DeepProbLog Tasks

Davide Lusuardi, 223821, *davide.lusuardi@studenti.unitn.it*

***Abstract*—DeepProbLog is a framework where general-purpose neural networks and expressive probabilistic-logical modeling and reasoning are integrated in a way that exploits the full expressiveness and strengths of both worlds and can be trained end-to-end based on examples.**

In this report, we show how to solve AI tasks that require the integration of high-level reasoning and low-level perception. We focus on the multi-digit MNIST octal-division task, a problem that involves computing the division between two lists of MNIST digits representing multi-digit octal numbers. We formulate the problem as a DeepProbLog program and demonstrate its effectiveness through experimental results. Using DeepProbLog we are able to solve the task given that supervision is only present at the output side of the probabilistic reasoner and considering that the approach can be extended to multi-digit numbers without being explicitly trained on them. Our experiment shows that DeepProbLog is able to achieve high accuracy using a relatively small number of training examples. This suggests that DeepProbLog is able to effectively leverage the logical rules and neural networks to generalize well to new data, even when the available training data is limited.

I. Introduction

ARTIFICIAL intelligence tasks often require both low-level perception and high-level reasoning, but integrating the two remains an open challenge in the field. Today, low-level perception is typically achieved by deep neural networks, while high-level reasoning is typically handled using logical and probabilistic representations and inference. Even if deep learning can create intelligent systems used to interpret images, text and speech with unprecedented accuracy, there is a growing awareness of its limitations: deep learning requires large amounts of data to train a network and the models are black-boxes that do not provide explanations and cannot be modified by domain experts.

The abilities of deep learning and probabilistic logic approaches are complementary: deep learning excels at low-level perception and probabilistic logic excels at high-level reasoning. Recently, there has been a lot of progress in both deep learning and high-level reasoning areas and today there exists approaches able to integrate logical and probabilistic reasoning with statistical learning.

DeepProbLog [1] is one possible approach. It is a neural probabilistic logic programming language that incorporates deep learning by means of neural predicates. With DeepProbLog, instead of integrating reasoning capabilities into a complex neural network architecture, the authors have decided to start from an existing probabilistic logic programming language, ProbLog [2], that has been extended with the neural predicates. In this way, the framework exploits the full expressiveness and strengths of general-purpose neural networks and expressive probabilistic-logical modeling and reasoning and can be trained end-to-end based on examples.

In this report, we focus on the application of DeepProbLog to the multi-digit MNIST octal-division task, a computer vision problem that involves dividing two multi-digit octal numbers. This task is challenging due to the large number of possible input-output pairs and the need to apply high-level reasoning to perform the division. Moreover the training set contains only single-digit numbers and the program is not explicitly trained on multi-digit numbers.

We begin by providing an overview of DeepProbLog and its key features in Section II. We then describe the multi-digit MNIST octal-division task in detail and explain how it can be formulated as a DeepProbLog program in Section III. Finally, in Section IV we present experimental results that demonstrate the effectiveness of the DeepProbLog approach on this task.

Overall, our goal is to highlight the potential of DeepProbLog as a powerful tool for solving complex problems in artificial intelligence and machine learning, particularly those that require the integration of logical and probabilistic reasoning with statistical learning.

II. DeepProbLog

In this section, we explain briefly the basics of probabilistic logic programming using ProbLog and how to extend it to obtain DeepProbLog.

A ProbLog program is made of a set of ground probabilistic facts $\mathcal{F}$ of the form $p :: f$ where p is a probability and f is a ground atom, and a set of rules $\mathcal{R}$. One extension introduced for convenience that is nothing else than syntactic sugar are the annotated disjunctions (ADs). An annotated disjunction is an expression of the form

$$p_1 :: h_1; \ldots; p_n :: h_n \text{: } -b_1, \ldots, b_m. \tag{1}$$

where the p_i are probabilities that sum to at most one, the h_i are atoms and the b_j are literals. Given the AD of Eq. 1, when all b_i hold then one of the heads h_j will be true with probability p_j or none of them with probability $1 - \sum p_i$. Several of the h_i may be true at the same time if they also appear as heads of other ADs or rules. Since ADs are syntactic sugar, they do not change the expressivity power of ProbLog and can be alternatively modeled as facts and logical rules [3].

While in ProbLog probabilities are explicitly specified as part of probabilistic facts or ADs, in DeepProbLog probabilities are specified through neural networks. A DeepProbLog program is a ProbLog program that is extended with a set of ground neural annotated disjunctions (nADs) of the form

$$nn(m_q, \vec{t}, \vec{u}) :: q(\vec{t}, u_1); \ldots; q(\vec{t}, u_n) : -b_1, \ldots, b_m \tag{2}$$

where nn is a reserved keyword that stands for 'neural network' and m_q is the identifier of the neural network model, $\vec{t} = t_1, \ldots, t_k$ is a vector of ground terms representing the

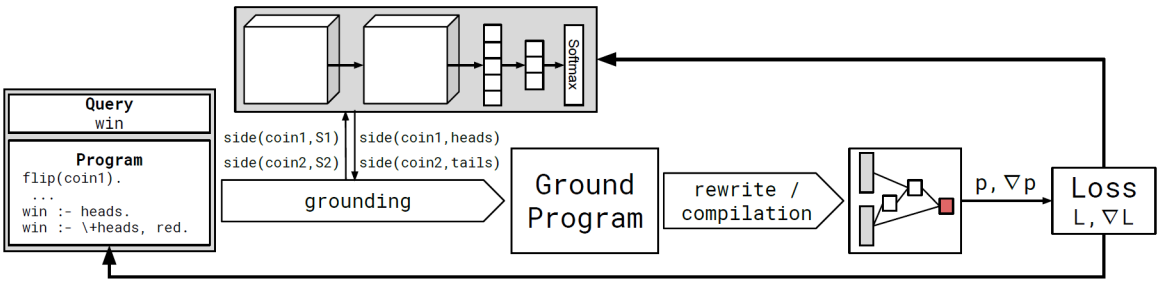


Fig. 1: The learning pipeline [2].

inputs of the neural network for predicate q, $\vec{u} = u_1, \ldots, u_n$ is the vector of the possible output values of the neural network and b_i are atoms. The NN defines a probability distribution over its output values $\vec{u}$ given the input $\vec{t}$. The neural network could be of any type, e.g. a recurrent or a convolutional network, but its output layer, which feeds the corresponding neural predicate, needs to be normalized. Note that a neural AD realizes a regular AD

$$p_1 :: q(\vec{t}, u_1); \ldots; p_n :: q(\vec{t}, u_n) : -b_1, \ldots, b_m \tag{3}$$

after a forward pass on the neural network.

A. DeepProbLog inference

Inference in DeepProbLog closely follows that in ProbLog. ProbLog inference proceeds in four steps. The first step is the grounding step, in which the logic program is grounded with respect to the query, generating all ground instances of clauses the query depends on. This step uses backward reasoning to determine which ground rules are relevant to derive the truth value of the query, and may perform additional logical simplifications that do not affect the query's probability.

The second step rewrites the ground logic program into a formula in propositional logic that defines the truth value of the query in terms of the truth values of probabilistic facts. We can calculate the query success probability by performing weighted model counting (WMC) on this logic formula.

The third step is knowledge compilation that compiles the logic formula into a Sentential Decision Diagram (SDD, [4]), a form that allows for efficient weighted model counting. SDDs are a subset of deterministic decomposable negational normal forms (d-DNNFs) and, as these ones, allow for polytime model counting.

The fourth step transforms the SDD into an arithmetic circuit (AC). The AC has the probabilities of the probabilistic facts on the leaves and is obtained replacing the OR nodes with addition and the AND nodes with multiplication. The AC is then evaluated to calculate the WMC.

Inference in DeepProbLog works as in ProbLog, the only difference is that we need to instantiate nADs and neural facts respectively to regular ADs and probabilistic facts. During the grounding step, we obtain ground nADs and ground neural facts. The concrete parameters are determined subsequently by making a forward pass on the relevant neural network with the ground input.

B. Learning in DeepProbLog

Learning in DeepProbLog consists in jointly train the learnable parameters in the logic program, i.e. the parameters of probabilistic facts, and learnable parameters of the neural network in DeepProbLog programs. DeepProbLog programs are trained in a discriminative training setting, which is called *learning from entailment* [5]. This approach proceeds as follows: given a DeepProbLog program with parameters X, a set Q of pairs (q, p) with q a query and p its desired success probability, and a loss function L, compute:

$$\arg\min_{\vec{x}} \frac{1}{|Q|} \sum_{(q,p)\in Q} L(P_{X=\vec{x}}(q), p) \tag{4}$$

In contrast to the approach proposed by Gutmann et al. [6], gradient descent is used rather than EM, as this method allows for seamless integration with neural network training. It is worth noting that we can use the same AC that ProbLog uses for inference for gradient computations as well: this AC is a differentiable structure, as it is composed of addition and multiplication operations only. The framework relies on the automatic differentiation capabilities already available in ProbLog to derive the gradients. More specifically, to compute the gradient with respect to the probabilistic logic program part, it relies on Algebraic ProbLog (aProbLog [7]), a generalization of the ProbLog language, and inference to arbitrary commutative semirings, including the gradient semiring [8]. An overview of the approach is shown in Figure 1. Given a DeepProbLog program, its neural network models, and a query used as training example, we first ground the program with respect to the query, getting the current parameters of nADs from the external models, then use the ProbLog framework to compute the loss and its gradient, and finally use these to update the parameters in the neural networks and the probabilistic program.

In the following section, we present and discuss the multi-digit MNIST octal-division task and how can be solved using DeepProbLog.

III. Multi-digit MNIST Octal-division Task

The multi-digit MNIST octal-division task is a problem in machine learning that involves dividing two multi-digit octal numbers. The problem is to predict the quotient of the division. The input to the problem is a pair of multi-digit octal numbers, which are represented as images in the MNIST dataset. We

constrain the problem assuming that the second number is an integer divisor of the first. Moreover, we generate the training set in such a way that it contains only single-digit numbers in order to show how the program can be extended to multi-digit numbers without being explicitly trained on them.

Formulating the multi-digit MNIST octal-division task as a DeepProbLog program involves defining logical rules that specify the relationships between the input and output. Specifically, we define the rules `number([H|T],Acc,Result)`, that constructs a decimal number from a list of octal digits, `dec2oct(S,X)`, that converts a decimal number to an octal number, and `division(X,Y,Z)`, that predicts the division between X and Y, where X and Y are lists of images of handwritten digits from the MNIST dataset that represent two base-8 numbers, and Z is the base-8 number corresponding to the the quotient of the division. While this last predicate can be learned directly by a standard neural classifier, such an approach cannot incorporate background knowledge such as the definition of the octal division between two numbers.

```
nn(mnist_net,[X],Y,[0,1,2,3,4,5,6,7]) :: digit(X,Y).

number([],Result,Result).
number([H|T],Acc,Result) :- digit(H,Nr),
                            Acc2 is Nr+8*Acc,
                            number(T,Acc2,Result).
number(X,Y) :- number(X,0,Y).

dec2oct(0,0).
dec2oct(S,X) :- X > 0 ,
                X1 is X // 8 ,
                dec2oct(S1,X1),
                S0 is X mod 8 ,
                S is S0 + S1 * 10 .

division(X,Y,Z) :- number(X,Xdec), number(Y,Ydec),
    Ydec > 0, Zdec is Xdec//Ydec, dec2oct(Z, Zdec).
```

Fig. 2: The DeepProbLog program.

The DeepProbLog program implemented to solve the task is shown in Fig. 2. The program specifies the neural annotated disjunction

$$nn(mnist\_net, [X], Y, [0, 1, 2, 3, 4, 5, 6, 7]) :: digit(X, Y).$$

where nn is a reserved functor, $mnist\_net$ is a neural network that classifies MNIST digits defining a probability distribution over the domain $1, 2, 3, 4, 5, 6, 7$ and $digit$ is the corresponding neural predicate. The output layer of the network that feeds the $digit$ neural predicate should be normalized in order to get proper probability values. In general, the neural network, apart from the output layer, could be of any kind, e.g., a recurrent network for sequence processing or a convolutional network for image processing. We decided to implement the standard convolutional neural network used for MNIST images [9], as it is the more appropriate neural network architecture for this dataset.

The DeepProbLog program proceeds as follows: given a training example composed of two lists of MNIST images and the result of the division in base-8, first we need to obtain the dividend and divisor from the list of images. This is done passing the images to the neural network and building the base-10 numbers from the single digits. Then the division can be calculated applying the standard Prolog operator for the integer division between decimal numbers and the result converted back to a base-8 number. The learning process can be applied based on the supervised label and on the obtained result.

A. Training and test sets

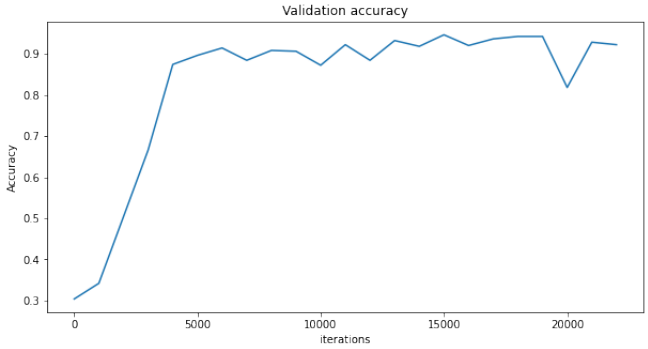
Starting from the MNIST dataset, we construct the training and test datasets as follows. The training set is composed of pairs of images that represent single-digit base-8 numbers and the test set is composed of pairs of lists of images that represent three-digit base-8 numbers. In both cases, the second number is a divisor of the first one. After removing the images representing digit 8 or 9 from the MNIST training and test sets, the algorithm proceeds constructing a random number n_1. Then, a random divisor n_2 is constructed: we select a uniform random number in the range $[1, n_1]$ and we select the nearest number that is a divisor of n_1. The images used to construct the two numbers are removed from the MNIST dataset. Proceeding in this way until no more pairs can be constructed, we managed to generate a training set made of 22,958 pairs and a test set made of 1,462 pairs. In this way, there are no repeated images in the training and test sets but not all the MNIST images will be used: we used 45,916 of the available 48,200 training images of MNIST (95.26%) and 7,835 of the available 8,017 test images of MNIST (97.73%). Note that the divisors in the test set can have up to three digits, whereas the dividends are forced to have exactly three digits. Finally, the train and test queries are generated based respectively on the train and test sets.

B. Implementation details

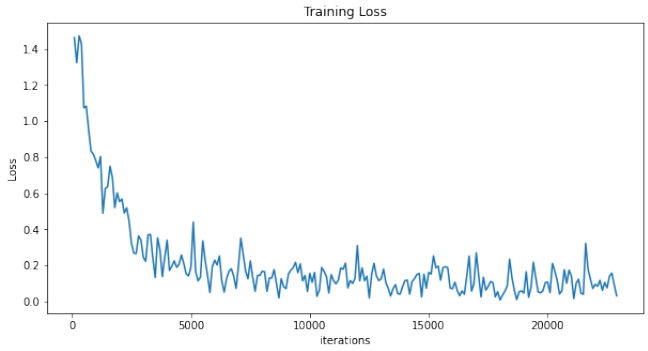
In our implementation, the neural network model used to classify the MNIST digit images has a total of 44,256 parameters and is a basic architecture based on the one discussed in [9]. The model architecture can be described as follows: it consists of 2 2D-convolutional layers both with a kernel size of 5 and with respectively input channels of 1 and 6 and output channels of 6 and 16; the convolutional layers are stacked with a 2D max pooling layer, with a kernel size of 2 and stride of 2, and a rectified linear unit layer. After these layers, the model has 3 fully connected layers of sizes 120, 84 and 8 with a rectified linear unit layer between them. The last layer is followed by a softmax layer in order to get a probability value. The learning process optimizes the cross-entropy loss between the predicted and desired query probabilities as implemented by the function $train\_model$ that is part of the DeepProbLog framework, performing gradient accumulation instead of mini-batching. As optimizer we used Adam with a learning rate of 0.001 for the network parameters.

IV. Results

In this section, we briefly present the results obtained evaluating the performance of the DeepProbLog framework on the multi-digit MNIST octal-division task.



(a) Validation accuracy



(b) Training loss

Fig. 3: Multi-digit MNIST octal-division task learning curves: accuracy on the multi-digit test set 3a and training loss on the single-digit training set 3b.

We trained the model on a training set made of 22,958 pairs and evaluated it on a test set made of 1,462 pairs. The learning process managed to achieve 97% of accuracy on the test set after around 15,000 iterations as shown in Fig. 3a. It is worth noting that the process is able to reach a validation accuracy of about 90% after just 5,000 iterations. Fig. 3b shows the loss obtained on the training set during the learning process: the loss significantly decreases during the first 5,000 iterations and then it tends to oscillate under the value of 0.2 in agreement with the validation accuracy.

This suggests that DeepProbLog is able to effectively leverage the logical rules and the neural network, handling multi-digit numbers in the test set without being explicitly trained on them, converging faster and requiring a smaller training set compared to traditional neural network models. These results demonstrates the effectiveness of DeepProbLog in solving challenging Deep Learning problems.

V. Conclusions

As illustrated in the previous sections, DeepProbLog is a powerful framework where general-purpose neural networks and expressive probabilistic-logical modeling and reasoning are integrated in a way that exploits the full expressiveness and strengths of both worlds. For these reasons, the framework is suitable for solving problems where both low-level data processing using deep networks and high-level reasoning using symbolic approaches are needed.

After an explanation of DeepProbLog, we have illustrated how to create and train a DeepProbLog program to solve the multi-digit MNIST octal-division task. Our experiments showed that DeepProbLog was able to accurately recognize the MNIST digits and perform the division operation accurately, even when presented with multi-digit numbers.

Compared to standard neural classifiers, DeepProbLog requires fewer iterations to converge, due to the fact that exploits symbolic reasoning, and provides a transparent and interpretable framework, which is particularly important in applications where decisions have significant consequences.

Overall, DeepProbLog is a powerful framework for solving complex probabilistic modeling tasks, and it has the potential to advance the state-of-the-art in various domains, including computer vision, natural language processing, and robotics.

References

[1] Robin Manhaeve et al. “DeepProbLog: Neural Probabilistic Logic Programming”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf.

[2] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. “ProbLog: A Probabilistic Prolog and Its Application in Link Discovery”. In: Jan. 2007, pp. 2462–2467.

[3] Luc De Raedt and Angelika Kimmig. “Probabilistic (logic) programming concepts”. In: *Machine Learning* 100.1 (June 2015), pp. 5–47. ISSN: 1573-0565. DOI: 10.1007/s10994-015-5494-z. URL: https://doi.org/10.1007/s10994-015-5494-z.

[4] Adnan Darwiche. “SDD: A New Canonical Representation of Propositional Knowledge Bases”. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*. IJCAI’11. Barcelona, Catalonia, Spain: AAAI Press, 2011, pp. 819–826. ISBN: 9781577355144.

[5] Michael Frazier and Leonard Pitt. “Learning From Entailment: An Application to Propositional Horn Sentences”. In: *Machine Learning Proceedings 1993*. San Francisco (CA): Morgan Kaufmann, 1993, pp. 120–127. ISBN: 978-1-55860-307-3. DOI: https://doi.org/10.1016/B978-1-55860-307-3.50022-8. URL: https://www.sciencedirect.com/science/article/pii/B9781558603073500228.

[6] Bernd Gutmann et al. “Parameter Learning in Probabilistic Databases: A Least Squares Approach”. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Walter Daelemans, Bart Goethals, and Katharina Morik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 473–488. ISBN: 978-3-540-87479-9.

[7] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. “An Algebraic Prolog for Reasoning about Possible Worlds”. In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. AAAI’11. San Francisco, California: AAAI Press, 2011, pp. 209–214.

[8] Jason Eisner. “Parameter Estimation for Probabilistic Finite-State Transducers”. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL ’02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 1–8. DOI: 10.3115/1073083.1073085. URL: https://doi.org/10.3115/1073083.1073085.

[9] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.