Probabilistic Modelling with Sum-Product Networks

Martin Trapp
November 14, 2019

Graz University of Technology
Outline

- Introduction to Sum-Product Networks
- Parameter Learning
- Structure Learning
- Applications of Sum-Product Networks
Introduction
• Sum-product networks (SPNs) [Poon2011] are tractable general-purpose machine learning models that admit exact probabilistic inference.

• Why do we care about probabilistic inference?

• Probabilistic inference allows us to derive how likely an event under a certain model is.

• Example: How likely is it that there is a traffic jam on Monday on the A2?
What are **tractable probabilistic models**?

A class of queries $Q$ on a class of models $M$ is tractable, iff for any query $q \in Q$ and model $m \in M$ the computational complexity is at most polynomial.

SPNs admit many probabilistic inference tasks, such as marginalisation, in linear time.

To model a probability distribution over $X$, SPNs use an **explicit** representation.

Note: Most recent generative models, such as GANs, are implicitly defined, intractable and require approximate inference.
Sum-Product Networks
What is a Sum-Product Network?

• Let $\mathbf{X} = \{X_1, \ldots, X_D\}$ be set of D random variables.
• An SPN is a distribution over $\mathbf{X}$ defined as a 4-tuple $S = (G, \psi, w, \theta)$. [Trapp2019]
  
  - $G$ is a computational graph.
  - $\psi$ is a so-called scope function.
  - $w$ denotes the set of sum-weights and $\theta$ the set of leaf node parameters.

Note: This definition is conceptually different to the classic definition of SPNs.
\( \mathcal{G} \) is a connected directed acyclic graph (DAG), containing three types of nodes: sums (S), products (P) and leaves (L).

Figure 1: Example of a tree-shaped computational graph.
Leaves $L$ in $\mathcal{G}$

$$L(x) = p(x \mid \theta_L)$$
Product Nodes $P$ in $G$

$$P(x) = \prod_{C \in \text{ch}(P)} C(x)$$
Sum Nodes $S$ in $G$

\[
S(x) = \sum_{C \in \text{ch}(P)} w_{s,c} C(x)
\]

Note: We assume that $w_{s,c} \geq 0$ and $\sum_{C \in \text{ch}(P)} w_{s,c} = 1$. 
• \( \psi \) is a function assigning each node \( N \) in the graph a sub-set of \( X \).\(^1\)

A scope function has to fulfil the following properties:

1. If \( N \) is the root node, then \( \psi(N) = X \).
2. If \( N \) is a sum or product, then
   \[
   \psi(N) = \bigcup_{N' \in \text{ch}(N)} \psi(N').
   \]
3. For each \( S \in S \) we have
   \[
   \forall N, N' \in \text{ch}(S): \psi(N) = \psi(N') \text{ (completeness)}.
   \]
4. For each \( P \in P \) we have
   \[
   \forall N, N' \in \text{ch}(P): \psi(N) \cap \psi(N') = \emptyset \text{ (decomposability)}.
   \]

\(^1\) This sub-set is often referred to as the scope of a node.
Completeness can be understood as requiring each sum node to be a well-defined mixture distribution.
Decomposability ensures that each product node is a proper factorisation of its scope. Also, decomposability ensures we can “pull” down expensive operations, such as marginalisation or computation of expectations, down to the leaves enabling tractable inference.
Example SPN $\mathcal{S} = (\mathcal{G}, \psi, w, \theta)$

After applying a scope function $\psi$ on $\mathcal{G}$ we obtain the SPN.

Note: We define that $L(x) := 1$ for every $x$ if and only if $\psi(L) = \emptyset$.

Also note: Most structure learners learn $\mathcal{G}$ and $\psi$ in an entangled way.
Parameter Learning of Sum-Product Networks
Parameter Learning

- The following selection is a small and biased sub-selection of the existing literature.

We will now look into generative learning, supervised (discriminative) learning and semi-supervised learning.
Generative Learning [Poon2011], [Peharz2017]

Goal: Model the target distribution to answer probabilistic queries or generate fake data.

Example:

- \( q: \) How likely is it that there is a traffic jam on Monday on the A2?
- \( q(m) = p_m(\text{Day} = \text{Monday}, \text{JamA2} = 1). \)
Generative Learning [Poon2011], [Peharz2017]

Let $\mathcal{X} = \{x_n\}_{n=1}^N$ with $x_n \in \mathcal{R}^D$ denote training examples and $
phi = (\mathbf{w}, \theta)$ be all parameters.

To learn the parameters, we maximise the log-likelihood (model fit)

$$L(\phi | \mathcal{X}) = \sum_{n=1}^{N} \log S(x_n | \phi)$$

This is usually done using expectation-maximisation or gradient-based optimisation.
Goal: Find a separating hyper-plane given a set of labelled training examples.
Let \( \mathcal{X} = \{x_n\}_{n=1}^{N} \) with \( x_n \in \mathcal{R}^D \) and \( \lambda = \{\lambda_n\}_{n=1}^{N} \) denote training examples and \( \phi = (w, \theta) \) be all parameters.

To learn the parameters, we maximise the conditional log-likelihood:

\[
    L(\phi, \lambda \mid \mathcal{X}) = \sum_{n=1}^{N} \log S(x_n, \lambda_n \mid \phi) - \log S(x_n \mid \phi)
\]

Note: [Peharz2019] proposed a hybrid generative-discriminative loss.
Semi-Supervised Learning [Trapp2017]

Goal: Find a separating hyper-plane given a few labelled and many unlabelled training examples.
Let \( \mathcal{X} = \{x_n\}_{n=1}^N \) with \( x_n \in \mathcal{R}^D \) and \( \lambda = \{\lambda_n\}_{n=1}^N \) denote labelled training examples.

Further, let \( \mathcal{U} = \{u_m\}_{m=1}^M \) with \( u_n \in \mathcal{R}^D \) denote unlabelled training samples and let \( q_m \) denote a soft-label for example \( m \). \( \phi^+ = (\mathbf{w}, \theta) \) denotes the parameters of an SPN trained solely on labelled data \( \mathcal{X}, \lambda \).

\[
\arg\max_{\phi \in \Phi} \arg\min_{q \in \Delta_M^{K-1}} \mathcal{L}(\phi, y, q \mid \mathcal{X}, \mathcal{U}) - \mathcal{L}(\phi^+, y, q \mid \mathcal{X}, \mathcal{U})
\]
Gradient based optimisation of any tree-shaped SPN with small (fixed) learning rate and near-zero initialisation of the weights is equivalent to gradient based optimisation with adaptive and time-varying learning rate and momentum term.
Structure Learning of Sum-Product Networks
There exists a large body of work on heuristics for structure learning.

Most existing approach generate the structure by incrementally optimise some local criterium.
Structure Learners for Images

- [Poon2011] presented a structure learning approach which recursively sub-divides images into sub-regions.
- Later, [Gens2012] proposed a structure that decomposes images into parts.
- Recently, [Peharz2019] presented a randomly constructed structure that divides images into randomly defined sub-regions.
• [Gens2013] proposed one of the most frequently used algorithms, called LearnSPN.

• Later, a wide range of variants of LearnSPN have been proposed to allow structure learning on continuous data or heterogeneous data and to prune the structures in order to improve the generalisation of the structures. See: https://github.com/arranger1044/awesome-spn#structure-learning.

• Recently, we introduced the first principled approach to structure learning by proposing a mathematical framework for Bayesian structure learning [Trapp2019].
LearnSPN constructs $G$ and $\psi$ in a deeply entangled way.

*Figure 1.* A recursive algorithm for learning SPNs.

*Figure 2:* Image adapted based on [Gens2013].
RAT-SPNs use a randomly defined $\psi$ for a pre-defined $G$ encoded using a so-called region graph.

Figure 3: Based on the illustration by [Peharz2019].
Bayesian structure & parameter learning introduces additional latent variables $Y_{P,d}$ for each dimension to encode $\psi$ for a pre-defined $G$. The result is a distribution over SPN structures & parameters.

Figure 4: Generative model for Bayesian structure and parameter learning.
For predictions we approximate the posterior predictive distribution:

\[ p(x^* | X) \approx \sum_{t=1}^{T} p(x^* | w_t, \theta_t, y_t) \] (1)
Applications of Sum-Product Networks
Applications

- Recent applications cover a wide range of fields, e.g. speech, semantic mapping in robotics, cognitive architectures, segmentation, activity-recognition, image-classification.
- See: https://github.com/arranger1044/awesome-spn for an exhaustive list.
- The following few applications are a biased selection.
Out-Of-Domain Detection [Peharz2019]


Figure 5: Image adapted based on [Peharz2019].
SuPAIR integrates SPNs as a subroutine into the Attend-Infer-Repeat (AIR) framework. SuPAIR learns an order of magnitude faster than AIR, treats object occlusions in a consistent manner and allows to define a background model.
Combining Gaussian process (which allow exact inference) with SPNs (which allow exact and efficient inference) enables to derive a non-linear regression model with exact and efficient inference.
Thank you for your attention!
References


