVB-SPN[20]	OBMM[21]	SGD[48]	EM[48]	SEG[48]
NLTCS	-6.08	-6.07	-8.76	-6.31	-6.85
MSNBC	-6.29	-6.03	-6.81	-6.64	-6.74
KDDCup2k	-2.14	-2.14	-44.53	-2.20	-2.34
Plants	-12.86	-15.14	-21.50	-17.68	-33.47
Audio	-40.36	-40.70	-49.35	-42.55	-46.31
Jester	-54.26	-53.86	63.89	-54.26	-59.48
Netflix	-60.69	-57.99	64.27	-59.35	-64.48
Accidents	-29.55	-42.66	53.69	-43.54	-45.59
Retail	-10.91	-11.42	-97.11	-11.42	-14.94
Pumsb-star	-25.93	-45.27	-128.48	-46.54	-51.84
DNA	-86.73	-99.61	-100.70	-100.10	-105.25
Kosarek	-10.70	-11.22	34.64	-11.87	-17.71
MSWeb	-9.89	-11.33	-59.63	-11.36	-20.69
Book	-34.44	-35.55	-249.28	-36.13	-42.95
EachMovie	-52.63	-59.50	-227.05	-64.76	-84.82
WebKB	-161.46	-165.57	-338.01	-169.64	-179.34
Reuters-52	-85.45	-108.01	-407.96	-108.10	-108.42
20-Newsgrp	-155.61	-158.01	-312.12	-160.41	-167.89
BBC	-251.23	-275.43	-462.96	-274.82	-276.97
Ad	-19.00	-63.81	-638.43	-63.83	-64.11

[20] Zhao et al., "Collapsed Variational Inference for Sum-Product Networks", 2016

[21] Rashwan et al., "Online and Distributed Bayesian Moment Matching for Parameter Learning in Sum-Product Networks", 2016

Parameter learning VS LearnSPN

	LearnSPN[22]	LearnSPN-b[23]	CVB-SPN[24]	OBMM[25]	SGD[52]	EM[52]	SEG[52]
NLTCS	-6.11	-6.05	-6.08	-6.07	-8.76	-6.31	-6.85
MSNBC	-6.11	-6.04	-6.29	-6.03	-6.81	-6.64	-6.74
KDDCup2k	-2.18	-2.14	-2.14	-2.14	-44.53	-2.20	-2.34
Plants	-12.98	-12.81	-12.86	-15.14	-21.50	-17.68	-33.47
Audio	-40.50	-40.57	-40.36	-40.70	-49.35	-42.55	-46.31
Jester	-53.48	-53.53	-54.26	-53.86	63.89	-54.26	-59.48
Netflix	-57.33	-57.73	-60.69	-57.99	64.27	-59.35	-64.48
Accidents	-30.04	-29.34	-29.55	-42.66	53.69	-43.54	-45.59
Retail	-11.04	-10.94	-10.91	-11.42	-97.11	-11.42	-14.94
Pumsb-star	-24.78	-23.31	-25.93	-45.27	-128.48	-46.54	-51.84
DNA	-82.52	-81.91	-86.73	-99.61	-100.70	-100.10	-105.25
Kosarek	-10.99	-10.72	-10.70	-11.22	34.64	-11.87	-17.71
MSWeb	-10.25	-9.83	-9.89	-11.33	-59.63	-11.36	-20.69
Book	-35.89	-34.30	-34.44	-35.55	-249.28	-36.13	-42.95
EachMovie	-52.49	-51.36	-52.63	-59.50	-227.05	-64.76	-84.82
WebKB	-158.20	-154.28	-161.46	-165.57	-338.01	-169.64	-179.34
Reuters-52	-85.07	-83.34	-85.45	-108.01	-407.96	-108.10	-108.42
20-Newsgrp	-155.93	-152.85	-155.61	-158.01	-312.12	-160.41	-167.89

[22] Gens and Domingos, “Learning the Structure of Sum-Product Networks”, 2013

[23] Vergari et al., “Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning”, 2015

[24] Zhao et al., “Collapsed Variational Inference for Sum-Product Networks”, 2016

[25] Rashwan et al., “Online and Distributed Bayesian Moment Matching for Parameter Learning in Sum-Product Networks”, 2016

Why learning parameters only

Even if simple, LearnSPN hardly scales on large datasets.

$\rightarrow$ generate a random (but valid) structure, then optimize the weights

	LearnSPN	OBMM	ODMM	SGB	OEM	OEG
KOS	-444.55	**-422.19**	-437.30	-3492.9	-538.21	-657.13
NIPS	-	**-1691.87**	-1709.04	-7411.20	-1756.06	-3134.59
ENRON	-	**-518.842**	-522.45	-13961.40	-554.97	-14193.90
NyTIMES	-	-1503.65	-1559.39	-43153.60	**-1189.39**	-6318.71

$\rightarrow$ distribute the computation of gradients and updates (over instances,...etc)

	LearnSPN	OBMM	ODMM	SGB	OEM	OEG
KOS	1439.11	89.40	**8.66**	162.98	59.49	155.34
NIPS	-	139.50	**9.43**	180.25	64.62	178.35
ENRON	-	2018.05	**580.63**	876.18	694.17	883.12
NyTIMES	-	12091.7	**1643.60**	5626.33	5540.40	6895.00

Rashwan et al., "Online and Distributed Bayesian Moment Matching for Parameter Learning in Sum-Product Networks", 2016

Other Tendencies in Parameter Learning

Jointly learning leaf distributions parameters while optimizing.
E.g. deriving EM update rules for leaf distributions **[Desana and Schnörr 2016; Peharz et al. 2015]**

Bayesian learning with continous leaf distributions. Extending OBMM to tree SPNs with continuous Gaussian leaves **[Jaini et al. 2016]**$\longrightarrow$ *PGM'16!*

Non bayesian **signomial programming approaches** still considering an SPN as a (very large) mixture over tree distributions.
Multiplicative updates (no projections, like EG, but faster convergence) for Sequential Monomial Approximations (SMA) and Concave-Convex procedure (CCCP) **[Zhao and Poupart 2016]**

Applications

Applications I: computer vision

Image reconstruction and *inpainting*: fill the missing pixels of test samples by the means of efficient MPE inference.
Fixed (taking spatial autocorrelation into account) or learned structures.

Reconstructing some simmetries (eyes, but not beards, glasses).

Testing different approximations for MPE inference **[Peharz 2015].**

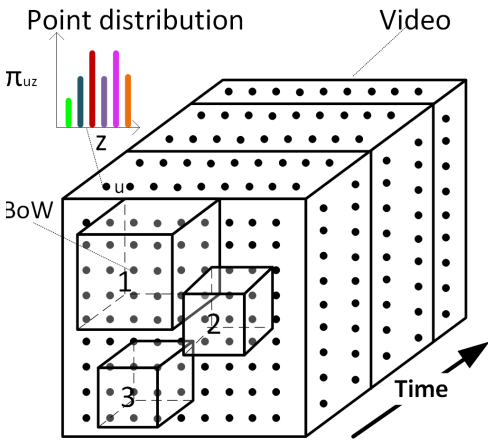
Poon and Domingos, "Sum-Product Networks: a New Deep Architecture", 2011

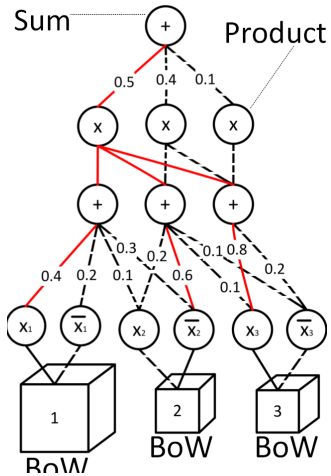
Applications II: activity recognition

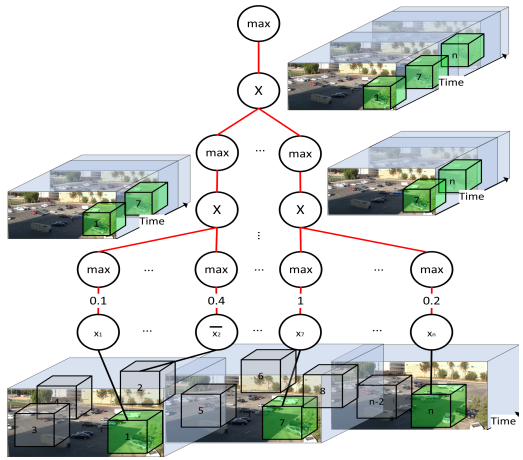
Videos represented as regular grids of points in space and time, described by Bag-of-Words (BoW).

An SPN structure models a hierarchy of BoW products.

Inference for activity recognition *and localization*.







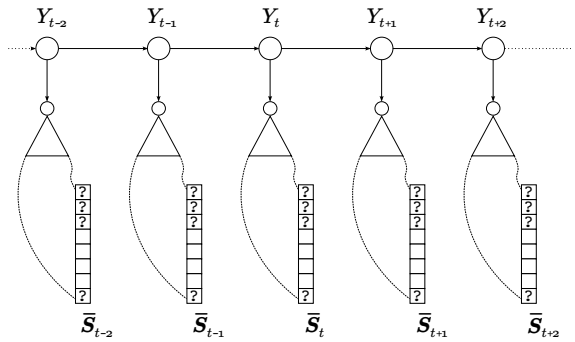
Exploiting part based decomposability along pixels *and time* (frames).

Amer and Todorovic, "Sum-Product Networks for Modeling Activities with Stochastic Structure", 2012

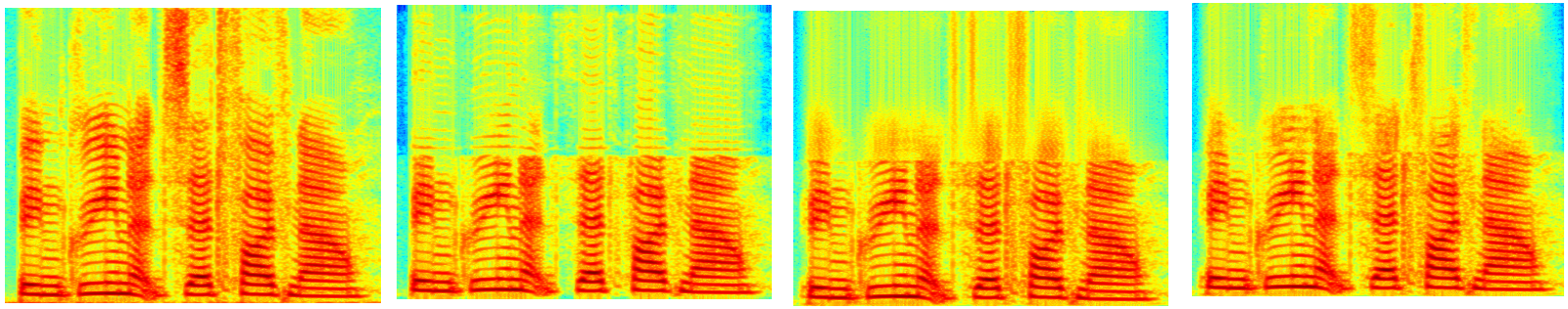
Amer and Todorovic, "Sum Product Networks for Activity Recognition", 2015

Applications III: speech

SPNs to model the joint pdf of observed RVs in HMMs (HMM-SPNs).



State-of-the-art high frequency reconstruction (MPE inference)

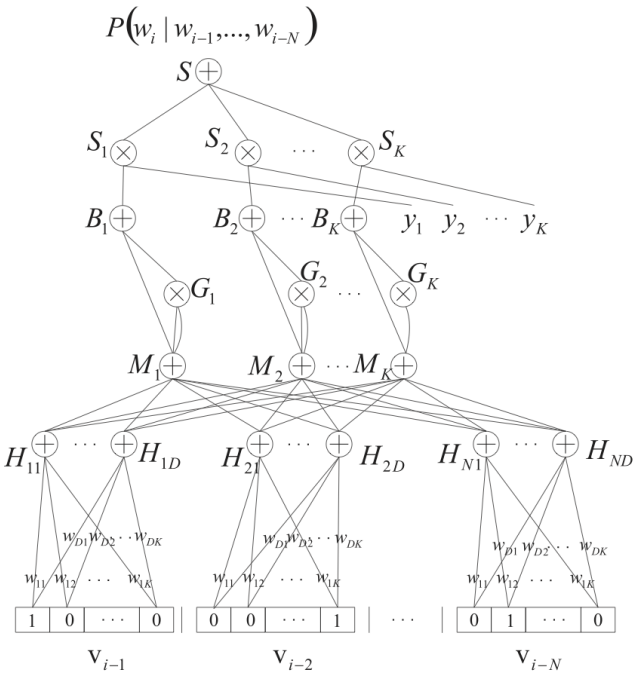


(a) Original full bandwidth (b) Reconstruction HMM-LP (c) Reconstruction HMM-GMM (d) Reconstruction HMM-SPN

Peharz et al., "Modeling speech with sum-product networks: Application to bandwidth extension", 2014

Applications IV: language modeling

Fixed structure SPN encoding the conditional probability $p(w_i|w_{i-1}, \ldots, w_{i-n})$ as an n-th order language model.



One-hot encoding of word vocabulary. Windowed representation of size

First **embedding layer** with size D, sharing word weights across different mixtures (position invariance).

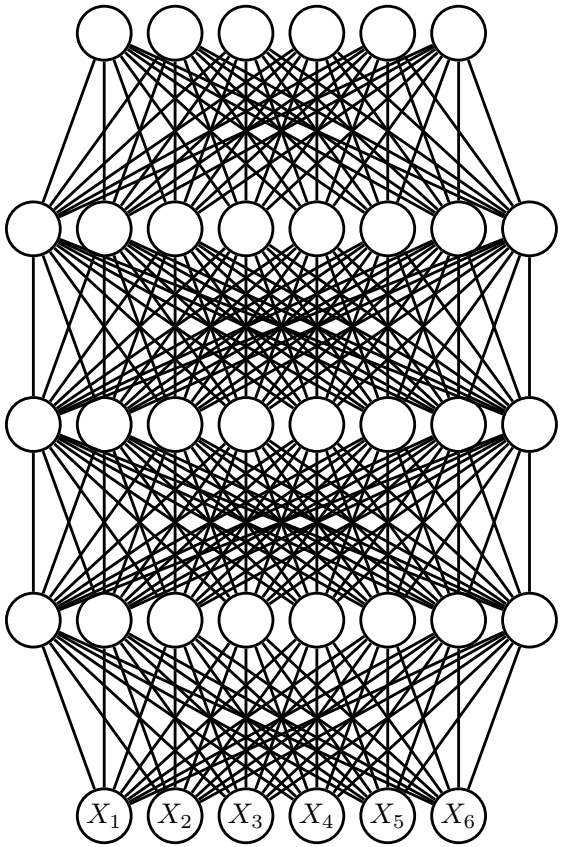
State-of-the-art *perplexity* on PennTreeBank even for low orders ($n = 4$).

Cheng et al., "Language modeling with Sum-Product Networks", 2014

Representation Learning

Extracting Embeddings

From deep neural networks



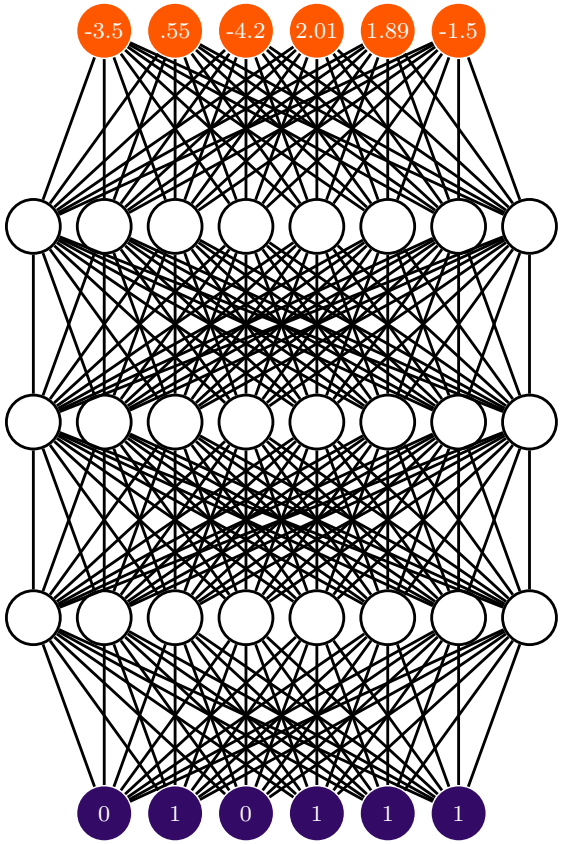
Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

Bengio et al., "Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives", 2012

Extracting Embeddings

From deep neural networks



Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by evaluating the network and collecting the last layer(s) activations

$$\mathbf{e}^i = \langle -3.5, .55, -4.2, 2.01, 1.89, -1.5 \rangle$$

Bengio et al., "Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives", 2012

Extracting Embeddings

Exploiting SPNs as feature extractors

Given an SPN S, a filtering criterion f, generate a dense vector for each sample $\mathbf{x}^i$

$$\mathbf{e}^i = f_S(\mathbf{x}^i)$$

Issues with SPNs as NNs:

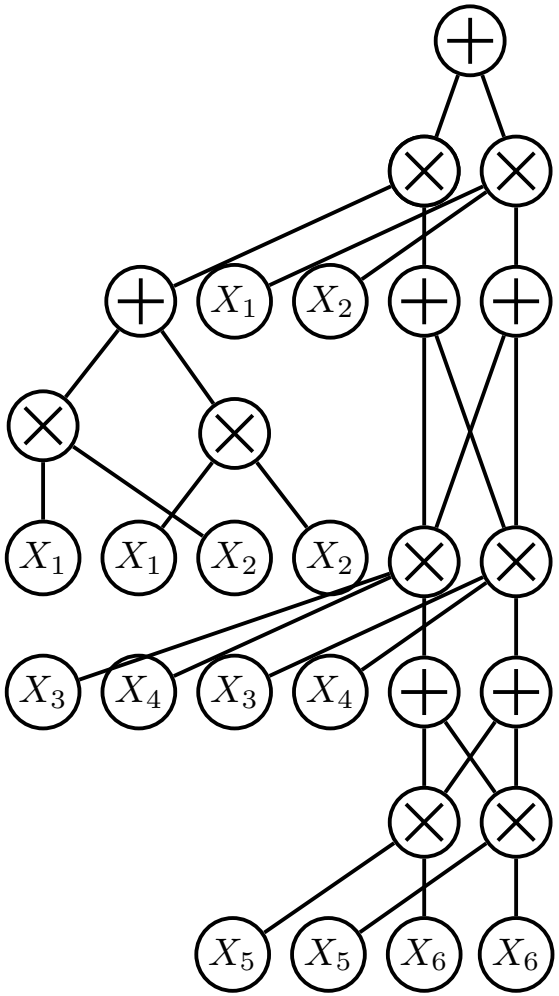
- layer-wise extraction may be arbitrary
- power law distribution of nodes by scopes
- scope lengths as proxy for feature abstraction levels (see filter visualizations)

$\rightarrow$ Which filtering criterion to employ?
$\rightarrow$ Which interpretation for the extracted features?

Vergari et al., "Visualizing and Understanding Sum-Product Networks", 2016

Extracting embeddings

Inner node activations

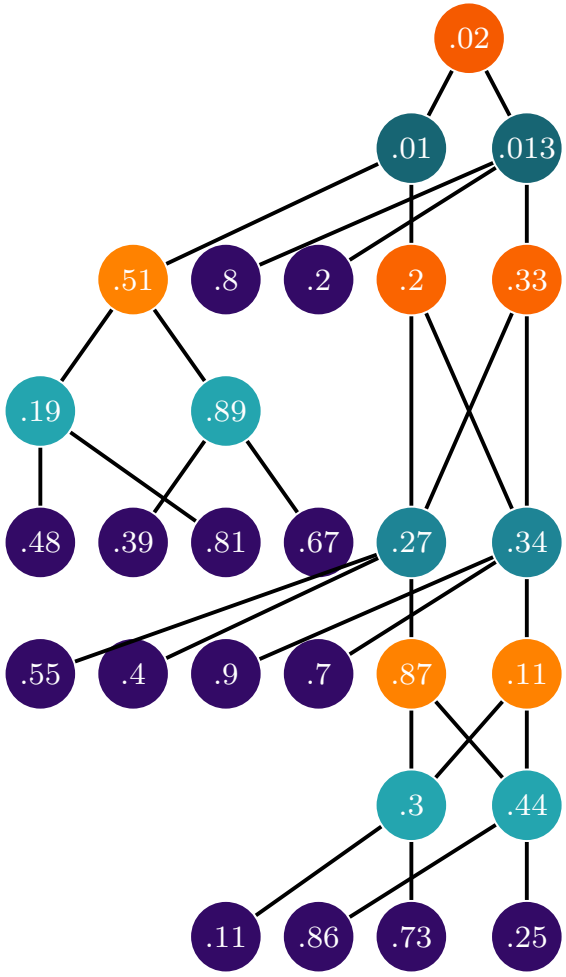


Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

Extracting embeddings

Inner node activations



Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

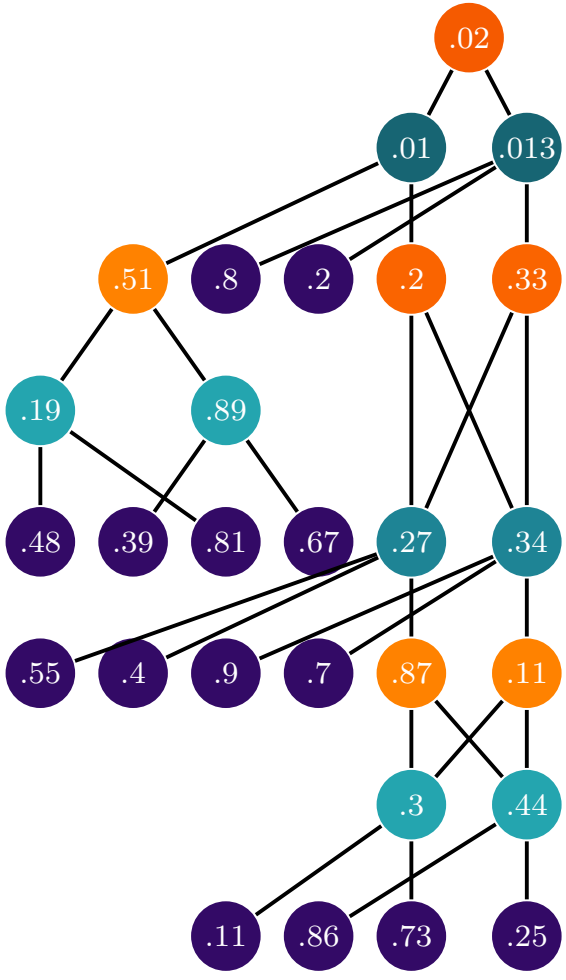
$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by evaluating $S(\mathbf{x}^i)$ and collecting inner node (sum , product but not leaves) activations

$$\mathbf{e}^i = \langle .02, .01, .013, .51, .2, .33, .19, .89, .27, .34, .87, .11, .3, .44 \rangle$$

Extracting embeddings

Filtering by type



Build embeddings $\mathbf{e}^i_{\text{sum}}, \mathbf{e}^i_{\text{prod}}$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

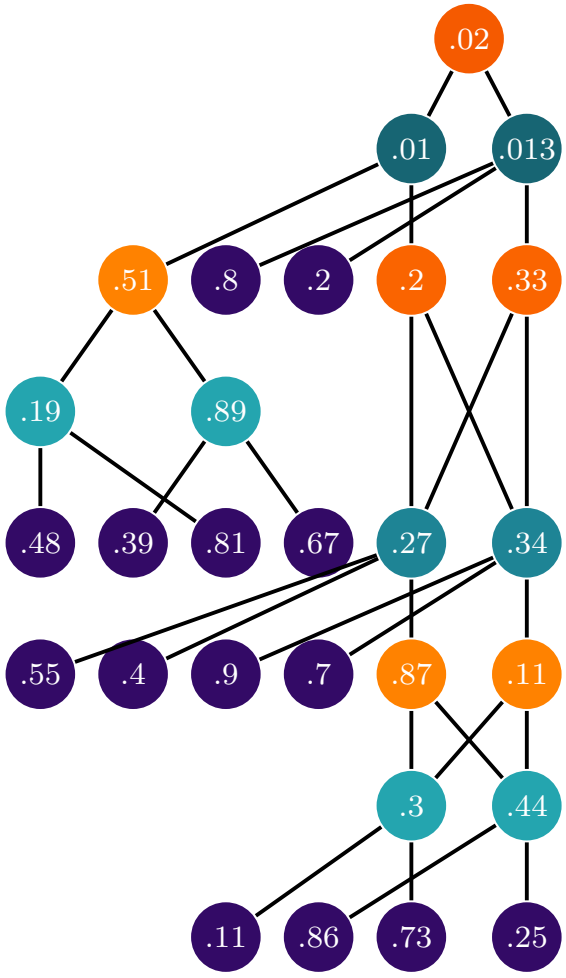
by evaluating $S(\mathbf{x}^i)$ and collecting inner node activations filtered by node type

$$\mathbf{e}^i_{\text{sum}} = \langle .02, .51, .2, .33, .87, .11 \rangle$$

$$\mathbf{e}^i_{\text{prod}} = \langle .01, .013, .19, .89, .27, .34, .3, .44 \rangle$$

Extracting embeddings

Filtering by scope length



Build embeddings $\mathbf{e}^i_{|\mathsf{sc}(n)|=k} \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by evaluating $S(\mathbf{x}^i)$ and collecting inner node activations filtered by scope length

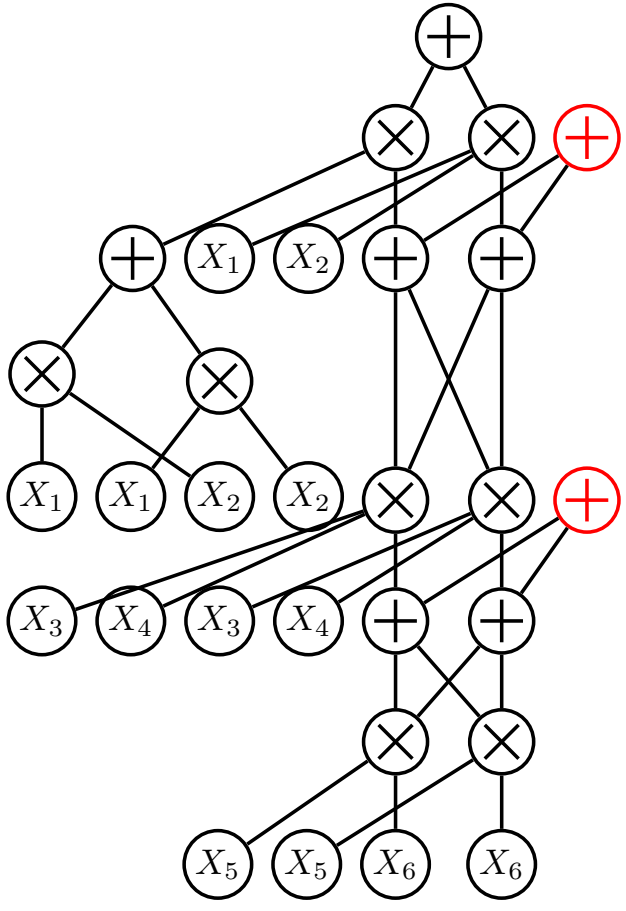
$$\mathbf{e}^i_{|\mathsf{sc}(n)|=2} = \langle .51, .19, .89, .87, .11, .3, .44 \rangle$$

$$\mathbf{e}^i_{|\mathsf{sc}(n)|=4} = \langle .2, .33, .27, .34 \rangle$$

$$\mathbf{e}^i_{|\mathsf{sc}(n)|=6} = \langle .02, .01, .013 \rangle$$

Extracting embeddings

Aggregating by scope



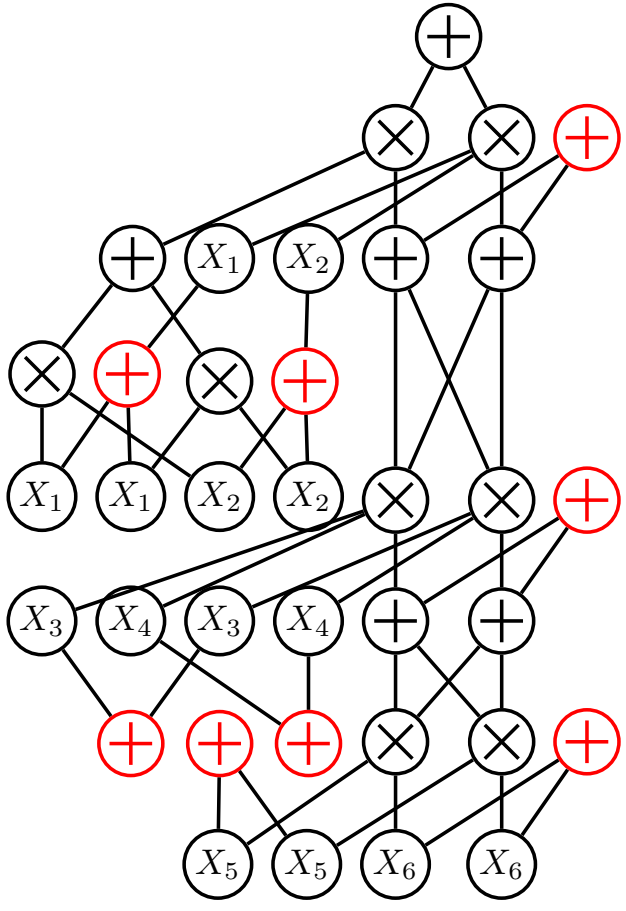
Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by adding fictitious sum nodes over unique scopes (as additional roots)

Extracting embeddings

Aggregating by scope



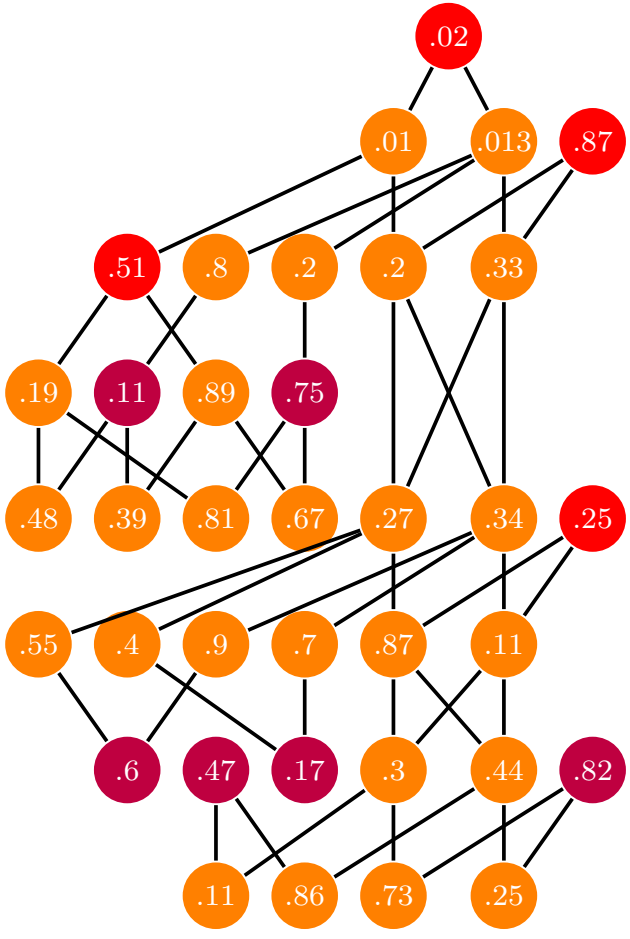
Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

by adding fictitious sum nodes over unique scopes (as additional roots), then evaluating $S(\mathbf{x}^i)$ and collecting they activations from inner nodes

Extracting embeddings

Aggregating by scope



Build an embedding $\mathbf{e}^i \in \mathbb{R}^d$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

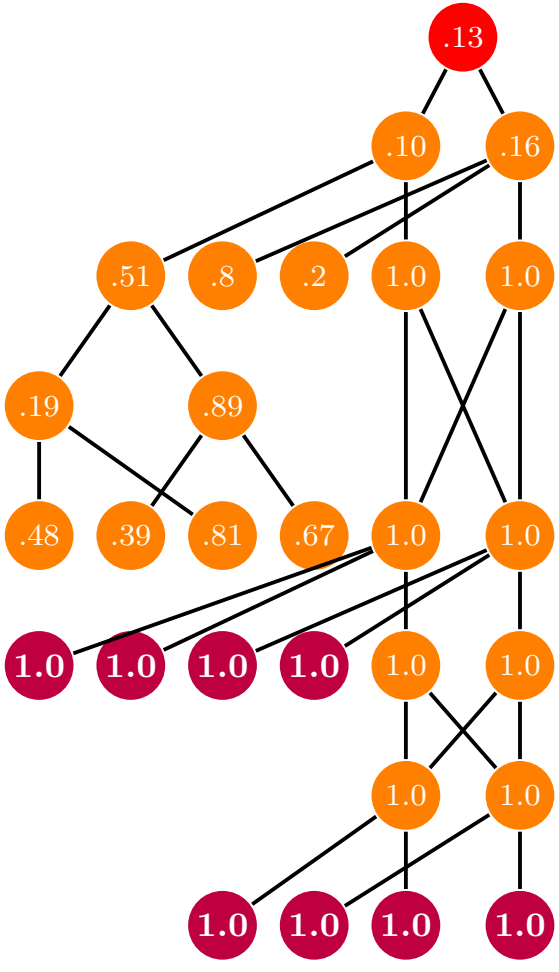
by adding fictitious sum nodes over unique scopes (as additional roots), then evaluating $S(\mathbf{x}^i)$ and collecting they activations from inner nodes and even leaves

$$\mathbf{e}^i_{\text{w/o}-\text{leaves}} = \langle .2, .87, .51, .25 \rangle$$

$$\mathbf{e}^i_{\text{w}-\text{leaves}} = \langle .2, .87, .51, .11, .75, .25, .6, .47, .17, .82 \rangle$$

Extracting embeddings

Random marginal queries



To build an embedding $\mathbf{e}^i \in \mathbb{R}^k$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

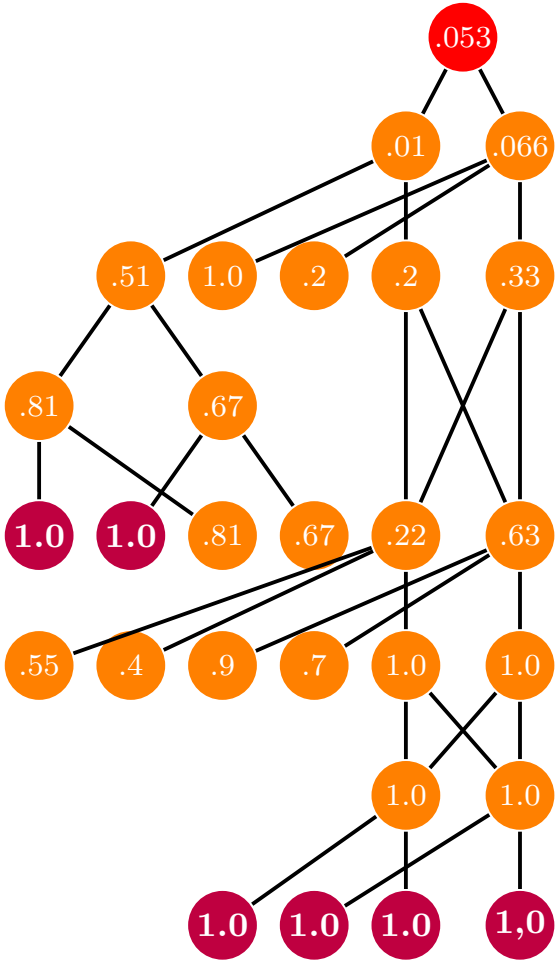
for each feature $j = 1, \ldots, k$, sample $\mathbf{Q}_j \subset \mathbf{X}$ and evaluate $p(\mathbf{Q}_j = \mathbf{x}_{\mathbf{Q}_j})$.
E.g.:

$$\mathbf{Q}_0 = \{X_1, X_2\}$$

$$\mathbf{e}^i_{\text{rand}} = \langle \boxed{.13}, \ldots \rangle$$

Extracting embeddings

Random marginal queries



To build an embedding $\mathbf{e}^i \in \mathbb{R}^k$ for sample

$$\mathbf{x}^i = \langle 0, 1, 0, 1, 1, 1 \rangle$$

for each feature $j = 1, \ldots, k$, sample $\mathbf{Q}_j \subset \mathbf{X}$ and evaluate $p(\mathbf{Q}_j = \mathbf{x}_{\mathbf{Q}_j})$.
E.g.:

$$\mathbf{Q}_0 = \{X_1, X_2\}$$

$$\mathbf{Q}_1 = \{X_1, X_3, X_4\}$$

$$\mathbf{e}^i_{\text{rand}} = \langle .13, .053 \ldots \rangle$$

Supervised classification

Experimental settings to evaluate embeddings

Extract embeddings unsupervisedly on $\mathbf{X}$, then train a logistic regressor on them to predict Y.

Five image datasets: REC, CON, OCR, CAL, BMN.

Grid search with LearnSPN-b for three models with different capacities: SPN-I, SPN-II and SPN-III for $m \in \{500, 100, 50\}$.

Compare them against RBM models: RBM-5h, RBM-1k and RBM-5k with 500, 1000 and 5000 hidden units.

Compare them against other tractable PGMs: mixtures of 3, 15, 30 Chow-Liu trees.

Embedding accuracies (I)

Table : Test set accuracy scores for the embeddings extracted with the best SPN, RBM models and with the baseline LR model on all datasets. Bold values denote significantly better scores than all the others for a dataset.

	LR	SPN-I	SPN-II	SPN-III	RBM-5h	RBM-1k	RBM-5k
REC	69.28	77.31	**97.77**	97.66	94.22	96.10	96.36
CON	53.48	67.48	78.31	**84.69**	67.55	75.37	79.15
OCR	75.58	82.60	**89.95**	**89.94**	86.07	87.96	88.76
CAL	62.67	59.17	65.19	66.62	67.36	**68.88**	67.71
BMN	90.62	95.15	**97.66**	97.59	96.09	96.80	97.47

$\rightarrow$ *comparable or better accuracies than (intractable) RBM embeddings*

Embedding accuracies (II)

Table : Test set accuracies for embeddings filtered by node type and by Small , Medium and Large scope lengths. Bold values denote significantly better scores than all the others. ▲ indicates a better score than an RBM embedding with greater or equal size. ▽ indicates worse scores than an RBM embedding with smaller or equal size.

	SPN-I		SPN-II		SPN-III		SPN-III		
	sum	prod	sum	prod	sum	prod	S	M	L
REC	72.46	62.25	**98.03**▲	97.06▲	**98.00**▲	97.04▲	88.73	**98.45**▲	93.91
CON	62.36	64.03	77.13▲	76.07▲	**83.59**▲	82.06▲	70.51▽	77.18	**83.32**▲
OCR	74.19	81.58	89.73▲	88.78▲	**90.02**▲	89.32	87.22▽	**89.29**▲	88.19▲
CAL	38.19	56.95	62.64	64.80	**66.58**▽	66.40▽	63.37▽	**66.23**▽	66.10
BMN	93.50	94.75	97.67	96.90▽	**97.80**	97.20▽	96.02▽	**97.42**▽	97.38

→ *sum nodes only are good compressors for larger models*

→ *mid size scope length embeddings provide best discriminative power*

Vergari et al., "Visualizing and Understanding Sum-Product Networks", 2016

Embedding accuracies (III)

Table : Test set accuracies for embeddings by aggregating node outputs with the same scope, with and without leaves. Bold values denote significantly better scores than all the others for each dataset. ▲ indicates a better score than an RBM embedding with greater or equal size. ▽ indicates worse scores than an RBM embedding with smaller or equal size.

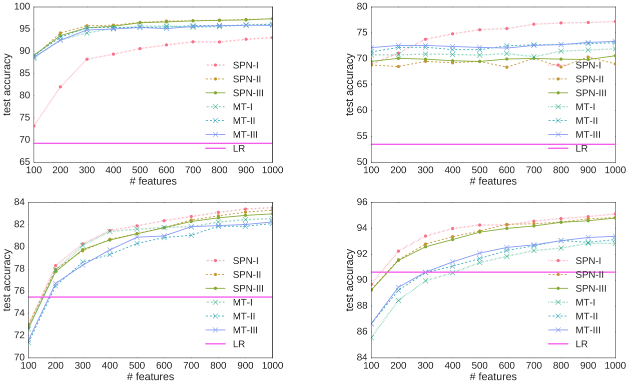
	SPN-I		SPN-II		SPN-III	
	no-leaves	leaves	no-leaves	leaves	no-leaves	leaves
REC	72.47	$75.92^{\triangledown}$	$\mathbf{97.94}^{\blacktriangle}$	$\mathbf{97.99}^{\blacktriangle}$	$\mathbf{97.94}^{\blacktriangle}$	$\mathbf{98.02}^{\blacktriangle}$
CON	62.35	$66.49^{\triangledown}$	$77.21^{\blacktriangle}$	78.05	$\mathbf{83.52}^{\blacktriangle}$	$\mathbf{83.84}^{\blacktriangle}$
OCR	74.32	81.85	$89.71^{\blacktriangle}$	$89.68^{\blacktriangle}$	$\mathbf{89.90}^{\blacktriangle}$	$\mathbf{89.91}^{\blacktriangle}$
CAL	38.10	$63.19^{\triangledown}$	62.59	$62.76^{\triangledown}$	$\mathbf{66.49}^{\triangledown}$	$\mathbf{66.58}^{\triangledown}$
BMN	93.51	$94.83^{\triangledown}$	$97.64^{\blacktriangle}$	$97.62^{\blacktriangle}$	**97.80**	**97.80**

$\rightarrow$ *shorter but still accurate embeddings*

$\rightarrow$ *leaves greatly contribute for smaller models*

Vergari et al., "Visualizing and Understanding Sum-Product Networks", 2016

Random Marginal Queries



→ *structure learning is meaningful!*

→ *an SPN representation power goes beyond single nodes*

Vergari et al., "Visualizing and Understanding Sum-Product Networks", 2016

Encoding/Decoding Embeddings

Treat an MPN as a sort of autoencoder:

- **encoding** a sample into an inner nodes embedding
- **decoding** an embedding back into input space by MPE top down traversal

Evaluating in the *Multi Label Classification* MLC case, where $\mathbf{X} \rightarrow \mathbf{Y}$ is much harder than $\mathbf{X} \rightarrow Y$.
Three proxy performance metrics: jaccard, hamming and exact match scores.
Learning with Logistic Regression (LR) and Ridge Regression (RR)

- $\mathbf{X} \overset{MPN}{\longmapsto} \mathbf{E_X} \overset{LR}{\longrightarrow} \mathbf{Y}$: slight improvements
- $(\mathbf{X} \overset{RR}{\longrightarrow} (\mathbf{Y} \overset{MPN}{\longmapsto} \mathbf{E_Y})) \overset{MPN}{\longmapsto} \mathbf{Y}$: huge improvements
- $((\mathbf{X} \overset{MPN_\mathbf{X}}{\longmapsto} \mathbf{E_X}) \overset{RR}{\longrightarrow} (\mathbf{Y} \overset{MPN_\mathbf{Y}}{\longmapsto} \mathbf{E_Y})) \overset{MPN_\mathbf{Y}}{\longmapsto} \mathbf{Y}$: harder

---

Vergari et al. Encoding and Decoding Representations with Sum-Product Networks, 2016, to appear

Trends & What to do next

Scalable structure learning to cope with million instances and RVs. LearnSPN can be tweaked some more, but... **[Krakovna and Looks 2016]**

Continuous RVs structure learning. Is enough to adapt LearnSPN clustering processes to operate on continuous RVs?

Compressing and lifting huge SPN models. Would it be fine to renounce to answer queries of a certain kind in tractable fashion?

End-to-end learning with hybrid NN architectures. Deep learning architectures leaped forward recently and on harder tasks...

References

awesome-spn

A curated and structured list of resources about SPNs[26].
`https://github.com/arranger1044/awesome-spn`

[26] Inspired by the SPN page http://spn.cs.washington.edu/ at the Washington University

awesome-spn

A curated and structured list of resources about SPNs[26].
`https://github.com/arranger1044/awesome-spn`

Thanks to the research teams working on SPNs!

University of Washington: Domingos, Poon, Gens
University of Graz: Peharz, Pernkopf, Tschiatschek
University of Waterloo: Poupart, Zhao, Adel, Melibari, Rashwan, Jaini
University of Oregon: Lowd, Rooshenas
University of Bari: DiMauro, Vergari
Oregon State University: Todorovic, Amer
University of Texas at Dallas: Gogate, Rahman
...

[26] Inspired by the SPN page http://spn.cs.washington.edu/ at the Washington University

Bibliography I

Adel, Tameem, David Balduzzi, and Ali Ghodsi (2015). "Learning the Structure of Sum-Product Networks via an SVD-based Algorithm". In: *Uncertainty in Artificial Intelligence*.

Amer, Mohamed and Sinisa Todorovic (2015). "Sum Product Networks for Activity Recognition". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on*.

Amer, Mohamed R and Sinisa Todorovic (2012). "Sum-Product Networks for Modeling Activities with Stochastic Structure". In: *(CVPR), 2012 IEEE Conference on*. IEEE, pp. 1314–1321.

Bengio, Yoshua, Aaron C. Courville, and Pascal Vincent (2012). "Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives". In: *CoRR* abs/1206.5538.

Cheng, Wei-Chen et al. (2014). "Language modeling with Sum-Product Networks". In: *INTERSPEECH 2014*, pp. 2098–2102.

Darwiche, Adnan (2003). "A Differential Approach to Inference in Bayesian Networks". In: *J.ACM*.

— (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge.

Dennis, Aaron and Dan Ventura (2015). "Greedy Structure Search for Sum-product Networks". In: *IJCAI'15*. Buenos Aires, Argentina: AAAI Press, pp. 932–938. ISBN: 978-1-57735-738-4.

Desana, Mattia and Christoph Schnörr (2016). "Expectation Maximization for Sum-Product Networks as Exponential Family Mixture Models". In: *CoRR* abs/1604.07243. URL: `http://arxiv.org/abs/1604.07243`.

Erhan, Dumitru et al. (2009). "Visualizing Higher-Layer Features of a Deep Network". In: *ICML 2009 Workshop on Learning Feature Hierarchies, Montréal, Canada.*

Gens, Robert and Pedro Domingos (2012). "Discriminative Learning of Sum-Product Networks". In: *Advances in Neural Information Processing Systems 25*, pp. 3239–3247.

— (2013). "Learning the Structure of Sum-Product Networks". In: *Proceedings of the ICML 2013*, pp. 873–880.

Germain, Mathieu et al. (2015). "MADE: Masked Autoencoder for Distribution Estimation". In: *CoRR* abs/1502.03509.

Jaini, Priyank et al. (2016). "Online Algorithms for Sum-Product Networks with Continuous Variables". In: *Probabilistic Graphical Models - Eighth International Conference, PGM 2016, Lugano, Switzerland, September 6-9, 2016. Proceedings*, pp. 228–239. URL: `http://jmlr.org/proceedings/papers/v52/jaini16.html`.

Krakovna, Viktoriya and Moshe Looks (2016). "A Minimalistic Approach to Sum-Product Network Learning for Real Applications". In: *CoRR* abs/1602.04259. URL: `http://arxiv.org/abs/1602.04259`.

Larochelle, Hugo and Iain Murray (2011). "The Neural Autoregressive Distribution Estimator". In: *International Conference on Artificial Intelligence and Statistics*, pp. 29–37.

Lowd, Daniel and Pedro Domingos (2012). "Learning Arithmetic Circuits". In: *CoRR* abs/1206.3271.

Bibliography II

Lowd, Daniel and Amirmohammad Rooshenas (2013). "Learning Markov Networks With Arithmetic Circuits". In: *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*. Vol. 31. JMLR Workshop Proceedings, pp. 406–414.

Martens, James and Venkatesh Medabalimi (2014). "On the Expressive Efficiency of Sum Product Networks". In: *CoRR* abs/1411.7717.

Melibari, Mazen et al. (2016). "Dynamic Sum Product Networks for Tractable Inference on Sequence Data". In: *Probabilistic Graphical Models - Eighth International Conference, PGM 2016, Lugano, Switzerland, September 6-9, 2016. Proceedings*, pp. 345–355. URL: http://jmlr.org/proceedings/papers/v52/melibari16.html.

Nath, Aniruddh and Pedro Domingos (2015). "Learning Relational Sum-Product Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Peharz, Robert (2015). "Foundations of Sum-Product Networks for Probabilistic Modeling". PhD thesis. Graz University of Technology, SPSC.

Peharz, Robert, Robert Gens, and Pedro Domingos (2014a). "Learning Selective Sum-Product Networks". In: *Workshop on Learning Tractable Probabilistic Models*. LTPM.

Peharz, Robert et al. (2014b). "Modeling speech with sum-product networks: Application to bandwidth extension". In: *ICASSP2014*.

Peharz, Robert et al. (2015). "On Theoretical Properties of Sum-Product Networks". In: *The Journal of Machine Learning Research*.

Peharz, Robert et al. (2016). "On the Latent Variable Interpretation in Sum-Product Networks". In: *CoRR* abs/1601.06180. URL: http://arxiv.org/abs/1601.06180.

Poon, Hoifung and Pedro Domingos (2011). "Sum-Product Networks: a New Deep Architecture". In: *UAI 2011*.

Rahman, Tahrima and Vibhav Gogate (2016). "Merging Strategies for Sum-Product Networks: From Trees to Graphs". In: *UAI*, ??–??

Rashwan, Abdullah, Han Zhao, and Pascal Poupart (2016). "Online and Distributed Bayesian Moment Matching for Parameter Learning in Sum-Product Networks". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 1469–1477.

Rooshenas, Amirmohammad and Daniel Lowd (2014). "Learning Sum-Product Networks with Direct and Indirect Variable Interactions". In: *Proceedings of ICML 2014*.

Vergari, A., N. Di Mauro, and F. Esposito (2016). "Visualizing and Understanding Sum-Product Networks". In: *preprint arXiv*. URL: https://arxiv.org/abs/1608.08266.

Vergari, Antonio, Nicola Di Mauro, and Floriana Esposito (2015). "Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning". In: *ECML-PKDD 2015*.

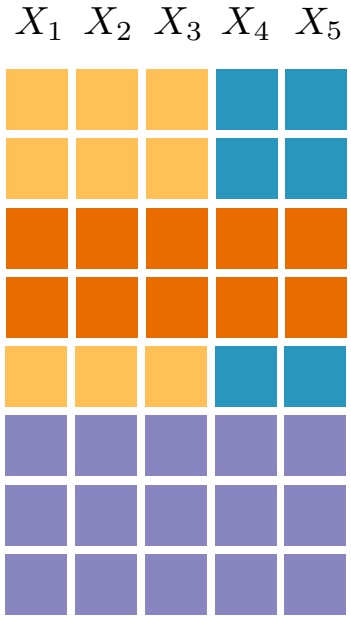
Zhao, Han and Pascal Poupart (2016). "A unified approach for learning the parameters of sum-product networks". In: *arXiv preprint arXiv:1601.00318*.

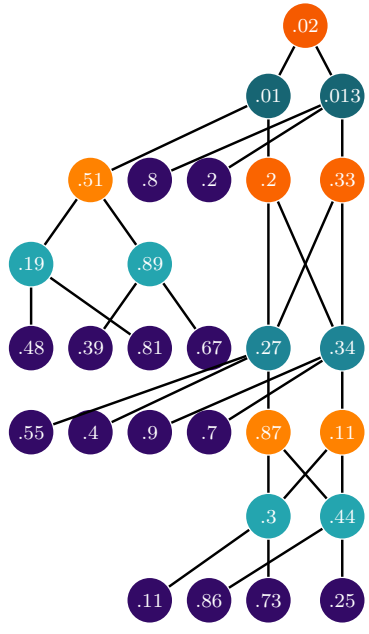
Zhao, Han, Mazen Melibari, and Pascal Poupart (2015). "On the Relationship between Sum-Product Networks and Bayesian Networks". In: *ICML*.

Bibliography III

Zhao, Han et al. (2016). “Collapsed Variational Inference for Sum-Product Networks”. In: *In Proceedings of the 33rd International Conference on Machine Learning*. Vol. 48.

Discuss


X_1 X_2 X_3 X_4 X_5


.02
.01
.013
.51
.8
.2
.2
.33
.19
.89
.48
.39
.81
.67
.27
.34
.55
.4
.9
.7
.87
.11
.3
.44
.11
.86
.73
.25