Sum-Product Networks: A New Deep Architecture

Hoifung Poon
Dept. Computer Science & Eng.
University of Washington
Seattle, WA 98195-2350
hoifung@cs.washington.edu

Pedro Domingos
Dept. Computer Science & Eng.
University of Washington
Seattle, WA 98195-2350
pedrod@cs.washington.edu

Abstract

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)*. SPNs are directed acyclic graphs with variables as leaves, sums and products as internal nodes, and weighted edges. We show that if an SPN is *complete* and *consistent* it represents the partition function and all marginals of some graphical model, and give semantics to its nodes. Essentially all tractable graphical models can be cast as SPNs, but SPNs are also strictly more general. We then propose learning algorithms for SPNs, based on backpropagation and EM. Experiments show that inference and learning with SPNs can be both faster and more accurate than with standard deep networks. For example, SPNs perform face completion better than state-of-the-art deep networks for this task. SPNs also have intriguing potential connections to the architecture of the cortex.

1 Introduction

The goal of probabilistic modeling is to represent probability distributions compactly, compute their marginals and modes efficiently, and learn them accurately. Graphical models [19] represent distributions compactly as normalized products of factors: $P(X=x) = \frac{1}{Z}\prod_k \phi_k(x_{\{k\}})$, where $x \in \mathcal{X}$ is a d-dimensional vector, each *potential* ϕ_k is a function of a subset $x_{\{k\}}$ of the variables (its scope), and $Z = \sum_{x\in\mathcal{X}} \prod_k \phi_k(x_{\{k\}})$ is the *partition function*. Graphical models have a number of important limitations. First, there are many distributions that admit a compact representation, but not in the form above. (For example, the uniform distribution over vectors with an even number of 1's.) Second, inference is still exponential in the worst case. Third, the sample size required for accurate learning is worst-case exponential in scope size. Fourth, because learning requires inference as a subroutine, it can take exponential time even with fixed scopes (unless the partition function is a known constant, which requires restricting the potentials to be conditional probabilities).

The compactness of graphical models can often be greatly increased by postulating the existence of *hidden variables* y: $P(X=x) = \frac{1}{Z}\sum_y \prod_k \phi_k((x,y)_{\{k\}})$. Deep architectures [1] can be viewed as graphical models with multiple layers of hidden variables, where each potential involves only variables in consecutive layers, or variables in the shallowest layer and x. Many distributions can only be represented compactly by deep networks. However, the combination of non-convex likelihood and intractable inference makes learning deep networks extremely challenging. Classes of graphical models where inference is tractable exist (e.g., mixture models [13], thin junction trees [2]), but are quite limited in the distributions they can represent compactly. This paper starts from the observation that *models with multiple layers of hidden variables allow for efficient inference in a much larger class of distributions*. Surprisingly, current deep architectures do not take advantage of this, and typically solve a harder inference problem than models with one or no hidden layers.

This can be seen as follows. The partition function Z is intractable because it is the sum of an exponential number of terms. All marginals are sums of subsets of these terms; thus if Z can be computed efficiently, so can they. But Z itself is a function that can potentially be compactly represented using a deep architecture. Z is computed using only two types of operations: sums and products. It can be computed efficiently iff $\sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$ can be reorganized using the distributive law into a computation involving only a polynomial number of sums and products. Given a graphical model, the inference problem in a nutshell is to perform this reorganization. But we can instead learn from the outset a model that is already in efficiently computable form, viewing sums as implicit hidden variables. This leads naturally to the question: what is the broadest class of models that admit such an efficient form for Z?

We answer this question by providing conditions for tractability of Z, and showing that they are more general than previous tractable classes. We introduce *sum-product networks (SPNs)*, a representation that facilitates this treatment and also has semantic value in its own right. SPNs can be viewed as generalized directed acyclic graphs of mixture models, with sum nodes corresponding to mixtures over subsets of variables and product nodes corresponding to features or mixture components. SPNs lend themselves naturally to efficient learning by backpropagation or EM. Of course, many distributions cannot be represented by polynomial-sized SPNs, and whether these are sufficient for the real-world problems we need to solve is an empirical question. Our experiments show they are quite promising.

2 Sum-Product Networks

For simplicity, we focus on Boolean variables. The extension to multi-valued discrete variables is straightforward. The extension to continuous variables is an item for future work. The negation of a Boolean variable X_i is represented by $\bar{X}_i$. The indicator function $[.]$ has value 1 when its argument is true, and 0 otherwise. Since it will be clear from context whether we are referring to a variable or its indicator, we abbreviate $[X_i]$ by x_i and $[\bar{X}_i]$ by $\bar{x}_i$.

We build on the ideas of Darwiche [4], and in particular the notion of *network polynomial*. Let $\Phi(x) \geq 0$ be an unnormalized probability distribution. The network polynomial of $\Phi(x)$ is $\sum_x \Phi(x)\Pi(x)$, where $\Pi(x)$ is the product of the indicators that have value 1 in state x. For example, the network polynomial for a Bernoulli distribution over variable X_i with parameter p is $px_i + (1-p)\bar{x}_i$. The network polynomial is a multilinear function of the indicator variables. The unnormalized probability of evidence (partial instantiation of X) e is the value of the network polynomial when all indicators compatible with e are set to 1 and the remainder are set to 0. For 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. Of course, the network polynomial has size exponential in the number of variables, but we may be able to represent and evaluate it in polynomial space and time using a *sum-product network*.

Definition 1 *A* sum-product network (SPN) *over variables $x_1, \ldots, x_d$ is a rooted directed acyclic graph whose leaves are the indicators $x_1, \ldots, x_d$ and $\bar{x}_1, \ldots, \bar{x}_d$ and whose internal nodes are sums and products. Each edge (i, j) emanating from a sum node i has a non-negative weight w_{ij}. The value of a product node is the product of the values of its children. The value of a sum node is $\sum_{j \in Ch(i)} w_{ij} x_j$, where $Ch(i)$ are the children of i. The value of an SPN is the value of its root.*

Figures 1 show examples of SPNs. In this paper we will assume (without loss of generality) that sums and products are arranged in alternating layers, i.e., all children of a sum are products or leaves, and vice-versa.

We denote the sum-product network S as a function of the indicator variables $x_1, \ldots, x_d$ and $\bar{x}_1, \ldots, \bar{x}_d$ by $S(x_1, \ldots, x_d, \bar{x}_1, \ldots, \bar{x}_d)$. When the indicators specify a complete state x (i.e., for each variable X_i, either $x_i = 1$ and $\bar{x}_i = 0$ or $x_i = 0$ and $\bar{x}_i = 1$), we abbreviate this as $S(x)$. When the indicators specify evidence e we abbreviate it as $S(e)$. When all indicators are set to 1, we abbreviate it as $S(*)$. The subnetwork rooted at an arbitrary node n in the SPN is itself an SPN, which we denote by $S_n(.)$. The values of $S(x)$ for all $x \in \mathcal{X}$ define an unnormalized probability distribution over $\mathcal{X}$. The unnormalized probability of evidence e under this distribution is $\Phi_S(e) = \sum_{x \in e} S(x)$,

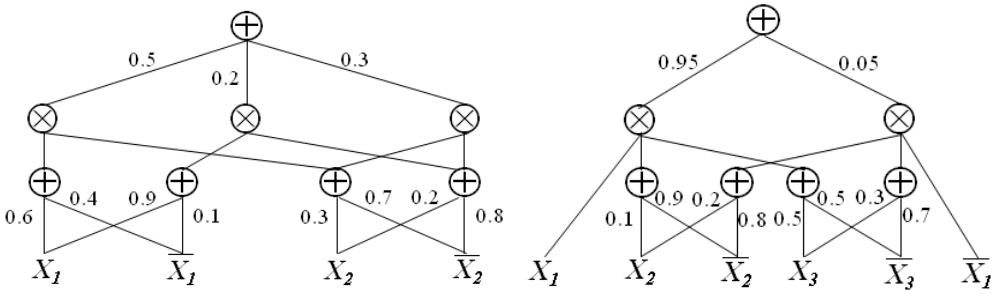

Figure 1: Left: SPN implementing a naive Bayes mixture model (three components, two variables). Right: SPN implementing a junction tree (clusters (X_1, X_2) and (X_1, X_3), separator X_1).

where the sum is over states consistent with e. The partition function of the distribution defined by $S(x)$ is $Z_S = \sum_{x \in \mathcal{X}} S(x)$. The scope of an SPN S is the set of variables that appear in S. A variable X_i appears negated in S if $\bar{x}_i$ is a leaf of S and non-negated if x_i is a leaf of S.

Definition 2 *A sum-product network S is* valid *iff $S(e) = \Phi_S(e)$ for all evidence e.*

In other words, an SPN is valid if it always correctly computes the probability of evidence. In particular, if an SPN S is valid then $S(*) = Z_S$. A valid SPN computes the probability of evidence in time linear in its size. We would like to learn only valid SPNs, but otherwise have as much flexibility as possible. We thus start by establishing general conditions for the validity of an SPN.

Definition 3 *A sum-product network is* complete *iff all children of the same sum node have the same scope.*

Definition 4 *A sum-product network is* consistent *iff no variable appears negated in one child of a product node and non-negated in another.*

Theorem 1 *A sum-product network is valid if it is complete and consistent.*

Proof. Every SPN S can be expressed as a polynomial $\prod_k s_k \Pi_k(\ldots)$, where $\Pi_k(\ldots)$ is a monomial over the indicator variables and $s_k \geq 0$ is its coefficient. We call this the *expansion* of the SPN; it is obtained by applying the distributive law bottom-up to all product nodes in the SPN, treating each x_i leaf as $1x_i + 0\bar{x}_i$ and each $\bar{x}_i$ leaf as $0x_i + 1\bar{x}_i$. An SPN is valid if its expansion is its network polynomial, i.e., the monomials in the expansion and the states x are in one-to-one correspondence: each monomial is non-zero in exactly one state (condition 1), and each state has exactly one monomial that is non-zero in it (condition 2). From condition 2, $S(x)$ is equal to the coefficient s_x of the monomial that is non-zero in it, and therefore $\Phi_S(e) = \sum_{x \in e} S(x) = \sum_{x \in e} s_x = \sum_k s_k n_k(e)$, where $n_k(e)$ is the number of states x consistent with e for which $\Pi_k(x) = 1$. From condition 1, $n_k = 1$ if the state x for which $\Pi_k(x) = 1$ is consistent with the evidence and $n_k = 0$ otherwise, and therefore $\Phi_S(e) = \sum_{k\,:\,\Pi_k(e)=1} s_k = S(e)$ and the SPN is valid.

We prove by induction from the leaves to the root that, if an SPN is complete and consistent, then its expansion is its network polynomial. This is trivially true for a leaf. We consider only internal nodes with two children; the extension to the general case is immediate. Let n^0 be an arbitrary internal node with children n^1 and n^2. We denote the scope of n^0 by V^0, a state of V^0 by x^0, the expansion of the subgraph rooted at n^0 by S^0, and the unnormalized probability of x_0 under S_0 by $\Phi^0(x^0)$; and similarly for n^1 and n^2. By the induction hypothesis, $S^1 = \sum_{x^1} \Phi^1(x^1)\Pi(x^1)$ and $S^2 = \sum_{x^2} \Phi^2(x^2)\Pi(x^2)$.

If n^0 is a sum node, then $S^0 = w_{01} \sum_{x^1} \Phi^1(x^1)\Pi(x^1) + w_{02} \sum_{x^2} \Phi^2(x^2)\Pi(x^2)$. If $V^1 \neq V^2$, then each state of V^1 (or V^2) corresponds to multiple states of $V^0 = V^1 \cup V^2$, and therefore each monomial from V^1 (V^2) is non-zero in more than one state of V^0, breaking the correspondence between monomials of S^0 and states of V^0. However, if the SPN is complete then $V^0 = V^1 = V^2$, and their states are in one-to-one correspondence. Therefore by the induction hypothesis the monomials of V^1 and V^2 are also in one-to-one correspondence and $S^0 = \sum_{x^0}(w_{01}\Phi^1(x^0) + w_{02}\Phi^2(x^0))\Pi(x^0)$; i.e., the expansion of S^0 is its network polynomial.

If n^0 is a product node, then $S^0 = \left(\sum_{x^1} \Phi^1(x^1)\Pi(x^1)\right)\left(\sum_{x^2} \Phi^2(x^2)\Pi(x^2)\right)$. If $V_1 \cap V_2 = \emptyset$, it follows immediately that the expansion of V_0 is its network polynomial. In the more general case, let $V_{12} = V_1 \cap V_2$, $V_{1-} = V_1 \setminus V_2$ and $V_{2-} = V_2 \setminus V_1$ and let x^{12}, x^{1-} and x^{2-} be the corresponding states. Since each $\Phi^1(x^1)$ is non-zero in exactly one state x^1 and similarly for $\Phi^2(x^2)$, each monomial in the product of S^1 and S^2 is nonzero in at most one state of $V^0 = V^1 \cup V^2$. If the SPN is not consistent, then at least one monomial in the product contains both the positive and negative indicators of a variable, x_i and $\bar{x}_i$. Since no monomial in the network polynomial contains both x_i and $\bar{x}_i$, this means the expansion of S^0 is not equal to its network polynomial. To ensure that each monomial in S^0 is non-zero in at least one state of V_0, for every $\Pi(x^{1-}, x^{12})$, $\Pi(x^{12}, x^{2-})$ pair there must exist a state $x^0 = (x^{1-}, x^{12}, x^{2-})$ where both $\Pi(x^{1-}, x^{12})$ and $\Pi(x^{12}, x^{2-})$ are 1, and therefore the indicators over x^{12} in both monomials must be consistent. Since by the induction hypothesis they completely specify x^{12}, they must be the same in the two monomials. Therefore all $\Pi(x^{1-}, x^{12})$ and $\Pi(x^{12}, x^{2-})$ monomials must have the same x^{12} indicators, i.e., the SPN must be consistent. □

Completeness and consistency are not necessary for validity; for example, the network $S(x_1, x_2, \bar{x}_1, \bar{x}_2) = \frac{1}{2}x_1x_2\bar{x}_2 + \frac{1}{2}x_1$ is incomplete and inconsistent, but satisfies $\Phi_S(e) = \sum_{x \in e} S(x)$ for all evidence e. However, completeness and consistency are necessary for the stronger property that every subnetwork of S be valid.

Completeness and consistency allow us to design deep architectures where inference is guaranteed to be efficient. This in turn makes learning them much easier.

Definition 5 *A sum-product network is* decomposable *iff no variable appears in more than one child of a product node.*

If an SPN S is complete and consistent and, for each sum node i, $\sum_{j \in Ch(i)} w_{ij} = 1$, where $Ch(i)$ are the children of i, then $Z_S = 1$. We can view a sum node with c children whose weights sum to 1 as the result of summing out an implicit hidden variable Y_i with c values y_{ij}, whose prior distribution is $P(Y_i = y_{ij}) = w_{ij}$. If the network is decomposable, the subnetwork rooted at the jth child then represents the distribution of the variables in it conditioned on $Y_i = y_{ij}$.

If an SPN S is complete but inconsistent, its expansion includes monomials that are not present in its network polynomial, and $S(e) \geq \Phi_S(e)$. If S is consistent but incomplete, some of its monomials are missing indicators relative to the monomials in its network polynomial, and $S(e) \leq \Phi_S(e)$. Thus invalid SPNs may be useful for approximate inference. Exploring this is a direction for future work.

Definition 6 *An unnormalized probability distribution $\Phi(x)$ is* representable *by a sum-product network S iff $\Phi(x) = S(x)$ for all states x and S is valid.*

S then correctly computes all marginals of $\Phi(x)$, including its partition function.

Theorem 2 *The partition function of a Markov network $\Phi(x)$, where x is a d-dimensional vector, can be computed in time polynomial in d if $\Phi(x)$ is representable by a sum-product network with a number of edges polynomial in d.*

Proof. Follows immediately from the definitions of SPN and representability. □

3 Sum-Product Networks and Other Models

Let $R_M(D)$ be the most compact representation of distribution D under moder class M, $\text{size}(R)$ be the size of representation R, $c > 0$ be a constant, and $\exp(x)$ be an exponential function. We say that model class M_1 is more general than model class M_2 iff for all distributions D $\text{size}(R_{M_1}(D)) \leq c \cdot \text{size}(R_{M_2}(D))$ and there exist distributions for which $\text{size}(R_{M_2}(D)) \geq \exp(\text{size}(R_{M_1}(D)))$. In this sense, sum-product networks are more general than both hierarchical mixture models [16] and thin junction trees [2]. Clearly, both of these can be represented as SPNs without loss of compactness. SPNs can be exponentially more compact than hierarchical mixture models because they allow mixtures over subsets of variables and their reuse. SPNs can be exponentially more compact than junction trees 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 [18] and can be exponentially larger than a DAG representation of the same function.

Graphical models with junction tree clique potentials that cannot be simplified to polynomial size cannot be represented compactly as SPNs. More interestingly, SPNs can compactly represent some classes of distributions in which no conditional independences hold, and whose graphical models are therefore completely connected graphs. Roth and Samdani [21] show this for multi-linear representations, which are essentially expanded SPNs. An SPN can thus be exponentially more compact than the corresponding multi-linear representation.

SPNs are closely related to data structures for efficient inference like arithmetic circuits [4] and AND/OR graphs [5]. However, to date these have been viewed purely as compilation targets for Bayesian network inference and related tasks, and have no semantics as models in their own right. As a result, the problem of learning them has not been considered. The only exception we are aware of is Lowd and Domingos [14]; their algorithm is a standard Bayesian network structure learner with the complexity of the resulting circuit as the regularizer, and does not have the flexibility of SPN learning. Case-factor diagrams [15] are another compact representation, similar to decomposable SPNs. No algorithms for learning them or for computing the probability of evidence in them have been proposed to date.

We can view the product nodes in an SPN as forming a feature hierarchy, with the sum nodes representing distributions over them; in contrast, standard deep architectures explicitly represent only the features, and require the sums to be inefficiently computed by Gibbs sampling or otherwise approximated. Convolutional networks [11] alternate feature layers with pooling layers, where the pooling operation is typically max or average, and the features in each layer are over a subset of the input variables. Convolutional networks are not probabilistic, and are usually viewed as a vision-specific architecture. SPNs can be viewed as probabilistic, general-purpose convolutional networks, with average-pooling corresponding to marginal inference and max-pooling corresponding to MAP inference (see next section). Lee at al. [12] have proposed a probabilistic version of max-pooling, but in their architecture there is no correspondence between pooling and the sum or max operations in probabilistic inference, as a result of which inference is generally intractable. SPNs can also be viewed as a probabilistic version of competitive learning [24] and sigma-pi networks [22]. Like deep belief networks, SPNs can be used for nonlinear dimensionality reduction [10], and allow objects to be reconstructed from the reduced representation (in the case of SPNs, a choice of mixture component at each sum node).

Probabilistic context-free grammars and statistical parsing [3] can be straightforwardly implemented as decomposable SPNs, with non-terminal nodes corresponding to sums (or maxes) and productions corresponding to products (logical conjunctions for standard PCFGs, and general products for head-driven PCFGs). Learning an SPN then amounts to directly learning a chart parser of bounded size. However, SPNs are more general, and can represent unrestricted probabilistic grammars with bounded recursion. SPNs are also well suited to implementing and learning grammatical vision models (e.g., [7, 26]).

4 Learning Sum-Product Networks

SPNs lend themselves naturally to efficient computation of the likelihood gradient by backpropagation [23]. 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 $\partial S(x)/\partial S_i(x) = \sum_{k \in Pa_i} \partial S(x)/\partial S_k(x)$. 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 kth 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. In particular, quasi-Newton and conjugate gradient methods can be applied without the difficulties introduced by approximate inference.

SPNs can also be learned using EM [17] by viewing sum nodes as implicit hidden variables, as described above. 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 [4]. Alternatively, we

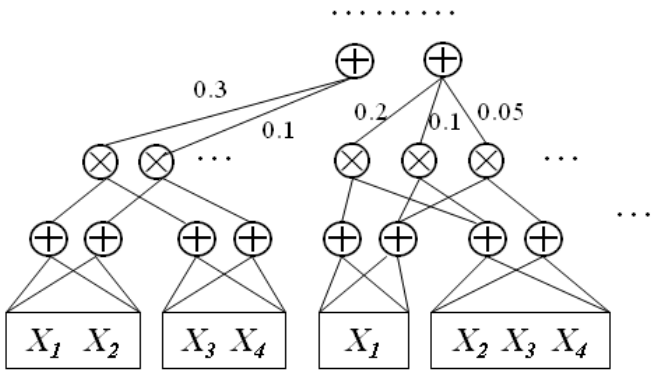

Figure 2: SPN architecture used in the experiments.

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.

Similar considerations apply to MAP learning, with (for example) a Dirichlet prior on the weights at each sum node.

5 Experiments

We evaluated SPNs by applying them to 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 [6] and Olivetti [25]. Caltech-101 contains 435 faces of various sizes; we rescaled the longer sides to 100. Olivetti contains 400 64×64 faces. For each dataset, we set aside 50 test images and used the rest for training. We used online learning with mini-batches; processing of instances in a batch can be trivially parallelized. We initialized all weights to zero and used an L_0 prior (i.e., each non-zero weight incurs a penalty of α). We used add-one smoothing when evaluating nodes.

In preliminary 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 the same problem, because the updates also become smaller and smaller as we go deeper. But hard EM does not have this problem, because it always produces updates of unit size, and it worked quite well. Running soft EM after hard EM yielded no improvement. The best results were obtained using sums on the upward pass and maxes on the downward pass (i.e., the MAP value of each hidden variable is computed conditioning on the MAP values of the hidden variables above it and summing out the ones below).

SPNs admit a very flexible choice of architectures, and can easily incorporate domain knowledge (e.g., invariances like translation or symmetry). In this paper, we used a simple yet very general architecture. For each non-unit rectangular region R, the SPN contains a fixed number k of sum nodes (except for the region of the entire image, which contains only the root). For every way to divide R into two rectangular subregions R_1 and R_2, and for every pair of sum nodes n_1 and n_2 from R_1 and R_2, respectively, there is a product node that has n_1 and n_2 as its children and is the child of all sum nodes in R. Figure 2 shows this architecture.

SPNs can also adopt multiple resolution levels. For example, for large regions we may only consider coarse region decompositions. In preliminary experiments, we found that this made learning much faster with little degradation in accuracy. In particular, we adopted an architecture that uses decompositions at a coarse resolution of m-by-m for large regions, and finer decompositions only inside each m-by-m block. We set m to 4 for Olivetti and 5 for Caltech.

Figure 3: Face completion results. From top to bottom: original, SPN, DBN, nearest neighbor. The first six images are from the Olivetti dataset, and the rest from Caltech-101.

The SPNs learned in our experiments were very deep: 36 layers for Olivetti, 46 for Caltech faces, and 54 for MNIST. In general, in our architecture there are $2(d-1)$ layers between the root and input for $d \times d$ images. The numbers for SPNs with multiple resolution levels can be computed similarly.

To handle gray-scale intensities, we scaled the intensities into $[0, 1]$ and treated each pixel variable X_i as a sample from a Gaussian mixture with k unit-variance components. For each pixel, the intensities of training examples are divided into k equal quantiles and the mean of each component is set to that of the corresponding quantile. We used four components in our experiments. (We also tried using more components and learning the mixing parameters, but it yielded no improvement in performance.)

Figure 3 shows the results for completing the left halves of previously unseen faces. (The complete set of results and the SPN code will be available for download in an online appendix.) Overall, the SPN successfully completed most faces by hypothesizing the correct locations and types of various parts like hair, eye, mouth, and face shape. On the other hand, the SPN also has some weaknesses. For example, there are often blocks in the completion. Also, completions for Caltech-101 faces are significantly worse. This is not surprising, since Caltech-101 faces contain many more variations in position, background and lighting conditions.

Compared to state-of-the-art deep architectures [10, 12], we found that SPNs have three significant advantages. First, SPNs are simple and require much less engineering. In preliminary experiments, we set $\alpha = 1$ and found that these values worked well for multiple datasets in both face completion and digit recognition. We also used the same architecture throughout and let learning adapt the SPN to the details of each dataset. In contrast, DBNs typically require a careful choice of parameters and architectures for each dataset. (For example, the learning rate of 0.1 that works well in digit recognition leads to massive oscillation when learning faces.) SPN learning terminates when the average log-likelihood does not improve beyond a threshold (we used 0.1, which typically converges in around 10 iterations; 0.01 yielded no improvement in initial experiments). For DBNs, however, the number of iterations has to be determined empirically using a large development set. Further, successful DBN training often requires extensive preprocessing of the examples, while we used essentially none for SPNs.

Second, SPNs are at least an order of magnitude faster in both learning and inference. For example, learning in Olivetti takes about 6 minutes with 20 cores, or about 2 hours with one CPU. In contrast, DBNs took 12 hours just to complete the recommended 200 iterations of pretraining, and another 30 hours for the subsequent fine-tuning stage. For inference, SPNs took less than a second to find the MAP completion of a face, to compute the likelihood of such a completion, or to compute the marginal probability of a variable, and all these results are exact. In contrast, estimating likelihood in DBNs is a difficult open problem; estimating marginals requires many Gibbs sampling steps that may take minutes or even hours, and the results are approximate with no quality guarantees.

Third, SPNs appear to learn much more effectively. For example, Lee et al. [12] show five faces with completion results in Figure 6 of their paper. Their network was only able to complete a small portion of the faces, leaving the rest blank (starting with images where the visible side already

contained more than half the face).[1] We were not able to make DBNs work very well for face completion using the code from Hinton and Salakhutdinov [10]. Despite several trials with different learning parameters and very long training time, the learned DBN only output white noise for the left half on Olivetti, and all black on Caltech-101. Hinton and Salakhutdinov [10] reported results for dimension reduction on Olivetti faces, but they used a training set containing over 120,000 reduced-scale images derived via transformations like rotation, scaling, etc. With their learned model, the results on Olivetti improve but are still very blurry.[2]

We also compared SPNs with nearest neighbor, which has been shown to be very effective for scene completion [9]. Specifically, for each test image, we found the training image with most similar right half using Euclidean distance, and returned its left half. While this gives very good completion if there is a very similar training image, in general nearest neighbor tend to produce very bad completions.

SPNs greatly outperform both DBNs and nearest neighbor (NN) on mean squared errors of completed pixels: 1140 vs. 2386 (DBN) and 1527 (NN) on Olivetti, and 3649 vs. 18527 (DBN) and 4243 (NN) on Caltech-101.

We also applied SPNs to classifying MNIST digits by learning a separate SPN for each digit and choosing the one with highest probability at test time. This achieved 96% accuracy, which, while not state-of-the-art, is noteworthy for a simple generative approach.

6 Sum-Product Networks and the Cortex

The cortex is composed of two main types of cells: pyramidal neurons and stellate neurons. Pyramidal neurons excite the neurons they connect to; most stellate neurons inhibit them. There is an interesting analogy between these two types of neurons and the nodes in SPNs, particularly when MAP inference is used. In this case the network is composed of max nodes and sum nodes (logs of products). (Cf. Riesenhuber and Poggio [20], which also uses max and sum nodes, but is not a probabilistic model.) Max nodes are analogous to inhibitory neurons in that they select the highest input for further propagation. Sum nodes are analogous to excitatory neurons in that they compute a sum of their inputs. In SPNs the weights are at the inputs of max nodes, while the analogy with the cortex suggests having them at the inputs of sum (log product) nodes. One can be mapped to the other if we let max nodes ignore their children's weights and consider only their values. Possible justifications for this include: (a) it potentially reduces computational cost by allowing max nodes to be merged; (b) ignoring priors may improve discriminative performance [8]; (c) priors may be approximately encoded by the number of units representing the same pattern, and this may facilitate online hard EM learning. Unlike SPNs, the cortex has no single root node, but it is straighforward to extend SPNs to have multiple roots, corresponding to simultaneously computing multiple distributions with shared structure. Of course, SPNs are still biologically unrealistic in many ways, but they may nevertheless provide an interesting addition to the computational neuroscience toolkit.

7 Conclusion

Sum-product networks (SPNs) are DAGs of sums and products that efficiently compute partition functions and marginals of high-dimensional distributions, and can be learned by backpropagation and EM. SPNs can be viewed as a deep combination of mixture models and feature hierarchies. Inference in SPNs is faster and more accurate than in previous deep architectures. This in turn makes learning faster and more accurate. Our experiments indicate that, because of their robustness, SPNs require much less manual engineering than other deep architectures. Much remains to be explored, including the extension of SPNs to continuous and sequential domains, design principles for SPN architectures, other learning methods for SPNs, and further applications.

[1] We were unable to obtain their code for head-to-head comparison. We should note that the main purpose of the figure is to illustrate the importance of top-down inference.

[2] We adopted the same preprocessing step as the authors by normalizing the intensity of input image to have zero mean and unit variance. We reported results after running 100 mean-field iterations. The results were similar with fewer or more iterations.

8 Acknowledgments

We thank Ruslan Salakhutdinov for help in experiments with DBNs. This research was partly funded by ARO grant W911NF-08-1-0242, AFRL contract FA8750-09-C-0181, NSF grant IIS-0803481, and ONR grant N00014-08-1-0670. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, AFRL, NSF, ONR, or the United States Government.

References

[1] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2:1–127, 2009.

[2] A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Adv. NIPS 20*, pages 273–280. MIT Press, 2008.

[3] M. Collins. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29:589–637, 2003.

[4] A. Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50:280–305, 2003.

[5] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171:73–106, 2006.

[6] 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.

[7] P. Felzenszwalb and D. McAllester. Object detection grammars. Technical Report TR-2010-02, Dept. Computer Science, University of Chicago, 2010.

[8] J. H. Friedman. On bias, variance, 0/1 loss, and the curse of dimensionality. *Data Mining and Knowledge Discovery*, 1:55–77, 1997.

[9] J. Hays and A. Efros. Scene completion using millions of photographs. In *Proc. SIGGRAPH-07*, 2007.

[10] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

[11] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[12] H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proc. ICML-09*, pages 609–616, 2009.

[13] D. Lowd and P. Domingos. Naive Bayes models for probability estimation. In *Proc. ICML-05*, pages 529–536, 2005.

[14] D. Lowd and P. Domingos. Learning arithmetic circuits. In *Proc. UAI-08*, pages 383–392, 2008.

[15] D. McAllester, M. Collins, and F. Pereira. Case-factor diagrams for structured probabilistic modeling. In *Proc. UAI-04*, pages 382–391, 2004.

[16] D. Mimno, W. Li, and A. McCallum. Mixtures of hierarchical topics with Pachinko allocation. In *Proc. ICML-07*, pages 633–640, 2007.

[17] R. Neal and G. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer, 1998.

[18] G. Pagallo. Learning DNF by decision trees. In *Proc. IJCAI-89*, pages 639–644, 1989.

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[20] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 1999.

[21] D. Roth and R. Samdani. Learning multi-linear representations of distributions for efficient inference. *Machine Learning*, 76:195–209, 2009.

[22] D. Rumelhart, G. Hinton, and J. McClelland. A general framework for parallel distributed processing. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 45–76. MIT Press, 1986.

[23] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

[24] D. Rumelhart and D. Zipser. Feature discovery by competitive learning. In D. E. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 151–193. MIT Press, 1986.

[25] 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.

[26] L. Zhu, Y. Chen, and A. Yuille. Unsupervised learning of probabilistic grammar-Markov models for object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:1–15, 2009.