

Sum-Product Networks for Deep Learning

Hoifung Poon **Pedro Domingos**
 Department of Computer Science and Engineering
 University of Washington
 Seattle, WA 98195-2350, U.S.A.
 {hoifung, pedrod}@cs.washington.edu

The key limiting factor in graphical model inference and learning is the complexity of the partition function. We thus ask the question: what are the most general conditions under which the partition function is tractable? The answer leads to a new kind of deep architecture, which we call *sum-product networks (SPNs)* and will present in this abstract.

The key idea of SPNs is to compactly represent the partition function by introducing multiple layers of hidden variables. An SPN is a rooted directed acyclic graph with variables as leaves, sums and products as internal nodes, and weighted edges.¹ Figure 1 shows two examples of SPNs, with \bar{x}_i being the negation of x_i . The root values $S(x)$ for all $x \in \mathcal{X}$ define an unnormalized probability distribution over \mathcal{X} . The unnormalized probability of evidence e is $\Phi_S(e) = \sum_{x \in e} S(x)$ for all x consistent with e . To ensure tractable inference, we want to replace the summation with $S(e)$, which is the root value when the input indicators consistent with e are set to 1 and 0 otherwise. We thus define that an SPN S is *valid* iff $S(e) = \Phi_S(e)$ for all evidence e . A valid SPN computes the probability of evidence in time linear in its size. In particular, the partition function is $Z_S = \sum_{x \in \mathcal{X}} S(x)$ and can be computed by $S(*)$ with all input indicators set to 1. We proved that S is valid if it is *complete* and *consistent*, where completeness means that all children of the same sum node are over the same set of variables, and consistency means that no variable can appear negated in one child of a product node and non-negated in another. Intuitively, completeness ensures that $S(e)$ does not undercount the summation whereas consistency ensures that it does not overcount. Completeness and consistency allow us to design deep architectures where inference is guaranteed to be efficient. This in turn makes learning them much easier.

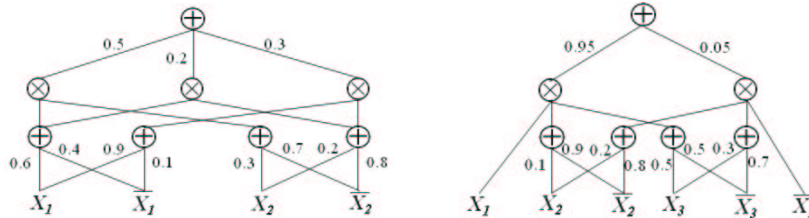


Figure 1: Left: SPN implementing a naive Bayes mixture model (three components, two variables). Right: SPN implementing a junction tree (clusters (X_1, X_2) and (X_1, X_3) , separator X_1).

Essentially all tractable graphical models can be cast as SPNs, but SPNs are also strictly more general. For example, SPNs can be exponentially more compact than hierarchical mixture models [6] because they allow mixtures over subsets of variables and their reuse; SPNs can be exponentially more compact than junction trees [1] when context-specific independence and determinism are present, since they exploit these and junction trees do not. This holds even when junction trees are formed from Bayesian networks with context-specific independence in the form of decision trees at the nodes, because decision trees suffer from the replication problem [8] and can be exponentially larger than a DAG representation of the same function. SPNs can also compactly represent some classes of distributions in which no conditional independences hold, and whose graphical models are therefore completely connected graphs [9].

¹For simplicity, we focus on Boolean variables. The extension to multi-valued discrete variables is straightforward. Continuous variables can be incorporated by assuming mixtures of Gaussians over them.

SPNs lend themselves naturally to efficient computation of the likelihood gradient by backpropagation [10]. Let n_i be an arbitrary node in SPN S , $S_i(x)$ be its value on input instance x , and Pa_i be its parents. If n_i is a product node, its parents (by assumption) are sum nodes, and $\partial S(x)/\partial S_i(x) = \sum_{k \in Pa_i} w_{ki} \cdot \partial S(x)/\partial S_k(x)$. If n_i is a sum node, its parents (by assumption) are product nodes, and $\partial S(x)/\partial S_i(x) = \sum_{k \in Pa_i} (\partial S(x)/\partial S_k(x)) \prod_{l \in Ch_{-i}(k)} S_l(x)$, where $Ch_{-i}(k)$ are the children of the k th parent of n_i excluding n_i . If n_j is a child of sum node n_i , $\partial S(x)/\partial w_{ij} = (\partial P(x)/\partial S_i(x)) S_j(x)$. Maximum-likelihood weights can then be computed by stochastic gradient descent or other methods. SPNs can also be learned using EM [7] by viewing sum nodes as implicit hidden variables. Specifically, we can view a sum node with c children as the result of summing out an implicit hidden variable Y_i with c values y_{ij} , whose prior distribution is $P(Y_i = y_{ij}) \propto w_{ij}$. The marginal distribution of the hidden variables Y given a training instance x can be computed by a downward pass through the SPN similar to backpropagation [2]. Alternatively, we can use hard EM, computing the MAP state of Y given x using a similar upward and downward pass in a network with sums replaced by maxes. The downward pass simply selects the most probable child of each sum node, starting at the root. The M step then increments the count of the winning child and renormalizes to obtain the new weights. The structure of SPNs can be learned by starting with a generic architecture and learning the weights with a sparse prior, with zero weights signifying the absence of connections in the learned structure.

In experiments, we found that standard backpropagation worked poorly for the usual reason in deep networks: the gradient rapidly becomes diluted as more layers are added. Soft EM suffers from a similar problem. But hard EM does not have this problem, because it always produces updates of unit size, and it worked quite well. We evaluated SPNs by applying them to the problem of completing faces. This is a good test for a deep architecture, because it is an extremely difficult task, where detecting deep structure is key. To our knowledge, no previous learner has done well on this problem. We used two well known datasets: Caltech-101 [3] and Olivetti [11]. The SPNs learned in our experiments contained up to 46 layers. Compared to state-of-the-art deep architectures such as DBNs [4, 5], SPNs are simple and require much less engineering, and are at least an order of magnitude faster in inference and learning. Visual inspection shows the SPNs completes faces quite well, including non-symmetric and other difficult cases, while DBNs fail miserably. The mean squared errors of completed pixels confirm the advantage of SPNs over DBNs: 1140 vs. 2386 on Olivetti and 3649 vs. 18527 on Caltech. SPNs also substantially outperform nearest-neighbor.

Much remains to be explored about SPNs, including the extension of SPNs to sequential domains, design principles for SPN architectures, other learning methods for SPNs, and further applications. While graphical models can compactly represent more distributions than SPNs, this potential is hampered by the need of approximate inference. Proper comparison between the two model classes is a fascinating question both theoretically and empirically.

References

- [1] A. Chechotka and C. Guestrin. Efficient principled learning of thin junction trees. In *Proc. NIPS-08*.
- [2] A. Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 2003.
- [3] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples. In *Proc. CVPR Wkshp. on Generative Model-Based Vision*, 2004.
- [4] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.
- [5] H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proc. ICML-09*.
- [6] D. Mimno, W. Li, and A. McCallum. Mixtures of hierarchical topics with Pachinko allocation. In *Proc. ICML-07*.
- [7] R. M. Neal and G. E. Hinton. A new view of the EM algorithm that justifies incremental and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*. 1998.
- [8] G. Pagallo. Learning DNF by decision trees. In *Proc. IJCAI-89*.
- [9] D. Roth and R. Samdani. Learning multi-linear representations of distributions for efficient inference. *Machine Learning*, 76:195–209, 2009.
- [10] D. Rumelhart, G. Hinton, and J. McClelland. Learning internal representations by error propagation. *D. Rumelhart and J. McClelland, editors*, Parallel Distributed Processing, 1:318–362, 1986.
- [11] F. Samaria and A. Harter. Parameterisation of a stochastic model for human face identification. In *Proc. 2nd IEEE Wkshp. on Applications of Computer Vision*, 1994.

Topic: learning algorithms **Preference:** oral