Neuro-Symbolic Reasoning for Handwritten Expression Evaluation

Kangrui Ren
School of Computer Science
McGill University
Montreal, Quebec, Canada
kangrui.ren@mail.mcgill.ca

ABSTRACT

In this report, we mainly use a language called DeepProbLog, which can integrate logical reasoning and neural networks effectively. We conducted experiments on reverse logical reasoning based DeepProbLog. At the same time, we found that gradient semiring are limited in solving non-basic arithmetic reasoning. During the experiment, the additional interference elements affected the accuracy of reasoning. We made corresponding explanations for the phenomenon and provide thoughts on follow-up questions.

1 INTRODUCTION

1.1 Integration of perception and reasoning

In recent years, Deep Learning has gradually become a research hot-spot and mainstream development direction in the field of Artificial Intelligence. Deep Learning is a computing model composed of multiple processing layers that is able to train data with multiple abstract levels. At the same time, it also brings revolutionary progress to machine learning and computer vision.

Integration of low-level perception with high-level reasoning is one of the greatest challenge for a long time and several computer scientists [2, 11, 15] are looking forward to modernize past ideas in neural-symbolic integration in order to find approaches to combine statistical learning with probabilistic logic programming effectively. Although Deep Learning and Neural Networks are extraordinarily promising in solving low-level perception problems with several breakthroughs in Deep Learning in recent researches and high-level reasoning problems can be optimally addressed using logical and probabilistic representations and inference nowadays, joining high-level probabilistic reasoning in full flexibility with the representation capability of deep neural networks is still an unsolved problem.

In our work, we used a new language called DeepProbLog [13] that is developed based on an existing probabilistic logic programming language, ProbLog [5], with extended capability to process neural predicates. Instead of integrating reasoning capabilities into a complex Neural Network architecture, DeepProbLog did extremely simple work. The basic idea is that since an atomic expression of form $q(t_1, ..., t_n)$ have probability p, then once the output of the Neural Network on the expression can be interpreted as a probability, the output of neural network components can be encapsulated in the form of neural predicates.

It is worth noting that [13] verified that DeepProbLog can be used for additive reasoning. To be specific, it can infer the sum of two numbers (

Handwritten digit images) accurately after training the model with related logical rule (sample rule are shown in Figure 1).

```
nn(mnist_net,[X],Y,[0,1,2,3,4,5,6,7,8,9]) :: digit(X,Y).

addition(X,Y,Z) :- digit(X,X2), digit(Y,Y2), Z is X2+Y2.
```

Figure 1: Additive Reasoning Logical Rule

The first line means $digit(X, Y)$ transmits input image X to a Neural Network and recognizes the correct number named Y. The training predicates are in forms of the second line, where X and Y represent the handwritten digits images and Z is the sum of X and Y. Furthermore, this framework can be extended for multi-digits numbers(i.e. X and Y are images lists) without additional training.

Based on this idea, we studied the ability of reverse reasoning of the framework. The baseline is to verify whether DeepProbLog could solve equations, even if it's only linear equation with one unknown. If DeepProbLog can complete this task, we only need to expand the logic rules from the base to solve more complex equations.

By saying "solve equation", we mean our predicates consist of handwritten digits images that can be recognized by the Neural Network with interference images that are not nn-recognizable for the representation of unknowns. For a simple instance, when given a predicate addition([|X|],[|3|],9), we mean an equation X+3=9 and the model should be able to retrieve number 6 for X. In order to achieve this goal, we carried out three experiments in section 4. It is worth noting that when there are unknowns in the input, although the accuracy of forward inference decreases to a certain extent, it can still infer sum accurately. Further evaluation results are displayed on Section 4.4

1.2 LP Concepts

In this section, we are going to review some basic concepts in logical programming that are usually talked in following sections.

A term t can be a mutable variable V, a constant c, or a n-ary functor $f(t_1, t_2, \ldots, t_n)$ where t_i is also a term. An atom is an expression of a n-ary predicate $p(t_1, t_2, \ldots, t_n)$, where t_i is a term. A substitution $S = \{V_1 = t_1, ..., V_n = t_n\}$ is an assignment of terms t_i to variables V_i. A literal is an atom or the negation of an atom. A rule r is an expression of form h: $-$ $b_1, ..., b_n$ where h is an atom and b_i is a literal. It means that h holds when the conjunction of b_i holds (See following section usage). Also we will use the notion ground several times in section 3. Ground means the expressions that do not contain any variables.

1.3 DeepProbLog Program

Now we will simply introduce what a DeepProbLog program is. First we are going to introduce the structure of a ProbLog program.

A ProbLog program consists of a set of rules $\mathcal{R}$ and a set of ground probabilistic facts $\mathcal{F}$ of the form p :: f where p is a probability and f is a ground atom.

A DeepProbLog program is a ProbLog program that is extended with a set of ground neural ADs (nADs). An AD, short for annotated disjunction, is an expression of the form p_1 :: $h1; .....; p_n$ :: h_n: $-b_1; .....; b_m$ where p_i are probabilities that $\sum p_i = 1$, and h_i and b_j are atoms. The meaning of an AD is that whenever all b_j hold, h_i will be true with probability p_i , with all other h_j false.

For example, the following nADs is the basis of our experiment for single digit addition problem.

$$nn(m_{\text{digit}}, \square, [0, \ldots, 9]) :: \texttt{digit}(\square, 0); \ldots; \texttt{digit}(\square, 9).$$

Figure 2: nADs for single digit images

Further details about DeepProbLog program will be explained in Section 3.

2 RELATED WORK

The neuro-symbolic reasoning literature [8] [9] introduced ways on combining logical reasoning with neural networks. These approaches are generally encoding logical terms in Euclidean space to approximate logical reasoning with neural networks. However, these approaches are unable to allow probabilistic reasoning or perception. Also, these methods are always limited only for non-recursive and acyclic logical programs [10]. DeepProbLog takes a different approach and integrates neural networks into a probabilistic logic framework, retaining the full power of both logical and probabilistic reasoning and deep learning.

Recent works are focusing on developing differentiable frameworks for logical reasoning. [2] introduced a differentiable framework by implementing Prolog theorem proving procedure in a differentiable manner and extended it with learning sub-symbolic representation of the existing symbols to handle data noise. In contrary to their idea of using logic only to construct a neural network and focus on learning sub-symbolic representations, DeepProbLog focuses on tight interactions between the two and parameter learning for both the neural and the logic components. [3] introduced a framework to compile a tractable subset of logic programs into differentiable functions and to execute it with neural networks. It provides an alternative probabilistic logic but it has a different and less developed semantics.

A different line of work centers around including background knowledge as a regularizer during training. [6] and [7] use first order logic to specify constraints on the output of the neural network. They use fuzzy logic to create a differentiable way of measuring the number of the output of the neural networks violates these constraints. This is then added as an additional loss term that acts as a regularizer. Most recent works introduce similar method that uses probabilistic logic instead of fuzzy logic, and is thus more similar to DeepProbLog. They also compile the formulas to an SDD for efficiency.

[4] show a different way to combine perception with reasoning. Just as in DeepProbLog, they combine domain knowledge specified as purely logical Prolog rules with the output of neural networks. The main difference is that DeepProbLog deals with the uncertainty of the neural network's output with probabilistic reasoning, while they do this by revising the hypothesis, iteratively replacing the output of the neural network with anonymous variables until a consistent hypothesis can be formed.

An idea similar in spirit to DeepProbLog is that of [1], who introduce a neural network for visual question answering composed out of smaller modules responsible for individual tasks, such as object detection. Whereas the composition of modules is determined by the linguistic structure of the questions, DeepProbLog uses logic programs to connect the neural networks. These successes have inspired a number of works developing (probabilistic) logic formulations of basic deep learning primitives.

3 FRAMEWORK

3.1 DeepProbLog Inference with SDD

In this section, we will explain how the DeepProbLog model is used for certain queries for inference tasks. To begin with, we first introduce how the ProbLog inference works since DeepProbLog inference closely follows that in ProbLog.

The inference generally proceeds in four steps. We used a simple query as for showing the SDD structure (Figure 3).

(1) First we do ground the logic program with respect to the query. In other words, we generate all ground instances of clauses in the program the query depends on.
(2) Next, we rewrite the ground logic program into a propositional logic formula that defines the truth value of the query in terms of the truth values of probabilistic facts.
(3) Then we compile the logic formula into a Sentential Decision Diagram. A SDD is a form that allows for efficient evaluation of the query.
(4) Finally, we evaluate the SDD bottom-up to calculate the success probability of the given query. In other words, we start from the leaves with the probability labels given by the program and calculate probability by performing addition in every OR-node and multiplication in every AND-node.

The following diagram displays the SDD structure during the inference with a simple query: "**addition (train(711), train(10783),2)**" , where train(711) and train(10783) are handwritten digit images.

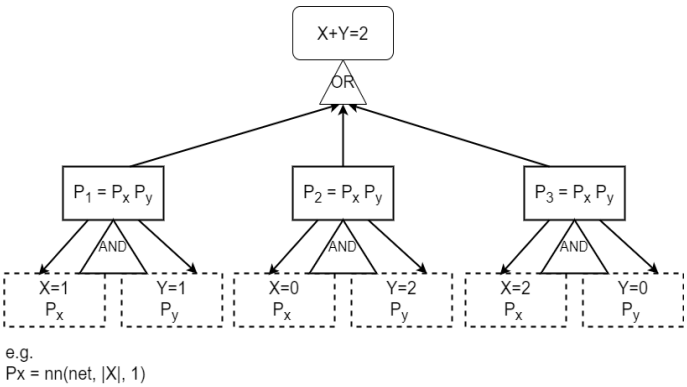

Figure 3: SDD structure

From Figure 3 we have the probability of the image that represents certain number by Neural Network. As this example, we can see that sum 2 can be grounded into 3 possible instances each with corresponding probability. These instances are in conjunction forms for each clause and in disjunction for the final calculation. Original SDD structures are shown below.

Inference in DeepProbLog works nearly the same as the steps described above, except that a forward pass on the neural network components is performed every time when encounter a neural predicate during grounding. Once this occurs, the images are fed into the neural network, after which the resulting scores of their softmax output layer are used as the probabilities of the ground AD.

3.2 Learning Process

3.2.1 Learning Pipeline. To begin with, we will introduce the learning pipeline of the whole program. As mentioned in [14], we used the following formula.

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

where $\mathcal{X}$ is the parameter, Q is a set of pair (q, p) with q for a query and p for a probability, and L is the loss function.

We displayed the learning pipeline of the given query from 3.1 as an example in Figure 4. We could see that we have the DeepProbLog program, the query, and the neural network model. The program is first grounded with respect to the query and we get the parameters for nADs and enters the rewrite and compilation. Then we are going to compute the loss and gradient which are further updated in the neural network.

3.2.2 Gradient Semiring. Instead of using EM, We used gradient descent in the learning. To compute the gradient with respect to the probabilistic logic program, we used Algebraic ProbLog[12], which is a generalized form of ProbLog with ability to process arbitrary commutative semirings, especially gradient semiring.

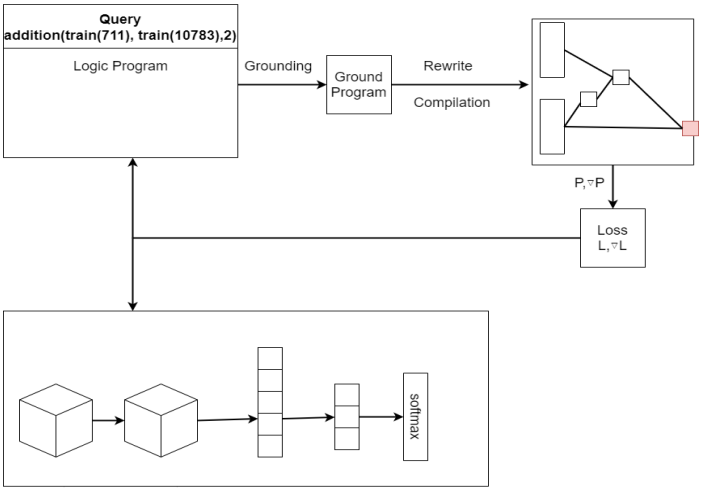

Figure 4: DeepProbLog Learning Pipeline

We will simply introduce aProbLog program. An aProbLog program consists of

(1) A commutative semiring $(\mathcal{A}, \bigoplus, \bigotimes, e^{\bigoplus}, e^{\bigotimes})$
(2) A finite set of ground algebraic facts $\mathbf{F} = \{f_1, ..., f_n\}$
(3) A finite set BK of background knowledge clauses
(4) A labeling function $\alpha : L(F) \rightarrow \mathcal{A}$

It should be noted that Background knowledge clauses are definite clauses, but their bodies may contain negative literals for algebraic facts.Their heads may not unify with any algebraic fact.

In fact, ProbLog uses probability semiring with normal addition and multiplication as operators to compute the probability of certain queries on the corresponding SDD. aProbLog generalized the idea to compute such values based on arbitrary commutative semirings. It uses a labeling function to associate values from the semiring with both facts and their negations, and combines these using semiring addition and multiplication on the SDD. Semiring addition $\bigoplus$, multiplication $\bigotimes$ and the neutral elements with respect to these operations are defined as follows:

$$(a_1, a_2) \bigoplus (b_1, b_2) = (a_1 + b_1, a_2 + b_2)$$

$$(a_1, a_2) \bigotimes (b_1, b_2) = (a_1 b_1, a_2 b_1 + a_1 b_2)$$

$$e^{\bigoplus} = (0, 0)$$

$$e^{\bigotimes} = (1, 0)$$

3.2.3 Gradient descent. In order to apply gradient semiring for gradient descent parameter learning in ProbLog, we convert the program into an aProbLog program, extending the label of each probabilistic fact $p :: f$ to include the probability p as well as the gradient vector of p with respect to the probabilities of all probabilistic facts in the program.

Instead of updating parameters based on gradients computed by aProbLog, the probabilities of neural predicates are in form of a function of the nn parameter. Neural Predicates are functioning as a connector that associate logic and neural. It should be noted that the logic side and neural logic side are independent. The logic side are able to calculate the gradient of the loss with respect to the output of the neural network without awareness of internal parameters. Nevertheless, the gradient with respect to the output can start back-propagation, which can achieve the internal parameter gradients. In our model, Adam (A standard gradient-based optimizer) is used for updating parameters. Also the probabilities of nADs are kept constant during computation and their update makes use of softmax output layer, which ensures that no additional normalization is required since it keeps a normalized distribution.

4 EXPERIMENT

4.1 Experiment Set-up

We performed all of our experiments on a MacBook Pro with 2.3GHz Dual-Core Intel i5, 8 GB RAM. The program and related data are posting on GitHub:

https://github.com/KangruiRen0102/Neural-Symbolic-Reasoning-for-Handwritten-Expression-Evaluation

4.2 Problem Statement

As stated in section 2, we split our experiment into three consecutive sections. The general purpose for this experiment is to verify when two sets of handwritten digit images are given as input, can DeepProbLog infer the sum accurately with interference, that is, some shape images that represent equation unknowns are mixed in the input sets. At the same time, Our final goal is to verify whether DeepProbLog can infer the exact number that the unknown (shape image) represents.

To make it simple, we used a shape image set (There are three kind of shape: circle, triangle, and square) to represent unknowns; details are described in 4.3.

Task1 Single-shape Inference. In this task we are going to measure the ability of DeepProbLog of forward reasoning with one interference image. Instead of using labeled single digits, we train on pairs of images, labeled with the sum of the individual labels. We have two single-image input sets, one is a handwritten digit image and the other is a shape image. For example we have **addition([|Square|],[|3|],6)**, we know in this case we have X+3=6, so the unknown X here, represented by Square, is 3 in fact.

We want to study the influence of the accuracy of image recognition on the final sum result in this task. Also, we compared the result of the task with original single-digit addition.

Task2 Multi-shape Inference. In this task, we measured the ability of DeepProbLog of forward reasoning with multiple interference images. For example: **addition([|Circle|,|2|],[|3|,|Triangle|],48)**, from which we could infer that the circle represents 1 and the triangle represents 6.

We want to study the influence of individual training data size on the final sum result. In other words, we are going to verify whether DeepProbLog can still perform well on multi-shape tasks after training on single-shape data with extensibility.

Task3 Multi-shape Reverse Inference. In this task, we did the same operations as in Task2 except that we are going to find the exact value of the multiple unknowns, whereas we only need to verify the accuracy of sum result in Task2.

However, this task is not well solved during our experiment. We tried to rewrite logical rules based on the probabilistic program, and we find that the ProbLog engine is not able to process additional inputs with additive reasoning. For example, we have **addition([|Circle|,|2|],[|3|,|Triangle|],48, 1, 6)** as a training data. The gradient semiring also need modification to allow this back inference. This task will be regarded as future work for further study.

4.3 Data Collection

Our two data sets are used to train handwritten digits and shapes respectively.

The data of handwritten digits come from MNIST data set, and we use the images in the data set as the main input of the model. The shape data come from the kaggle image data set, which contains 300 pictures for circles, squares, and triangles. We use the shapes in the data set as the unknown number in the random position of the input image sequence. Although they are unknown in the operation, the number represented by the image is fixed when the sum of two groups of images is given during the generation process. The model needs to give a specific value to it in the process of inference. In order to get the specific number of the shape to reach the correct sum in generating our dataset, we randomly assign it with a number from uniform distribution. Specifically, we divide the data into training data and test data.

- For **Task1 Single-shape Inference**, the training data and test data are 3-ary tuples. There

are two image arrays of single element (one of which comes from the shape data set, and the other comes from MNIST dataset), and a integer for accurate sum. For example, we have **addition([|Circle(20)|],[|7|],9)**.

- For **Task2 Multi-shape Inference**, we generated two training datasets (single-shape set and multi-shape set) with the same test datasets . Training data in single-shape set are the same as the example in Task1, whereas those in multi-shape set are 3-array tuple with multiple images in two input image arrays; test data are also in this form. For instance: **addition([|Square(2)|,|9|],[|3|,|Circle(8)|],68)**. We generate two training sets because we are going to study whether DeepProbLog has extensibility when input images contain interference.

- For **Task3 Multi-shape Reverse Inference**, we also use the training data from Task2, but we use the 5-ary tuple as test data. Compared with task 2, these tuples also contains the specific number represented by two images. Examples are also displayed in 4.2

4.4 Evaluation

Figure 5 compares the result of accuracy of original single-digit addition with that of single-shape addition (Task 1) for 30000 iterations.

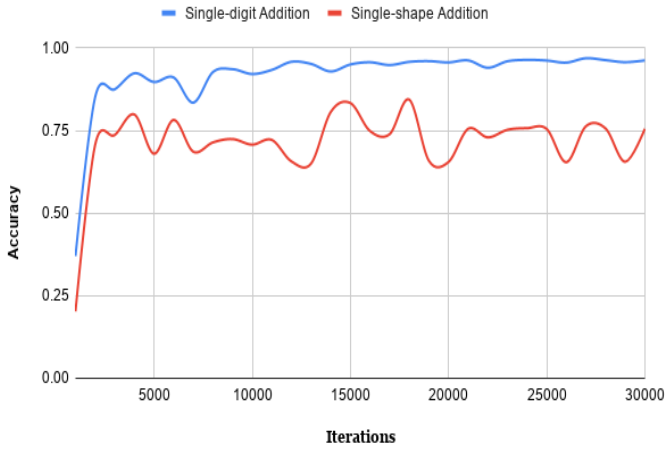

Figure 5: Single-shape Accuracy Comparison

We can see from the figure that original single-digit addition is apparently more accurate than the addition with one interference image. In fact, the final accuracy for single-digit addition is around 0.96, whereas the final accuracy for single-shape addition is around 0.79. Also we could see that the line for single-shape addition fluctuates more obviously, which might due to the reason that the probability distribution generated by the neural network is not stable, and back-propagation is harder to get improvement for shape images.

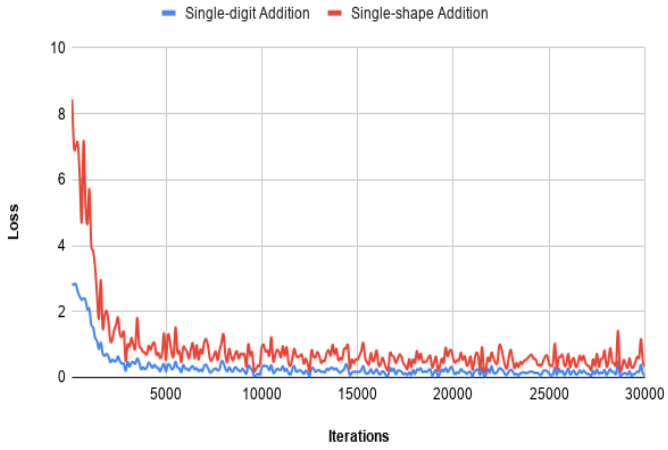

Figure 6: Single-shape Loss Comparison

Figure 6 shows that the loss of single-shape addition is obviously greater than that of single-digit addition during the training process. This also indicates that interference images will lead to lower accuracy and higher loss.

Figure 7 and 8 displayed the performance of Task2. Figure 8 shows the accuracy of testing the same dataset with the two different training sets. It's apparent that using Multi-shape training set provide higher accuracy, and DeepProbLog does not offer great extensibility when input sets have interference. Although Multi-shape training gives higher accuracy, it cost almost 10 times longer for the whole training time compared to the Single-shape training. This phenomenon is easy to explain. The training pipeline in 3.2 gives the intuition that grounding and calculation of loss and gradient will take more time when the size of input increases, leading the whole process slow down.

Figure 7 shows the loss of single-shape addition and multi-shape addition with single-shape training set. It displays that since they are using the same training set, the loss does not vary too much during the whole training process. This proves that the accuracy and loss of the model are strictly related with the number of interference, as testing data displayed non-ideal results when training data set are the same.

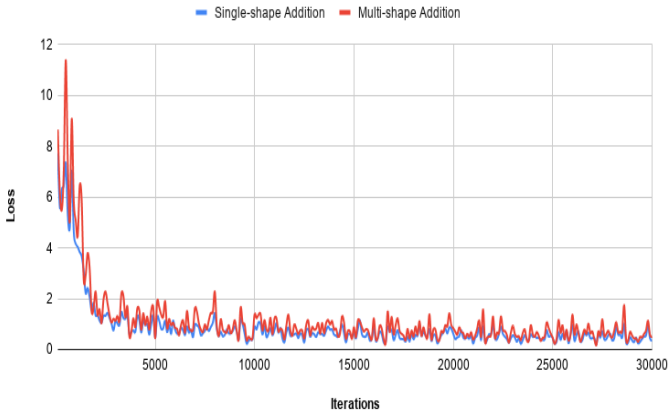

Figure 7: Loss for Multi-shape Addition and Single-shape Addition

Figure 8: Accuracy for two types of training in Multi-shape Addition

5 CONCLUSION

In general, this report briefly analyzes the advantages of DeepProbLog as a language that combines logical reasoning and neural networks. At the same time, we discussed the basic operation of the pipeline. Our main purpose in the trial phase is to verify its ability of reverse reasoning. Although the third task did not produce optimal results, we found the limitations of gradient semiring and the ProbLog engine still needs further expansion to

process complex logic rules. In the first two well-performed tasks, we demonstrated the powerful reasoning ability of DeepProbLog. When the input contains interference, it can still predict the result accurately. When the accuracy decreases with increasing interference, one possible solution is to increase the digit length of the training data.

ACKNOWLEDGEMENT

This work is supervised by Prof. Xujie Si at McGill University.

REFERENCES

[1] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 39–48.

[2] Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. 2017. Programming with a differentiable forth interpreter. In *International conference on machine learning*. PMLR, 547–556.

[3] William W Cohen, Fan Yang, and Kathryn Rivard Mazaitis. 2017. Tensorlog: Deep learning meets probabilistic dbs. *arXiv preprint arXiv:1707.05390* (2017).

[4] Wang-Zhou Dai, Qiu-Ling Xu, Yang Yu, and Zhi-Hua Zhou. 2018. Tunneling neural perception and logic reasoning through abductive learning. *arXiv preprint arXiv:1802.01173* (2018).

[5] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery.. In *IJCAI*, Vol. 7. Hyderabad, 2462–2467.

[6] Michelangelo Diligenti, Marco Gori, and Claudio Sacca. 2017. Semantic-based regularization for learning and inference. *Artificial Intelligence* 244 (2017), 143–165.

[7] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. 2017. Logic tensor networks for semantic image interpretation. *arXiv preprint arXiv:1705.08968* (2017).

[8] Artur S d'Avila Garcez, Krysia B Broda, and Dov M Gabbay. 2012. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media.

[9] Barbara Hammer and Pascal Hitzler. 2007. *Perspectives of neural-symbolic integration*. Vol. 77. Springer.

[10] Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. 1999. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence* 11, 1 (1999), 45–58.

[11] William W Cohen Fan Yang Kathryn and R Mazaitis. 2018. Tensorlog: Deep learning meets probabilistic databases. *Journal of Artificial Intelligence Research* 1 (2018), 1–15.

[12] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. 2011. An algebraic Prolog for reasoning about possible worlds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 25.

[13] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2018. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems* 31 (2018), 3749–3759.

[14] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. 2016. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10, 2 (2016), 1–189.

[15] Tim Rocktäschel and Sebastian Riedel. 2017. End-to-end differentiable proving. *arXiv preprint arXiv:1705.11040* (2017).