SP^3 – Sum Product Probabilistic Programming

Extended Abstract

Karl Stelzner
TU Darmstadt, Germany
stelzner@cs.tu-darmstadt.de

Alejandro Molina
TU Darmstadt, Germany
molina@cs.tu-darmstadt.de

Robert Peharz
Univ. of Cambridge, UK
rp587@cam.ac.uk

Antonio Vergari
MPI for Intelligent Systems, Germany
antonio.vergari@tue.mpg.de

Martin Trapp
OFAI, Austria
martin.trapp@ofai.at

Isabel Valera
MPI for Intelligent Systems, Germany
isabel.valera@tue.mpg.de

Zoubin Ghahramani
Univ. of Cambridge, UK
Uber AI Lab, USA
zoubin@eng.cam.ac.uk

Kristian Kersting
TU Darmstadt, Germany
kersting@cs.tu-darmstadt.de

1 Deep Probabilistic Programming

Probabilistic models provide a framework for describing abstract prior knowledge and using it to reason under uncertainty. Deep probabilistic programming languages (PPL) are a powerful tool to ease the development of probabilistic models [2, 10]. They let users specify generative probabilistic models as programs and then "compile" those models down into inference procedures. Since probabilistic inference is still intractable, existing deep PPLs leverage deep learning for inference. The key idea is to describe inference via a second model called an inference model trained in variational fashion. Both the generative and the inference models can include deep neural networks as components. While the details of the employed neural architectures may differ, they typically model distributions only *implicitly*, i.e., they allow for sampling but not the computation of marginal probabilities.

This extended abstract makes the case to not "go down the full neural road," but to explicitly obtain uncertainties in an arithmetic circuit manner [3]. To this end, sum-product networks (SPNs) are a promising candidate, as they are a class of probabilistic model which permit explicit uncertainties and efficient inference [9]. More precisely, SPNs can compute any marginalization and conditioning query in time linear of the model's representation size. Although SPNs can be described in a nutshell as "deep mixture models", they have received no attention in the deep PPL community, despite their attractive inference properties.

2 Sum-Product Probabilistic Programming

Very much like deep neural networks, sum-product networks (SPNs) are able to approximate any prediction function via probabilistic inference in an asymptotic sense, see e.g. [8] for a sketch of the argument. Formally, a sum-product network is a rooted directed acyclic graph, comprising *sum*, *product* or *leaf* nodes. The scope of an SPN is the set of random variables appearing on the network. An SPN can be defined recursively as follows: (1) a tractable univariate distribution is an SPN; (2) a product of SPNs defined over different scopes is an SPN; and (3), a convex combination of SPNs over the same scope is an SPN. Thus, a product node in an SPN represents a factorization over independent distributions defined over different random variables, while a sum node stands for a mixture of distributions defined over the same variables. From this definition, it follows that the joint distribution modeled by such an SPN is a valid probability distribution, i.e., each complete and partial evidence inference query produces a consistent probability value [7, 9].

More importantly, Peharz *et al.* [8] have recently introduced a particularly simple way to construct SPNs, waiving the necessity for structure learning and simplifying their use for deep learning: *SPNs are obtained by first creating a random region graph laying out the overall network structure, and subsequently populating the region graph with tensors of SPN nodes.* The resulting *Random Tensorized SPNs*, we refer to [8] for more details, can naturally be implemented in deep learning frameworks such as TensorFlow [1] and (Probabilistic) PyTorch [6]. The nodes of a region are represented by a matrix with rows corresponding to samples in a mini-batch and columns corresponding to the number of distributions in the region. All computation are performed in the log-domain, using the well known log-sum-exp trick, readily provided in deep learning frameworks. Sum weights, which we require to be non-negative and normalized, are re-parameterized via log-softmax layers. Product tensors are implemented by taking outer products (actually sums in the log-domain) of the two matrices below, realized by broadcasting. This way, SPNs are easily optimized end-to-end using automatic differentiation, SGD, and automatic GPU parallelization. To avoid overfitting, one can adopt the well known dropout heuristic, which yields an elegant probabilistic interpretation in our

PROBPROG'18, October 04–06, 2018, Boston, MA, USA

.

models as marginalization of missing features (dropout at inputs) and as injection of discrete noise (dropout at sum nodes).

Moreover, with (stochastic) computation graphs, the support of PPLs such as Pyro (pyro.ai) by SPNs on the backend is in reach. As a first step towards such an integration, we have implemented SPFlow, a Python library for SPNs available at https://github.com/alejandromolinaml/SPFlow. In addition to featuring several SPN inference and learning approaches, including Random Tensorized SPNs, and compilation to deep learning frameworks, it implements a domain specific programming language for specifying SPNs:

```
spn = 0.4 * (Categorical(p=[0.9, 0.1], scope=0) *
             (0.3 * (Gaussian(mean=10, stdev=3, scope=1) *
                     Gamma(alpha=2, beta=2, scope=2)) +
              0.7 * (Gaussian(mean=0, stdev=0.1, scope=1) *
                     Gamma(alpha=1, beta=5, scope=2)))) +
      0.6 * (Categorical(p=[0.2, 0.8], scope=0) *
             Gaussian(mean=-2, stdev=1, scope=1) *
             Gaussian(mean=1, stdev=1, scope=2))
```

3 Illustration: Attend-Infer-Repeat

To investigate the potential of the resulting Sum Product Probabilistic Programming (SP^3), we considered probabilistic programming for scene understanding. Structured problem representations are a key ingredient to higher level reasoning tasks such as planning and executing interactions with physical objects. Generative modeling aims to infer such representations $\mathbf{z}$ from unstructured sensory input $\mathbf{x}$ by postulating a process $p_\theta(\mathbf{x} \mid \mathbf{z})$ by which the observed data is generated. A prominent example are Attend-Infer-Repeat (AIR) models [4], see also pyro.ai/examples/air.html for an implementation in Pyro. AIR decomposes the process of generating a scene $\mathbf{x}$ into discrete steps, at each of which a small part of the scene, an object, is generated. For instance, on the multi-MNIST dataset, the objects are single digits, as illustrated in Fig. 1. To this end, the distribution over objects is represented using a variational autoencoder (VAE) model [5]. Additionally, the AIR model samples the location and size of each object. To produce the final image, they are then placed into the larger scene using this pose information $\mathbf{z}$. Inference is performed using a recurrent neural network which generates estimates of the number, location, and scale of objects given a scene. The entire system can then be optimized end-to-end by sampling estimates of the parameters $\mathbf{z}$ from the inference network, and maximizing the evidence lower bound.

We follow the basic AIR idea, but opt to use a Random Tensorized SPN

```
SPNobjects = CreateRandomSPN(obj_width * obj_height)
```

instead of a VAE to represent the distribution of objects and to use a second SPN, a shallow product of Gaussians with zero means, to capture the distribution of the background:

```
SPNbackground = Normal(mean=0, stdev=0.1, scope=0) * ...
    * Normal(mean=0, stdev=0.1, scope=max_pixel)
```

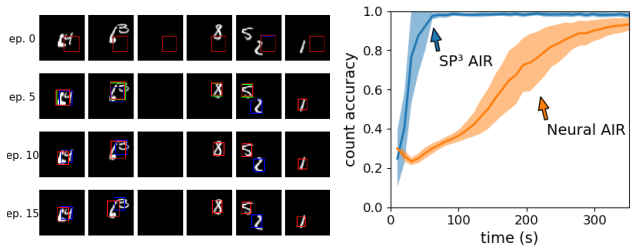

Figure 1. SP^3 AIR models. (Left) SP^3 model's attention windows at different stages of training. The 1st, 2nd and 3rd time-steps are displayed using red, blue and green borders respectively. (Right) Count accuracy over time.

This allows us to *explicitly* compute the likelihood that a given patch of the scene is either an object or background. It also generalizes the original AIR model, in which the background is assumed to always be zero (i.e., black). Without going into details, using Random Tensorized SPNs allows us to interpret the inference network as a deterministic function $f_\phi : \mathbf{x} \mapsto \mathbf{z}$. The AIR training objective can then be rewritten to directly maximize the conditional likelihood $p_\theta\left(\mathbf{x} \mid f_\phi(\mathbf{x})\right)$ of an image $\mathbf{x}$ under the model given the prediction of the inference network: instead of sampling latent variables to obtain a potentially high-variance estimate of the objective's gradient, one can exactly compute it using SPNs.

Our preliminary experimental results indicate the benefits of this explicit AIR approach. We trained the SP^3 AIR as well as the original Neural AIR model with 3 inference steps on 60,000 images from the 50×50 dataset of multi-MNIST digits. Each image contains zero, one or two non-overlapping random MNIST digits with equal probability. The SP^3 model's attention windows at different stages of training are shown in Fig. 1(left). As one can see, it manages to locate the digits without requiring supervision. Fig. 1(right) shows the count accuracy over time. As one can see, the SP^3 model (blue) detects the counts of digits accurately and >5× faster then the neural AIR model (orange).

4 Conclusions

The potential of deep PPLs is to combine the advantages of probabilistic models, deep learning, and programming languages. Existing deep PPLs, however, employ deep neural networks for generative and inference models and, consequently, allow for sampling but not the computation of marginal probabilities. To overcome this, we outlined Sum Product Probabilistic Programming (SP^3), a deep PP framework with an explicit representation of uncertainties. SP^3 expands the scope of probabilistic programming to be as flexible as deep neural learning but computationally more efficient using sum-product networks. Our preliminary experimental results on unsupervised scene understanding suggest that this dream is not unattainable. A tight integration with a PPL such as Pyro, however, is left to future work.

References

[1] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Guillaume Baudart, Martin Hirzel, and Louis Mandel. 2018. Deep Probabilistic Programming Languages: A Qualitative Study. *CoRR* abs/1804.06458 (2018). arXiv:1804.06458 http://arxiv.org/abs/1804.06458

[3] Arthur Choi and Adnan Darwiche. 2017. On Relaxing Determinism in Arithmetic Circuits. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 825–833.

[4] S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E. Hinton. 2016. Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. In *Annual Conference on Neural Information Processing Systems (NIPS)*. 3225–3233.

[5] Diederik P Kingma and Max Welling. 2013. Auto-Encoding Variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*.

[6] Siddharth Narayanaswamy, Brooks Paige, Jan-Willem van de Meent, Alban Desmaison, Noah D. Goodman, Pushmeet Kohli, Frank D. Wood, and Philip H. S. Torr. 2017. Learning Disentangled Representations with Semi-Supervised Deep Generative Models. In *Annual Conference on Neural Information Processing Systems (NIPS)*. 5927–5937.

[7] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. 2015. On Theoretical Properties of Sum-Product Networks. *The Journal of Machine Learning Research (JMLR)* (2015).

[8] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. 2018. Probabilistic Deep Learning using Random Sum-Product Networks. In *Working Notes of the UAI 2018 Workshop on Uncertainty in Deep Learning (UDL); also arXiv preprint arXiv:1806.01910*.

[9] Hoifung Poon and Pedro Domingos. 2011. Sum-Product Networks: a New Deep Architecture. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

[10] Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. 2017. Deep Probabilistic Programming. *CoRR* abs/1701.03757 (2017). arXiv:1701.03757 http://arxiv.org/abs/1701.03757