Sum-Product Logic: Integrating Probabilistic Circuits into DeepProbLog

Arseny Skryagin [1] **Karl Stelzner** [1] **Alejandro Molina** [1] **Fabrizio Ventola** [1] **Zhongjie Yu** [1] **Kristian Kersting** [1]

Abstract

We introduce Sum-Product Logic (SPLog), a deep probabilistic logic programming language that incorporates learning through predicates encoded as probabilistic circuits, specifically sum-product networks. We show how existing inference and learning techniques can be adapted for the new language. Our empirical illustrations demonstrate the benefits of supporting symbolic and deep representations, both neural and probabilistic circuit ones for inference and (deep) learning from examples. To the best of our knowledge, this work is the first to propose a framework where deep neural networks, probabilistic circuits, expressive probabilistic-logical modeling and reasoning are integrated.

1. Introduction

In the past three years, many deep probabilistic programming languages (DPPL) have been proposed, e.g., Edward (Tran et al., 2017), Pyro (Bingham et al., 2018) and Turing (Ge et al., 2018), among others. The main focus of these works was to leverage the expressive power of deep neural networks within probabilistic programming systems. In particular, DeepProblog (Manhaeve et al., 2018) targets similar goals in the relational setting, by allowing probabilistic predicates to be specified as (conditional) distributions defined via deep neural networks. It further supports end-to-end training, program induction, and (sub-)symbolic representation as well as reasoning.

However, in this setting, deep models are used purely as (unfaithful) conditional density estimators. This limits the types of inferences that are possible in DeepProbLog: Lacking any model for the inputs of the neural network, missing values can not be inferred or sampled. A possible remedy to these issues may be to use deep generative models such as variational autoencoders (Kingma & Welling, 2014) or normalizing flows (Rezende & Mohamed, 2015). While these models specify joint probability distributions, inference within them is highly intractable or limited to a restricted set of queries. If the goal of the overall system is to answer a variety of probabilistic queries, the cost of computing them within these models can become prohibitive.

Sum-Product Networks (SPNs) (Poon & Domingos, 2011) is a probabilistic graphical model that allows us to do inference, marginalize, and sample in linear time in the size of the model. The expressiveness and and speed make SPNs an ideal candidate for our purposes.

Consequently, we propose a novel deep probabilistic programming language, called Sum-Product Logic (SPLog). Its main components are sum-product predicates, which encode the predicates as SPNs so that all conditional and marginal queries on these predicates may be answered exactly in linear time.

2. Sum-Product Logic (Programming)

ProbLog. The ProbLog language (Kimmig et al., 2007) allows its users to write probabilistic logic programs, in which some logical facts are annotated with probabilities. It uses sentential decision diagrams (SDD) (Darwiche, 2011) to represent propositional knowledge bases. In particular one can use the annotated disjunctions (AD) to assign logical rules a probability: $p :: a(\vec{x}) :- b_1, \ldots, b_m$. This example statement indicates that if the atoms $b_1, \ldots, b_m$ are true, then the predicate $a(x)$ is implied with probability p. This function opened the doors for further possibilities, beyond just specifying constant probabilities. One approach is to delegate the choice of probability to an external model which assigns them based on the predicate's arguments $\vec{x}$. This is the approach taken in Hybrid ProbLog (Gutmann et al., 2010): here, primitive continuous distributions such as Gaussians are used to specify distributions $p(\vec{X})$, enabling the definition of flexible predicates such as: $p(\vec{X} = \vec{x}) :: a(\vec{x}) :- b_1, \ldots, b_m$. To employ more flexible distributions, DeepProbLog (Manhaeve et al., 2018) uses deep neural networks to provide conditional probabilities for predicates. In this case, predicates of the form: $p(\vec{Y} = \vec{y} | \vec{X} = \vec{x}) :: a(\vec{x}, \vec{y}) :- b_1, \ldots, b_m$. are specified, where $p(\vec{Y} = \vec{y} | \vec{X} = \vec{x})$ is the probability the neural network assigns

[1]CS Department, TU Darmstadt, Darmstadt, Germany. Correspondence to: Arseny Skryagin <arseny.skryagin@cs.tu-darmstadt.de>.

Working Notes of the ICML 2020 Workshop on "Bridge Between Perception and Reasoning: Graph Neural Networks and Beyond".

the output $\vec{y}$ when given the input $\vec{x}$. By implementing automatic differentiation in ProbLog, the network can be trained jointly with the ProbLog model.

However, not all deep neural networks are safe to encode calibrated distributions with complex dependency structures nor do they encode joint distributions over inputs and outputs. To elevate this problem, we now show how, analogously, ProbLog programs may refer to SPNs as external probabilistic models. This will give rise to the SPLog[1] framework.

SPNs. Sum-Product Networks (SPNs)(Poon & Domingos, 2011) are deep mixture models represented via a rooted directed acyclic graph with a recursively defined structure. There are three types of nodes (same for the root node). *Sum-nodes* that represent a mixture over distributions, and they have weighted edges pointing to their children as the mixing weights. *Product-nodes* with edges pointing to their children, representing decompositions via context-specific independencies. Finally, we have the *Leaf-nodes* representing parametric distributions (e.g Gaussian, Poisson, etc.). To compute probabilities with an SPN, we simply have to compute the values of the nodes starting from the leaves. Since each leaf is a univariate distribution, one simply sets the evidence on those distributions, obtains the probabilities and then evaluate the network bottom up. On product nodes, we simply multiply the values of the children nodes. On sum nodes, we sum the weighted values of the children nodes. The value at the root indicates the probability of the given configuration. To compute marginals, i.e., the probability of partial configurations, it is necessary to sum out one or more variables. This can be achieved by setting the probability at the leaves for those variables to 1 and then proceed as before. Conditional probabilities can then be computed as the ratio of partial configurations. To compute MPE states, we replace sum nodes by max nodes and then evaluate the graph first with a bottom up pass, but instead of weighted sums we pass along the weighted maximum value. Finally, in a top down pass, we select the paths that lead to the maximum value, finding the MPE states of fired leaves. Differently to other probabilistic graphical models, sum-product networks (SPNs) support exact tractable inference i.e. inference complexity is linear in the size of the graph.

SPLog. An SPLog program is a ProbLog program that is extended with a set of ground sum-product annotated disjunctions (spADs) of the form:

$$spn(m_a, \vec{Q}, \vec{E} = \vec{e}) :: a(\vec{e}, \vec{q_1}); \ldots; a(\vec{e}, \vec{q_n}) :- b_1, \ldots, b_m,$$

where the b_i are atoms, $\vec{E} = \vec{e}$ is a vector of random variables representing *evidence* as the input of the SPN for predicate a, $\vec{Q}$ is the set of random variables being *queried*, and $q_1, \ldots, q_n$ are vectors of its realisations. m_a is the identifier of a SPN model which specifies a probability distribution over the set of variables $\vec{X}$, where $\vec{Q} \subseteq \vec{X}, \vec{E} \subseteq \vec{X}, \vec{E} \cap \vec{Q} = \emptyset$. We use the notation $spn(m_a, \vec{Q}, \vec{E} = \vec{e})$ to indicate that this set of logical rules is true with the probabilities $p_{m_a}(\vec{Q} = \vec{q_i} | \vec{E} = \vec{e})$, which the SPN assigns to the query realisations given the evidence. It is clear that SPLog directly inherits its semantics, and to large extent also its inference mechanism, from (Deep)ProbLog.

Algorithm 1 SPLog - training procedure

ground = ProbLog.ground(q)
sdd = SDD.create_from(ground)
$\vec{x}$ = ProbLog.extract_parameters(model)
p, $\frac{\partial p}{\partial x}$ = ProbLog.solve(sdd, $\vec{x}$)
optimizer.backward($-\frac{1}{p+\varepsilon} \cdot \frac{\partial p}{\partial x}$)

The approach to jointly train the parameters of probabilistic facts and sum-product networks in the SPLog program is the following. Similar to (Manhaeve et al., 2018), we use the *learning from entailment* (i.e. learning from queries) setting. That is, for a given SPLog program with parameters $\mathscr{X}$ and a set $\mathscr{Q}$ of pairs (q, p) where q is a query and p its desired success probability, we compute for a loss function L:

$$\arg\min_{\vec{x}} \frac{1}{|\mathscr{Q}|} \sum_{(q,p) \in \mathscr{Q}} L(P_{\mathscr{X}=\vec{x}}(q), p).$$

Alg. 1 describes the training loop given an SPLog program, its SPN models and a query. Specifically, to compute the gradient of the SPN parameters with respect to the loss function (step 4), we can apply automatic differentiation both to the ProbLog program and to the SPNs. For the former, the algebraic extension of ProbLog (aProbLog) (Kimmig et al., 2011) is used. This associates to each probabilistic fact a value from an arbitrary commutative semiring. aProbLog uses a labeling function that explicitly associates values from the chosen semiring with both facts and their negations combining these using semiring addition $\oplus$ and multiplication $\otimes$ on the SDD. For SPLog we are using gradient semiring, whose elements are tuples of the form $(p, \frac{\partial p}{\partial x})$, where p is a probability and $\frac{\partial p}{\partial x}$ is the partial derivative of the probability p with respect to a parameter x. We write $t(p_i) :: f_i$ for the learnable probability of a probabilistic fact. All this forms the basis for forward mode automatic differentiation within ProbLog, and it results in the partial derivatives of the loss with respect to the SPN probabilities $\partial L / \partial p_{m_a}(\vec{Q} = \vec{q_i} | \vec{E} = \vec{e})$. For all technical details we refer to (Manhaeve et al., 2018). In order to compute the gradients within the SPN, we can use backward mode automatic differentiation as implemented in standard deep learning frameworks. This provides the partial deriva-

[1] Code is available at https://github.com/askrix/Sum-Product-Logic

tives of SPN outputs with respect to its parameters θ, i.e., $\partial p_{m_a}(\vec{Q}=\vec{q}_i|\vec{E}=\vec{e})/\partial\theta$. By multiplying the two sets of gradients, we obtain the partial derivatives of the loss with respect to the SPN parameters $\partial L/\partial\theta$, which we use for gradient descent.

3. Empirical Illustrations

Our intention here is to investigate the benefits of SPLog. To this aim, we implemented everything using PyTorch based on the SPFlow library (Molina et al., 2019) and the DeepProbLog code (Manhaeve et al., 2018) and ran two experiments on the MNIST data set.

Tractable inference of probabilistic circuits within ProbLog. To illustrate the benefit of combining ProbLog and SPNs, we revisited the MNIST Addition experiment from (Manhaeve et al., 2018) using SPNs instead of deep neural networks. The idea behind the experiments is to perform the addition of two digits represented by two randomly chosen images from the MNIST data set. One query of the program contains the indices of the two MNIST images as the inputs and the resulting sum as the label. The corresponding SPLog program is:

```
spn(mnist,[X],Y,[0,1,2,...,9]) :: dig(X,Y).
add(X,Y,Z):-dig(X,X2),dig(Y,Y2),Z is X2+Y2.
```

That is, we specify the spAD as: $spn(m_{dig},\vec{X},\vec{Y}) :: dig(\vec{x},0);\ldots;dig(\vec{x},9)$, where $\vec{X}=x_1,\ldots,x_k$ is a vector of ground terms representing the inputs of the SPN for predicate dig while $0,\ldots,9$ are possible classes of the MNIST data set, i.e., we specify the spAD for an image of the digit 5 as follows:

$$spn(m_{dig},\boxed{5},[0,\ldots,9]) :: dig(\boxed{5},0),\ldots,dig(\boxed{5},9).$$

Because SPNs can handle incomplete data, we utilized this advantage by making the task more difficult. Namely, it was performed with only partially available data, i.e., the right half of the pixels were removed from each image. The results proved that the model converges well as can bee seen in Figure 1.

End-to-end learning across deep neural networks, probabilistic circuits and ProbLog. Even though SPNs resolve the issues of DeepProbLog we are addressing, the idea of SPLog is not to replace DNNs. On the contrary, DNNs play an important role in our framework. Using a variational auto-encoder based on a feed-forward DNN we have run the MNIST addition experiment in the following fashion. The original images were replaced by the codes produced by the encoder network of the VAE which was trained jointly within our framework. The results are summarized in Fig.1.

Probabilistic Auto-Encoder. Furthermore, instead of modeling only the code $P(C_i|code(X))$ for the ProbLog, the distribution of the original image and its code can be jointly modeled with SPNs, i.e. $P(C_i|X,code(X))$. This results in a novel auto-encoder equipped with tractable probabilistic models, namely Probabilistic Auto-Encoder (PAE). Next to the AE, the PAE comprises two SPNs, one modeling the joint distribution of the original image and its code, and the other modeling the code and its reconstructions. Adding the lower bound of the KL divergence (KLD) between these two SPNs into the loss, which is calculated employing the Gauss-Hermite quadrature based on the mean-field assumption (Hershey & Olsen, 2007), we keep the distributions of the two SPNs close. Compared to a VAE, the PAE provides several relaxations. Firstly, the isotropic multivariate Gaussian assumption made on the code distribution can be extended to a mixture of Gaussians. Secondly, conditional samples of the code given an original image can be acquired by one bottom-up pass and another top-down pass on the SPN, without the re-parameterization trick. Last but not least, both encoder and decoder have a probabilistic density measure, pointing out how likely the code or a reconstruction meet the training data domain. The main advantage of the PAE lies in the utilization of the SPNs. Even if the given data set contains missing and/or noisy data, one can train the PAE by sampling the code from the first SPN. The initial results are summarized in Fig. 2. As in our first experiment, we removed the right half of the image pixels. Only in the rows 4, 7 and 8 with MPE completed images (second column) admit the shape of 0. Whereas the reconstructed images from the MPE sampled code (third column) yield better results.

4. Conclusion and Future Work

Triggered by the re-emerging research area of Hybrid AI—the computational and mathematical modeling of complex AI systems— in this work we sketched SPLog, a novel deep probabilistic PL (DPPL), which combines deep probabilistic models—SPNs—with neural probabilistic logic programming —DeepProbLog. Specifically, SPLog incorporates the advantages of SPNs into ProbLog and so facilitates PAE. It has shown good performance in solving challenging tasks on MNIST data set. This paves the way towards a unifying deep programming language for System 2 approaches (Bengio, 2019).

The performed empirical illustrations open the door to further advancements, e.g. "reverse" Neural-Symbolic Visual Question Answering (NS-VQA) (Yi et al., 2018). Since one can handle the non-probabilistic, i.e. true/false facts within ProbLog, one can consider the following example. After the MNIST addition model is trained as it is stated in 3, we can ask for the visual solution of the given query. We provide an SPLog program that takes as an input query the result of the addition of two single digits. Utilizing (conditional) sam-

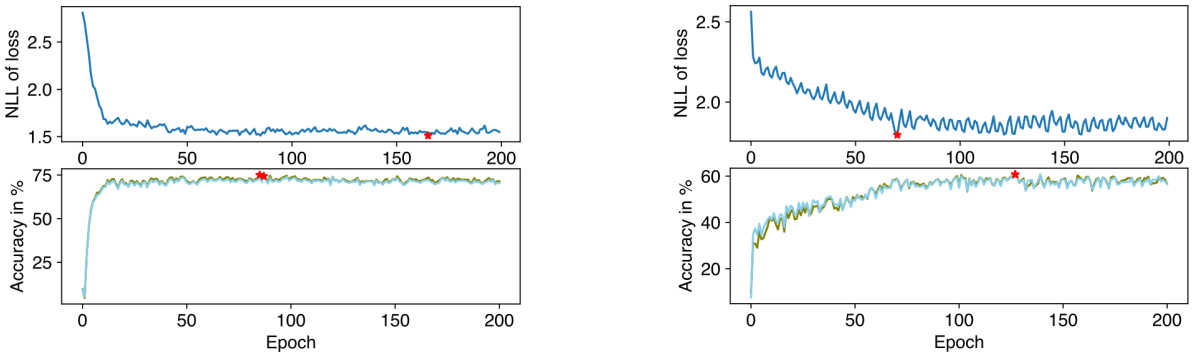

Figure 1. Loss and accuracy of the addition experiment for **(left)** codes of full images and **(right)** partial images. The highest accuracy values for the joint model **(left)** are 74.31% for train (sky blue) and 75.03% for test (olive). For the marginal model **(right)**, we get 64.76% for train and 66.03% for test. Note that the accuracy in this task involves the prediction of two images, indicating a classification accuracy of more than 80% per image (best in color).

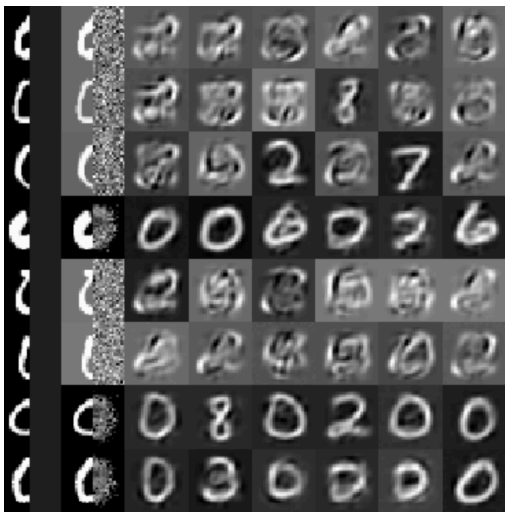

Figure 2. Initial results of PAE. First column represents "marginalized" images, second - image completion with MPE, third - reconstructions by decoder from MPE sampled code, the rest - reconstructions by decoder from conditional sampled code.

pling, the framework retrieves the images for all possible solutions in accordance with the commutative property of $+$. The output of such a program, given the query $\mathit{result}(10)$, will then be

$$\mathit{result}(10) = \boxed{1}+\boxed{9} = \boxed{2}+\boxed{8} = \ldots = \boxed{5}+\boxed{5}$$

The Einsum Networks (EiNets) (Peharz et al., 2020) feature the advancement in scalability of *probabilistic circuits*. This term was introduced by (Van den Broeck et al., 2019) as the umbrella term for tractable probabilistic models. The EiNets are an important step stone towards (sub-)symbolic reasoning on "big" data sets like Celeba (Liu et al., 2015) and SVHN (Netzer et al., 2011) (format 1). We envision their integration in our framework as interesting direction for scaling up SPLog.

Acknowledgements

The authors would like to thank Steven Lang for very valuable discussions. AS and KK acknowledge support by the German Federal Ministry of Education and Research and the Hessian Ministry of Science and the Arts within the National Research Center for Applied Cybersecurity ATHENE. FV and KK acknowledge the support by the German Research Foundation (DFG) as part of the Research Training Group Adaptive Preparation of Information from Heterogeneous Sources (AIPHES) under grant No. GRK 1994/1. KK also acknowledges the support of the Rhine-Main Universities Network for "Deep Continuous-Discrete Machine Learning" (DeCoDeML) as well as the discussion within the AI lighthouse BMWi project SPAICER, 01MK20015E. ZY and KK acknowledge the funding of the German Federal Ministry of Education and Research (BMBF) project "MADESI", 01IS18043B.

References

Bengio, Y. From System 1 Deep Learning to System 2 Deep Learning. Invited talk NeurIPS, 2019.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. Pyro: Deep Universal Probabilistic Programming. In *Journal of Machine Learning Research*, 2018.

Darwiche, A. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 819–826, 2011.

Ge, H., Xu, K., and Ghahramani, Z. Turing: a language

for flexible probabilistic inference. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, pp. 1682–1690, 2018. URL http://proceedings.mlr.press/v84/ge18b.html.

Gutmann, B., Jaeger, M., and De Raedt, L. Extending ProbLog with Continuous Distributions. *Inductive Logic Programming*, 2010.

Hershey, J. R. and Olsen, P. A. Approximating the kullback leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 4, pp. IV–317–IV–320, 2007.

Kimmig, A., De Raed, L., and Toivonen, H. ProbLog: A probabilistic Prolog and its application in link discovery. In *IJCAI*, pp. 2462–2467, 2007.

Kimmig, A., Van den Broeck, G., and De Raed, L. An Algebraic Prolog for Reasoning about Possible Worlds. In *AAAI*, 2011.

Kingma, D. P. and Welling, M. Auto-Encoding Variational Bayes. In *ICLR*, 2014.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., and De Raedt, L. DeepProbLog: Neural Probabilistic Logic Programming. In *NeurIPS*, pp. 3753–3763, 2018.

Molina, A., Vergari, A., Stelzner, K., Peharz, R., Subramani, P., Di Mauro, N., Poupart, P., and Kersting, K. SPFlow: An Easy and Extensible Library for Deep Probabilistic Learning using Sum-Product Networks. arXiv:1901.03704, 2019.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Reading digits in natural images with unsupervised feature learning. *NIPS*, 01 2011.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K., and Ghahramani, Z. Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits. In *ICML*, 2020.

Poon, H. and Domingos, P. Sum-Product Networks: A New Deep Architecture. In *UAI*, pp. 337–346, 2011.

Rezende, D. J. and Mohamed, S. Variational Inference with Normalizing Flows. *Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.*, 2015.

Tran, D., Hoffman, M. D., Saurous, R. A., Brevdo, E., Murphy, K., and Blei, D. M. Deep Probabilistic Programming. In *ICLR*, 2017.

Van den Broeck, G., Di Mauro, N., and Vergari, A. Tractable Probabilistic Models: Representations, algorithms, learning, and applications. In *Tutorial at UAI*, Juli 2019.

Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., and Tenenbaum, J. B. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems*, pp. 1039–1050, 2018.