Non-Parametric Bayesian Sum-Product Networks

Sang-Woo Lee SLEE@BI.SNU.AC.KR
School of Computer Sci. and Eng., Seoul National University, Seoul 151-744, Korea

Christopher J. Watkins C.J.WATKINS@RHUL.AC.UK
Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK

Byoung-Tak Zhang BTZHANG@BI.SNU.AC.KR
School of Computer Sci. and Eng., Seoul National University, Seoul 151-744, Korea

Abstract

We define two non-parametric models for Sum-Product Networks (SPNs) (Poon & Domingos, 2011). The first is a tree structure of Dirichlet Processes; the second is a dag of hierarchical Dirichlet Processes. These generative models for data implicitly define a prior distribution on SPN of tree and of dag structure. They allow MCMC fitting of data to SPN models, and the learning of SPN structure from data.

1. Introduction

We describe two non-parametric Bayesian models for sum-product networks (SPNs). SPNs are tractable probabilistic graphical models and a new deep architecture (Poon & Domingos, 2011). It is claimed that SPNs show remarkable performance for computer vision, including image classification tasks (Gens & Domingos, 2012) and video learning (Amer & Todorovic, 2012), and it is reported (Gens & Domingos, 2013; Rooshenas & Lowd, 2014) that SPNs outperformed other probabilistic graphical models in variable retrieval problems. SPNs have a specialized structure for fast inference. They are constrained to have rooted directed acyclic graphs, and their leaves represent univariate distributions such as the multinomial distribution for discrete variables, and the Gaussian distribution for continuous variables. Internal nodes in the structure are either 'product-nodes' or 'sum-nodes'.

Definition 1: SPNs are defined inductively as follows (Poon & Domingos, 2011):

Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

1. A tractable univariate distribution is an SPN.
2. A product of SPNs with disjoint scopes (i.e. variable sets) is an SPN.
3. A weighted sum of SPNs with the same scope is an SPN, provided that all weights are positive and sum to 1.
4. Nothing else is an SPN.

First, let us define the probability distribution corresponding to an SPN. Let there be D variables $X_1, \ldots, X_D$; the SPN specifies a joint probability distribution over these variables. We assume throughout that $D \geq 2$. Each node in a SPN defines a joint probability distribution over a subset of the variables; we refer to this subset of the variables as the *scope* of the sum or product node. We use s to denote a scope as a subset of indices; in other words a scope with k indices is $s = \{s_1, \ldots, s_k\} \subseteq \{1, \ldots, D\}$. The indexed set of the corresponding variables is written $X_s = (X_{s_1}, \ldots, X_{s_k})$. The number of elements in the scope of a node is termed the *level* of the node. In this paper we consider only binary data. Each node of level 1 (a univariate node) defines a Bernoulli distribution for the (binary) variable that is its scope. Univariate nodes have no children. We require that each product node and each sum node must have level at least 2.[1] We write P_s and S_s to denote a product or sum node respectively with scope s, where $s \subseteq \{1, \ldots, D\}$, and write children($P_s$) and children($S{-}s$) for the (indexed) sets of children of product and sum nodes.

[1] For non-binary variables – for example, for real variables – it could be meaningful to have sum-nodes of level one, which would represent mixtures of univariate distributions, but this is not useful for binary data considered here.

A **product-node** defines the p.d. over its scope that is the product of the p.d.s of its children, which are sum-nodes or univariate nodes, with disjoint scopes. Each product node P defines a partition of its scope; we require this partition to have at least two elements. For a product node P of level k, let

$$\text{partitition}(\mathrm{P}) = (s_1, \ldots, s_k)$$
$$\text{children}(\mathrm{P}_s) = (S_{s_1}, \ldots, S_{s_k})$$

where the s_i are non-empty, disjoint, and $s_1 \cup \cdots \cup s_k = \{1, \ldots, D\}$,and the children $(\mathrm{S}_{s_1}, \ldots, \mathrm{S}_{s_k})$ are sum nodes or univariate nodes. If $|s_i|$, which is level(S_{s_i}), is equal to 1, then S_{s_i} is a univariate distribution node, otherwise S_{s_i} is a sum-node. Each product node P only has $|\text{partitition}(P)|$ child nodes.

The probability distribution denoted by a product note P is denoted $P(\cdot)$; and the p.d. of a sum-node or univariate node S as $S(\cdot)$. Then product node P_s with children $(S_1, \ldots, S_k)$ defines a joint p.d. over X_s, which is

$$P_s(X_s) = \prod_{i=1}^{k} S(X_{s_i})$$

Note that a product node always has D or fewer children, because the scopes of its children are required to be disjoint.

A **sum-node** is either a univariate node, or else it defines a mixture distribution over its children, which are product-nodes, all with the same scope. In our model, each sum-node S_s may have a countable infinity of children $(P_s^1, P_s^2, P_s^3, \ldots)$ For each child there is a corresponding non-negative weight $w_1, w_2, w_3, \ldots$, such that $\sum_{i=1}^{\infty} w_i = 1$. The distribution defined S_s is:

$$S_s(X_s) = \sum_{i=1}^{\infty} w_i P_s^i(X_s)$$

The top node of the SPN is the sum-node $S_{\{1,\ldots,D\}}$, which we also denote S_{top}.

2. A prior distribution for SPN trees

We describe two non-parametric generative models for multivariate data, which indirectly specify non-parametric prior distributions over SPNs. In the first model, the SPN is always a tree, in that no product node is ever a child of two sum-nodes, and no sum-node is ever a child of two product nodes. In the second model, the SPN is a dag, in that every product node is potentially the child of many sum-nodes, but no sum-node is ever the child of two product nodes. The first model is simpler; the second model is more general. The prior distributions are defined recursively, node by node.

In both the tree and the dag models, the prior distribution of each sum-node S_s is a Dirichlet process (Teh et al., 2004), with concentration parameter α_s and base distribution denoted by $\mathrm{G_P}(s)$. The base distribution $\mathrm{G_P}(s)$ is a probability distribution over the set of possible product nodes with scope s. The distribution $\mathrm{G_P}(s)$ is a basic input to the model which expresses prior beliefs about the form of product nodes. In principle, any distribution over product nodes could be used, but a simple and in some sense elegant choice is specify $\mathrm{G_P}(s)$ as a probability distribution over the partitions of s. Only the partition of s is chosen at this level; the prior distribution for each child sum-node of the product node is defined recursively, in the same manner, at its own level. if the partition generated contains singletons, $\mathrm{G_P}(s)$ must also specify the a p.d. over the possible univariate distributions; for binary data, a natural choice is a beta distribution parametrised as a vague prior, such as Beta$(\frac{1}{2}, \frac{1}{2})$.

Each sum-node S_s has a countably infinite number of child product-nodes (note that a sum-node can have multiple child product nodes with the same partition). The probability distribution over these children is a Dirichlet Process over base distribution $\mathrm{G_P}(\text{scope(S)}, \alpha_s)$.

In this model, sum-nodes with identical scopes are distinct and independent. In short, the prior distribution over SPNs is specified as a tree of Dirichlet Processes over product-nodes; each product-node has a finite branching factor, and each sum-node has an infinite branching factor. The tree has maximal depth D, and there is a unique top sum-node $\mathrm{S}_{Top} = S_{\{1,\ldots,D\}}$.

The prior is parametrised by:

- For each s, a concentration parameter $\alpha_s > 0$. These may, for example, plausibly be a function of $|s|$.
- For each s, a probability distribution $\mathrm{G_P}(s)$ over partitions of s. These distributions may express significant prior knowledge concerning the desired shape of the tree, and also which partitions of the particular variables indexed by s are plausible. For example, different variables may have different meanings, and there may be prior beliefs that some partitions of s are more probable than others.
- For each $i \in \{1, \ldots, D\}$, a prior distribution on the univariate distribution $S_{\{i\}}(\cdot)$. For binary data, this is plausibly a vague beta prior.

Computation is greatly simplified if the predictive distribution for a new product node P_s, not yet associated with any data, is of a simple known form. For example, a newly generated product node P_s with no data might predict the uniform distribution over X_s.

2.1. Generating data with the tree model

Sampling and inference with the tree model are rather straightforward: we place a Blackwell-MacQueen urn process (Blackwell & MacQueen, 1973) at each sum node, and sample from each such process as required, starting from the top sum-node S_{Top}.

Each sample is generated recursively, as a tree of data requests: the root request, for all D variables is sent to S_{Top}. The recursion is as follows. For a node of type:

univariate-node : a sample from the univariate distribution defined by the node is returned.

sum-node : A child product-node P is sampled from the Blackwell-MacQueen urn process at the sum-node, and the sample request is then sent to that product node. The sample received from the product node is returned.

product-node : Let the node partition be $(s_1, \ldots, s_k)$. The sample request is partitioned into k requests with scope s_1 to s_k, and these requests are sent to the corresponding child nodes. When the samples from the k child nodes are returned, they are recombined to form a sample for the scope requested, and returned to the parent node.

This recursive sampling process generates an exchangeable sequence of samples. To carry out this sampling procedure, each sum-node (and each distribution-node) must maintain state information sufficient for its urn process: this information consists of the indexed sequence of samples taken from that node's generative process.

The univariate distribution nodes may simply be fixed probability distributions on one variable, or the may have an arbitrarily complex structure. To ensure exchangeability of the samples from the entire structure, it is sufficient that each univariate distribution node should provide an exchangeable sequence of samples, separately from all other nodes.

This generative model, with urn-processes, can be used in several ways:

- to generate an exchangeable sequence of samples

Algorithm 1 Update_Instance_Full_MH

```
Input: instances $\{x_1, ..., x_N\}$
Initialize $M$, $l$
for $lo = 1$ to $nLoop$ do
  for $n = 1$ to $N$ do
    $M' \leftarrow Update\_Instance\_One\_MH(M, root, x_n, n)$
    $temp\_l \leftarrow likelihood(M', x_n)$
    if $rand() < min(1, temp\_l/l(n))$ then
      $l(n) \leftarrow temp\_l$
      $M \leftarrow M'$
    end if
  end for
end for
```

Algorithm 2 Update_Instance_Full_Gibbs

```
Input: SPN $M$, instances $\{x_1, ..., x_N\}$
Initialize $M$
for $lo = 1$ to $nLoop$ do
  for $n = 1$ to $N$ do
    $M \leftarrow Update\_Instance\_One\_Gibbs(M, root, x_n, n)$
  end for
end for
```

- to fit a distribution to given data, using Gibbs sampling, Metropolis-Hastings, or many other MCMC methods. Gibbs and Metropolis-Hastings can be performed recursively throughout the tree, and in parallel on different branches.
- for any given state of the sampling system, an SPN is defined implicitly by the predictive sampling probabilities for the next data item at each node. This SPN can then be used for inference as described in (Poon & Domingos, 2011)

2.2. Learning algorithm for the tree structure

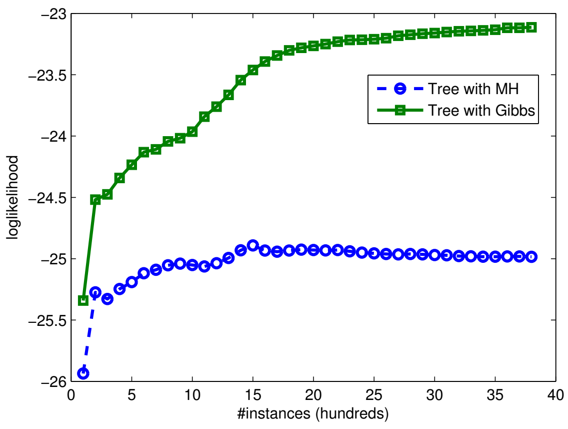

Figure 1. Loglikelihoods with different sampling schemes

Algorithm 3 Update_Instance_One_MH_Tree

```
**Input:** SPN $M$, sumnode index $i$, instances $\{x_1, ..., x_N\}$, instance index $n$
**Output:** SPN $M'$
$M' \leftarrow M$
$\{pidx_1, ..., pidx_K\} \leftarrow$ indexes of child nodes of sum node $M'.S_i$
**if** $M.S_i.allocate(n) \neq empty$ **then**
    $M'.P_{[M.S_i.allocate(n)]}.w - = 1$
**end if**
**for** $k = 1$ **to** $K$ **do**
    $p(k) = \frac{(M'.P_{[pidx_k]}.w)}{[(\sum_k^K M'.P_{[pidx_k]}.w)+\alpha]}$
**end for**
$p(K+1) = \frac{\alpha}{[(\sum_k^K M'.P_{[pidx_k]}.w)+\alpha]}$
select $k \sim p(k)$
**if** $k \leq K$ **then**
    $M'.S_i.allocate(n) = pidx_k$
    $P_{[M'.S_i.allocate(n)]}.w + = 1$
    $\{sidx_1, ..., sidx_L\} \leftarrow$ indexes of child nodes of product node $M'.P_{pidx_k}$
    **for** $l = 1$ **to** $L$ **do**
        $M' \leftarrow Update\_Instance\_One\_MH\_Tree(M, sidx_l, x_n, n)$
    **end for**
**else**
    $M' \leftarrow make\_prodnode\_in\_sumnode(M', i, x_n, n)$
**end if**
```

In this section, we explain two sampling methods for learning SPN topology. Algorithm 1 explains a Metropolis-Hastings (MH) sampling method, whereas Algorithm 2 explains a Gibbs sampling method. Algorithm 3 explains the MH learning rule used for updating the tree structure with one instance at a time. α is parameter of DP. Gibbs learning rule for updating one instance is a little difficult compared to MH method, because likelihood of choosing one child should be calculated in each step. It is more difficult to use the Gibbs learning rule for updating one instance at a time compared to the learning rule of the MH method. This is because the Gibbs sampling requires the calculation of the likelihood of choosing one child in each step. However, as calculating the likelihood in SPN is tractable, this step can be performed in a less complicated manner.

The MH and Gibbs sampling scheme was evaluated using a "hand-written optic digits" dataset which includes 8x8 pixels and digit classes. Only the image pixels with binarization used in the experiments. Partitions were randomly split every time a new product node was made. In figure 1, Algorithm 1 is used for the 'Tree with MH' condition, whereas Algorithm 2 is used for the 'Tree with Gibbs'. In this graph, the x-axis represents the number of instances used by the model. This simulation only used the instances once. We found the Gibbs sampling scheme performs better than the MH sampling scheme.

3. Model 2: A prior distribution for a class of dag-SPNs

Model 1 is a tree, and this may be undesirable in some applications: this is because sampling requests follow the tree, and nodes deep in the tree must therefore typically handle only a small fraction of all sample requests. Models of this type have been used (Gens & Domingos, 2013), but (Bengio, 2009) shows that deep dag-structured SPNs can express a more interesting class of distributions.

It is straightforward to alter the model to allow the sampling-request paths to form a dag. Each individual sampling request will take the form of a tree within the dag, but the totality of sampling requests will lie on the dag. In such a model, even nodes deep in the dag may handle a large fraction of sampling requests, so that 'deep learning' becomes possible.

We alter model 1 as follows. For each of the $2^D - D - 1$ possible scopes s of level at least 2, we set up a 'sum-node-group' $\mathbf{S}_s$, which consists of a hierarchical Dirichlet Process (Teh et al., 2004). A hierarchical DP consists of a set of 'layer 1' DPs which share a common 'layer 0' base distribution, which is itself a DP. For $\mathbf{S}_s$, the base distribution for the layer 0 DP is $\mathrm{G_P}(s)$, which is the same as the base-distribution of a sum-node with scope s in the tree-SPN model above.

For layer 1 of $\mathbf{S}_s$, we set up a separate layer 1 DP for each possible parent node from which sampling requests may come. More specifically, for any product-node with a child sum-node with scope s, then that child sum-node is placed as a DP in layer 1 of $\mathbf{S}_s$. In other words, all sum-nodes with scope s share the same base-distribution of product-nodes, which is itself a DP with base distribution $\mathrm{G_P}(s)$. The effect of this is that all sum-nodes with scope s in the dag will tend to share, and route requests to, common child product-nodes. Hence sampling requests from different sum-nodes can be routed to the same child product-node. Samples from a hierarchical DP are exchangeable, hence samples from the entire model are exchangeable, as before.

Remarkably, the only new parameters required are the concentration parameters of the layer 1 and layer 2 Dirichlet processes for each scope: once again, these may plausibly be functions only of the size of the scope.

Algorithm 4 Update_Instance_One_MH_DAG

```
**Input:** SPN $M$, sumnode index $i$, instances $\{x_1, ..., x_N\}$, instance index $n$
**Output:** SPN $M'$
$M' \leftarrow M$
$\{pidx_1, ..., pidx_K\} \leftarrow$ indexes of product nodes which have same *scope* to sum node $M'.S_i$
**if** $M.S_i.allocate(n) \neq empty$ **then**
    $M'.P_{[M.S_i.allocate(n)]}.w_i -= 1$
**end if**
**for** $k = 1$ **to** $K$ **do**
    $p(k) = \frac{(M'.P_{[pidx_k]}.w_i + \alpha \times M'.S_i.\beta_k)}{(\sum_k^K M'.P_{[pidx_k]}.w_i) + \alpha]}$
**end for**
$p(K+1) = \frac{(\alpha \times M'.S_i.\beta_{K+1})}{[(\sum_k^K M'.P_{[pidx_k]}.w_i) + \alpha]}$
select $k \sim p(k)$
**if** $k \leq K$ **then**
    $M'.S_i.allocate(n) = idx_k$
    $P_{[M'.S_i.allocate(n)]}.w_i += 1$
    $\{sidx_1, ...sidx_L\} \leftarrow$ indexes of child nodes of product node $M'.P_{pidx_k}$
    **for** $l = 1$ **to** $L$ **do**
        $M' \leftarrow Update\_Instance\_One\_MH\_DAG(M, sidx_l, x_n, n)$
    **end for**
**else**
    $M' \leftarrow make\_prodnode\_in\_sumnode(M', i, x_n, n)$
    $M'.S_i.\beta \leftarrow sample\_beta(M'.S_i.allocate, \alpha, \gamma)$
**end if**
```

Algorithm 1 and 2 can also be used for learning both the tree and dag structure. Algorithm 4 explains the MH learning rule at updating dag structure with one instance. α and γ are parameters of the hierarchical DP. Algorithm 4 uses the inference scheme explained in Ch 5.3 of (Teh et al., 2004).

Experiments are in progress. There are a number of MCMC sampling techniques available for individual HDPs (Teh et al., 2004; Neal, 2000), and many possible schemes for managing the MCMC sampling for both models recursively.

4. Discussion

There are relatively few effective algorithms for learning the structures of graphical models: MCMC sampling using the dag-SPN (our model 2) appears to be a particularly elegant possibility, which is as far as we know unexplored. The construction appears both modular and generic, and could be applied to the generation of many types of structured object, provided the generative decisions take the form of a tree, and node-scopes respect the same partial ordering for all objects.

Learning in product nodes is, however, problematic because there are many possible partitions and it is hard to find a good partition by Gibbs sampling, proposing random partitions. This is slow because the merit of a good partition only becomes apparent when many data-requests have been transferred to the new product node: when a product node is first generated, even if its partition is optimal, it has no data assigned to it, and so at first it predicts a uniform distribution over its scope variables. This means that every new product node is at an initial disadvantage compared to existing competitor product nodes which have data currently assigned to them. An effective sampling method, therefore, should make proposals of altering the product partitions directly: however, such proposals are expensive, since if high-level partition elements are altered in this way, then lower level partition elements need to be changed as well. We are investigating such algorithms.

In the above experiment, random partition priors were explored, but other partition priors could also be used. It would be possible to have a hybrid method, in which the 'prior' for partition of variables would be made sensitive to the number of allocated instances; when few instances are allocated to some product node, the partition of this node would be fully separated into univariate nodes. To solve this problem, we are examining other SPN structure leraning methods to find good initialisations. One possibility is the LearnSPN algorithm in (Gens & Domingos, 2013). When a tree structure is made with the DP, the whole algorithm becomes a non-parametric Bayesian biclustering. If a dag-SPN is made with the HDP, the algorithm becomes more interesting and is an idea we will explore in the future. Additionally, (Lowd & Domingos, 2013; Peharz et al., 2013) study the bottom-up learning of SPN and the arithmetic circuit. These ideas can be used for making good candidates of product nodes. It is also interesting to note that (Rooshenas & Lowd, 2014) uses the hybrid approach of top-down and bottom-up.

In summary, we have defined prior distributions on SPNs that specify only mixture-distribution concentration parameters, priors on partitions of scopes, and vague priors on univariate distributions. However, finding effective sampling methods is a challenging problem.

Acknowledgments

This research was conducted while C Watkins was visiting the Biointelligence Lab., School of Computer Science and Engineering at Seoul National University, with support from the MSIP (Ministry of Science, ICT & future Planning), Korea. SW Lee and BT Zhang were were supported by the National Research Foundation (NRF) Grant funded by the Korea government (MSIP) (NRF-2013M3B5A2035921)

References

Poon, H. and Domingos, P. Sum-product networks: A new deep architecture. *Uncertainty in Artificial Intelligence 27 (UAI11)*, 2011.

Poon, H. and Domingos, P. Discriminative learning of sum-product networks. *Advances in Neural Information Processing Systems 25 (NIPS12)*, 2012.

Amer, M. R. and Todorovic, S. Sum-product networks for modeling activities with stochastic structure. *Computer Vision and Pattern Recognition (CVPR12)*, 2012.

Gens, R. and Domingos, P. Learning the structure of sum-product networks. *International Conference on Machine Learning 30 (ICML13)*, 2013.

Teh, Y. W., Jordan, M. I., Beal, M. J, and Blei, D. M. Hierarchical Dirichlet Process. *Journal of the American Statistical Association*, 101(476): 1566–1581, 2004.

Blackwell, D. and MacQueen, J. B. Ferguson Distributions Via Polya Urn Schemes. *Annals of Statistics*, 1(2):353–355, 1973.

Bengio, Y. Learning deep architectures for AI. *Foundation and Trends in Machine Learning*, 2(1):1–127, 2009.

Neal, R. M. Markov Chain Sampling Methods for Dirichlet Process Mixture Models . *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000.

Rooshenas, A. and Lowd, D. Learning Sum-Product Networks with Direct and Indirect Variable Interactions. *International Conference on Machine Learning 31 (ICML14)*, 2014.

Lowd, D. and Domingos, P. Learning Markov networks with arithmetic circuits. *International Conference on Artificial Intelligence and Statistics 16 (AISTATS13)*, 2013.

Peharz, R., Geiger, B. C., and Pernkopf, F. Greedy Part-Wise Learning of Sum-Product Networks. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD13)*, 2013.