# Extending Answer Set Programs with Neural Networks

Zhun Yang
Arizona State University, Tempe, AZ, USA
zyang90@asu.edu

The integration of low-level perception with high-level reasoning is one of the oldest problems in Artificial Intelligence. Recently, several proposals were made to implement the reasoning process in complex neural network architectures. While these works aim at extending neural networks with the capability of reasoning, a natural question that we consider is: can we extend answer set programs with neural networks to allow complex and high-level reasoning on neural network outputs? As a preliminary result, we propose NeurASP — a simple extension of answer set programs by embracing neural networks where neural network outputs are treated as probability distributions over atomic facts in answer set programs. We show that NeurASP can not only improve the perception accuracy of a pre-trained neural network, but also help to train a neural network better by giving regularization through logic rules. However, training with NeurASP implementation takes much more time than pure neural network training due to the internal use of a symbolic reasoning engine. For future work, we plan to investigate the potential ways to solve the scalability issue of NeurASP implementation. One way is to embed logic programs directly in neural networks. On this route, we plan to design a SAT solver using neural networks and extend such a solver to allow logic programs.

## 1 Introduction and Problem Description

The integration of low-level perception with high-level reasoning is one of the oldest problems in Artificial Intelligence. This topic is revisited with the recent rise of deep neural networks. Several proposals were made to implement the reasoning process in complex neural network architectures, e.g., [5, 22, 6, 8, 24, 19, 16]. However, it is still not clear how complex and high-level reasoning, such as default reasoning [21], ontology reasoning [2], and causal reasoning [20], can be successfully computed by these approaches. The latter subject has been well-studied in the area of knowledge representation (KR), but many KR formalisms, including answer set programming (ASP) [15, 3], are logic-oriented and do not incorporate high-dimensional vector space and pre-trained models for perception tasks as handled in deep learning, which limits the applicability of KR in many practical applications involving data and uncertainty. A natural research question we consider is: can we extend answer set programs with neural networks to allow complex and high-level reasoning on information provided in vector space?

Our research tries to answer this question. To start with, we extended $\text{LP}^{\text{MLN}}$, a probabilistic extension of ASP, with neural networks by turning neural network outputs into weighted rules in $\text{LP}^{\text{MLN}}$. However, there is a technical challenge: the existing parameter learning method of $\text{LP}^{\text{MLN}}$ is too slow to be coupled with typical neural network training. This motivates us to consider a simpler probabilistic extension of ASP, for which we design and implement an efficient parameter learning method.

In this research summary, we present our preliminary work — NeurASP, which is a simple and effective way to integrate sub-symbolic and symbolic computation under stable model semantics while the neural network outputs are treated as the probability distribution over atomic facts in answer set programs. We demonstrate how NeurASP can be useful for some tasks where both perception and reasoning are required, and show that NeurASP can not only improve the perception accuracy of a pre-

F. Ricca, A. Russo et al. (Eds.): Proc. 36th International Conference
on Logic Programming (Technical Communications) 2020 (ICLP 2020)
EPTCS 325, 2020, pp. 313–322, doi:10.4204/EPTCS.325.41

trained neural network, but also help to train a neural network better by giving restrictions through ASP rules.

The biggest issue we are encountering now is that training with NeurASP implementation still takes much more time than pure neural network training due to the internal use of a symbolic reasoning engine (i.e., CLINGO in our case). We plan to investigate two directions to resolve this issue. One is to compute NeurASP in a circuit. For example, we can turn an NeurASP program into a Probabilistic Sentential Decision Diagram (PSDD) [10] so that the probability and gradient computation would take linear time in every iteration. The challenge for this route is how to construct a circuit efficiently. The other direction is to embed the logic reasoning part completely in neural networks without referring to any symbolic reasoning engines. The challenge is how to embed logic rules in neural networks while maintaining the expressivity.

Besides, we plan to apply NeurASP to domains that require both perception and reasoning. The first domain we are going to investigate is visual question-answering and we limit our attention to the problems whose reasoning can be potentially represented in ASP. Some well-known datasets in this domain include NLVR2 [25], CLEVR [7], and CLEVRER [28]. We plan to start with replacing the reasoning part in existing works with NeurASP and analyze the pros and cons of applying NeurASP on those visual reasoning domains. We also plan to apply NeurASP to predict whether a vulnerability will be exploited so that the knowledge from domain experts can help neural network training.

The paper will give a summary of my research, including some background knowledge and reviews of existing literature (Section 2), goal of my research (Section 3), the current status of my research (Section 4), the preliminary results we accomplished (Section 5), and some open issues and expected achievements (Section 6). The implementation of our prelimilary work — NeurASP, as well as codes used for the experiments, is publicly available online at

`https://github.com/azreasoners/NeurASP`.

## 2 Background and Overview of the Existing Literature

Recent years have observed the rising interests of combining perception and reasoning.

DeepProbLog [17] extends ProbLog with neural networks by means of neural predicates. We follow similar idea to design NeurASP to extend answer set programs with neural networks. Some differences are: (i) The computation of DeepProbLog relies on constructing an SDD whereas we use an ASP solver internally. (ii) NeurASP employs expressive reasoning originating from answer set programming, such as defaults, aggregates, and optimization rules. This not only gives more expressive reasoning but also allows the more semantic-rich constructs as guide to learning. (iii) DeepProbLog requires each training data to be a single atom, while NeurASP allows each training data to be arbitrary propositional formulas.

Xu et al. [26] used the semantic constraints to train neural networks better, but the constraints used in that work are simple propositional formulas whereas we are interested in answer set programming language, in which it is more convenient to encode complex KR constraints. Logic Tensor Network [6] is also related in that it uses neural networks to provide fuzzy values to atoms.

Another approach is to embed logic rules in neural networks by representing logical connectives by mathematical operations and allowing the value of an atom to be a real number. For example, Neural Theorem Prover (NTP) [22] adopts the idea of dynamic neural module networks [1] to embed logic conjunction and disjunction in and/or-module networks. A proof-tree like end-to-end differentiable neural network is then constructed using Prolog's backward chaining algorithm with these modules. Another

method that also constructs a proof-tree like neural network is TensorLog [5], which uses matrix multiplication to simulate belief propagation that is tractable under the restriction that each rule is negation-free and can be transformed into a polytree.

Graph neural network (GNN) [9] is a neural network model that is gaining more attention recently. Since a graph can encode objects and relations between objects, by learning message functions between the nodes, one can perform certain relational reasoning over the objects. For example, in [19], it is shown that GNN can do well on Sudoku, but the input there is not an image but a textual representation. Another work NeuroSAT [23] shows that GNN can solve general small SAT problems after only being trained as a classifier to predict satisfiability. However, this is still restrictive compared to the more complex reasoning that KR formalisms provide.

Neuro-Symbolic Concept Learner [18] separates between visual perception and symbolic reasoning. It shows the data-efficiency by using only 10% of the training data and achieving the state-of-the-art 98% accuracy on CLEVR dataset. Our preliminary result NeurASP is similar in the sense that using symbolic reasoning, we could use fewer data to achieve a high accuracy.

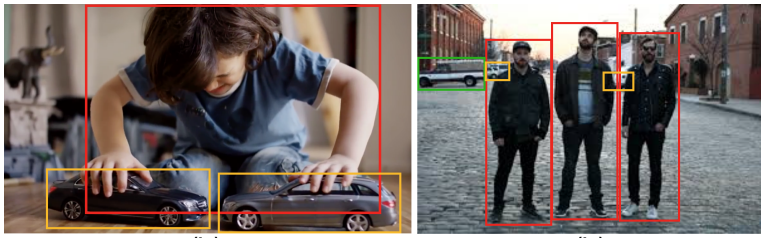
LP$^{\text{MLN}}$ [11] serves as the foundation of our research. We started with extending LP$^{\text{MLN}}$ with neural networks since existing LP$^{\text{MLN}}$ parameter learning is already by the gradient descent method [12] as used in the neural network training. However, there is a technical challenge: the previous parameter learning method does not scale up to be coupled with typical neural network training. This motivates us to consider a fragment of LP$^{\text{MLN}}$ first, for which we design and implement an efficient parameter learning method. It turns out that this fragment is general enough to cover the ProbLog language as well as enjoys the expressiveness of full answer set programming language.

## 3 Goal of the Research

The goal of our research is to develop some methods to integrate low-level perception with high-level reasoning so that, in inference tasks, reasoning can help identify perception mistakes that violate semantic constraints while, in learning tasks, a neural network not only learns from implicit correlations from the data but also from the explicit complex semantic constraints expressed by the rules.

To achieve this goal, we investigate the integration of answer set programs with neural networks. We require that such integration should be not only expressive in representation but also efficient in computation. Besides, such integration should be able to apply reasoning in both inference and learning. For example, such integration should be able to reason about relations among perceived objects in Figure 1 and tell that: the cars in ($i_1$) are toy cars since they are smaller than a person, while cars in ($i_2$) are real cars (by default) since there is no evidence to show they are smaller than those persons.

Figure 1: Reasoning about relations among perceived objects



($i_1$) ($i_2$)

## 4 Current Status of the Research

To achieve our goal, we investigate and answer the following three research questions. This research is at a middle phase and the questions below are partially answered.

1. How do we design a formalism that allows complex and high-level reasoning on information provided in vector space?
   *We have one such design,* NeurASP *[27], that extends answer set programs with neural networks, using ideas from DeepProbLog [17] to interface neural networks and ASP (i.e., treating the neural network output as the probability distribution over atomic facts in answer set programs), and using ideas from* $\text{LP}^{\text{MLN}}$ *[12] to design the probability and gradient computation under stable model semantics.*
   *We plan to design a new formalism that is less expressive but more scalable compared to* NeurASP *by doing all the reasoning within neural networks. To start with, we followed the ideas in [26] to define a semantic loss using the logic rules. We represented implication rules by neural network regularizers for small domains including Nqueens problem, Sudoku, and Einstein's puzzle. We plan to design a general way to represent a CNF by a regularizer. We also plan to investigate into the rule forms that can be represented by a regularizer and then use these rule forms to design the new formalism.*

2. How do we implement such a formalism to make it as scalable as possible?
   *The current implementation of* NeurASP *uses* CLINGO *as its internal reasoning engine and uses* PyTorch *to back-propagate the gradients from logic layer to neural networks, which yields the most scalable prototype among all of our trials so far and preserves the expressivity of ASP. Initially, we implemented the prototype of* NeurASP *where* $\text{LP}^{\text{MLN}}$ *[12] is used to compute the probability and gradient in* NeurASP *since* $\text{LP}^{\text{MLN}}$ *is a probabilistic extension of ASP with well-defined probability and gradient computation under stable model semantics. However, the parameter learning method in* $\text{LP}^{\text{MLN}}$ *does not scale up to be coupled with typical neural network training – even for the MNIST addition example proposed in [17]. To resolve this issue, we tried two approaches. First, we tested the idea of turning an answer set program into a Sentential Decision Diagram (SDD) but found that constructing such an SDD through the route "ASP to CNF to SDD" would take exponential time w.r.t. the number of atoms. There needs more research on SDD and possibly its variation that is more suitable for encoding answer set programs. Second, we tried to simplify* $\text{LP}^{\text{MLN}}$ *to its fragment that is simple enough to have efficient computation and also expressive enough to capture all the ASP constructs. This leads to our last version of* NeurASP *to the date.*
   *We are working on the prototype of the new formalism where logic rules are encoded in neural networks. We embedded implication rules in neural networks and successfully solved Nqueens problem, Sudoku, and Einstein's puzzle using neural networks only. We plan to embed CNFs in neural networks and implement a SAT solver using neural network only. Ultimately, we plan to embed answer set programs directly in neural networks. One challenge in this route is how to embed different constructs, such as negation, choice rules, and aggregation, in a neural network.*

3. How do we evaluate and improve such a formalism?
   *We evaluated* NeurASP *in [27] w.r.t. the following domains: common-sense reasoning about image as in Figure 1, MNIST digit addition in [17], solving Sudoku in images in [19], and the shortest path problem in [26]. We showed that* NeurASP *is very expressive and is able to help both neural network inference and training. We also showed that training with* NeurASP *still takes much more time than pure neural network training due to the internal use of a symbolic reasoning engine (i.e.,*

CLINGO).

*We plan to analyze the effect of* NeurASP *on other domains that require both perception and reasoning. The first domain in our agenda is visual question-answering while we limit our attention to those problems whose reasoning can be potentially represented in ASP. Some well-known datasets in this domain include NLVR2 [25], CLEVR [7], and CLEVRER [28]. We plan to start with replacing the reasoning part in the existing work with* NeurASP *and analyze the pros and cons of applying* NeurASP *on those visual reasoning domains. We also plan to apply* NeurASP *to predict whether a vulnerability will be exploited so that the knowledge from domain experts can help neural network training. What's more, since our preliminary results show that a neural network is not always trained better with more constraints, to know how to add constraints in real world problems, we also plan to analyze the effects of different constraints systematically.*

# 5 Preliminary Results Accomplished

We designed the syntax and the semantics of NeurASP, and show how NeurASP can be useful for some tasks where both perception and high-level reasoning provided by answer set programs are required.

## 5.1 NeurASP

In [27], we present a simple extension of answer set programs by embracing neural networks. Following the idea of DeepProbLog [17], by treating the neural network output as the probability distribution over atomic facts in answer set programs, the proposed NeurASP provides a simple and effective way to integrate sub-symbolic and symbolic computation.

In NeurASP, a neural network $M$ is represented by a *neural atom* of the form

$$nn(m(e,t),[v_1,\ldots,v_n]), \tag{1}$$

where (i) $nn$ is a reserved keyword to denote a neural atom; (ii) $m$ is an identifier (symbolic name) of the neural network $M$; (iii) $t$ is a list of terms that serves as a "pointer" to an input tensor; related to it, there is a mapping $\mathbf{D}$ (implemented by an external Python code) that turns $t$ into an input tensor; (iv) $v_1,\ldots,v_n$ represent all $n$ possible outcomes of each of the $e$ random events.

Each neural atom (1) introduces propositional atoms of the form $c=v$, where $c \in \{m_1(t),\ldots,m_e(t)\}$ and $v \in \{v_1,\ldots,v_n\}$. The output of the neural network provides the probabilities of the introduced atoms.

**Example 1** *Let $M_{digit}$ be a neural network that classifies an MNIST digit image. The input of $M_{digit}$ is (a tensor representation of) an image and the output is a matrix in $\mathbb{R}^{1\times 10}$. The neural network can be represented by the neural atom*

$$nn(digit(1,d),\ [0,1,2,3,4,5,6,7,8,9]),$$

*which introduces propositional atoms $digit_1(d)=0$, $digit_1(d)=1$, …, $digit_1(d)=9$.*

A NeurASP *program* $\Pi$ is the union of $\Pi^{asp}$ and $\Pi^{nn}$, where $\Pi^{asp}$ is a set of propositional rules and $\Pi^{nn}$ is a set of neural atoms. Let $\sigma^{nn}$ be the set of all atoms $m_i(t)=v_j$ that is obtained from the neural atoms in $\Pi^{nn}$ as described above. We require that, in each rule $Head \leftarrow Body$ in $\Pi^{asp}$, no atoms in $\sigma^{nn}$ appear in $Head$.

The semantics of NeurASP defines a *stable model* and its associated probability originating from the neural network output. For any NeurASP program $\Pi$, we first obtain its ASP counterpart $\Pi'$ where each neural atom (1) is replaced with the set of rules

$$\{m_i(t)=v_1;\ \ldots\ ;m_i(t)=v_n\}=1 \quad \text{for } i\in\{1,\ldots e\}.$$

We define the *stable models* of $\Pi$ as the stable models of $\Pi'$.

**Example 1 Continued** The ASP counter-part of the neural atom in Example 1 is the following rule.

$$\{digit_1(d)=0;\ \ldots\ ;digit_1(d)=9\}=1.$$

To define the probability of a stable model, we first define the probability of an atom $m_i(t)=v_j$ in $\sigma^{nn}$. Recall that there is an external mapping $\mathbf{D}$ that turns $t$ into a specific input tensor $\mathbf{D}(t)$ of $M$. The probability of each atom $m_i(t)=v_j$ is defined as $M(\mathbf{D}(t))[i,j]$:

$$P_\Pi(m_i(t)=v_j)=M(\mathbf{D}(t))[i,j].$$

The probability of a stable model $I$ of $\Pi$ is defined as the product of the probability of each atom $c=v$ in $I|_{\sigma^{nn}}$, divided by the number of stable models of $\Pi$ that agree with $I|_{\sigma^{nn}}$ on $\sigma^{nn}$. That is, for any interpretation $I$,

$$P_\Pi(I)=\begin{cases}\frac{\prod_{c=v\in I|_{\sigma^{nn}}} P_\Pi(c=v)}{Num(I|_{\sigma^{nn}},\Pi)} & \text{if } I \text{ is a stable model of } \Pi;\\ 0 & \text{otherwise.}\end{cases}$$

where $I|_{\sigma^{nn}}$ denotes the projection of $I$ onto $\sigma^{nn}$ and $Num(I|_{\sigma^{nn}},\Pi)$ denotes the number of stable models of $\Pi$ that agree with $I|_{\sigma^{nn}}$ on $\sigma^{nn}$.

## 5.2 Examples of NeurASP

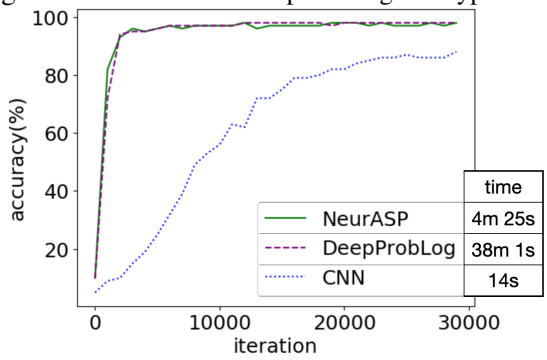
We show how NeurASP can be applied to the following 4 examples where both perception and high-level reasoning provided by answer set programs are required.

- **MNIST Digit Addition** This is a simple example used in [17] to illustrate DeepProbLog's ability for both logical reasoning and deep learning. The task is, given a pair of digit images (MNIST) and their sum as the label, to let a neural network learn the digit classification of the input images. The NeurASP program is as follows.

  $$img(d_1).\quad img(d_2).$$
  $$nn(digit(1,X),[0,1,2,3,4,5,6,7,8,9])\leftarrow img(X).$$
  $$addition(A,B,N)\leftarrow digit_1(A)=N_1,digit_1(B)=N_2,N=N_1+N_2.$$

  Figure 2 shows the accuracy on the test data after each training iteration. The method CNN denotes the baseline used in [17] where a convolutional neural network (with more parameters) is trained to classify the concatenation of the two images into the 19 possible sums. As we can see, the neural networks trained by NeurASP and DeepProbLog converge much faster than CNN and have almost the same accuracy at each iteration. However, NeurASP spends much less time on training compared to DeepProbLog. The time reported is for one epoch (30,000 iterations in gradient descent). This is because DeepProbLog constructs an SDD at each iteration for each training instance (i.e., each pair of images). This example illustrates that generating many SDDs could be more time-consuming than enumerating stable models in NeurASP computation. In general, there is a trade-off between the two methods and other examples may show the opposite behavior.

Figure 2: NeurASP v.s. DeepProbLog v.s. Typical CNN




- **Commonsense Reasoning about Image** We show how expressive reasoning originating from answer set programming, such as recursive definition and defaults can be used in NeurASP inference. We also show that reasoning in NeurASP can help identify perception mistakes that violate semantic constraints, which in turn can make perception more robust.

  Take the problem in Figure 1 as an example. A neural network for object detection may return a bounding box and its classification "car," but it may not be clear whether it is a real car or a toy car. The distinction can be made by applying reasoning about the relations with the surrounding objects and using commonsense knowledge. To reason about the size relationship among objects, we need to define a recursive definition of "smaller than" relationship.

  $$smaller(cat, person).\ \ smaller(person, car).\ \ smaller(person, truck).$$
  $$smaller(X,Y) \leftarrow smaller(X,Z), smaller(Z,Y).$$

  We also need to use default reasoning to assert that, by default, we conclude the same size relationship as above between the objects in bounding boxes $B_1$ and $B_2$.

  $$smaller(I,B_1,B_2) \leftarrow not\ \ {\sim}smaller(I,B_1,B_2), label(I,B_1)=L_1, label(I,B_2)=L_2, smaller(L_1,L_2).$$

  NeurASP allows the use of such expressive reasoning originating from answer set programming.

- **Solving Sudoku in Image** We show that NeurASP alleviates the burden of neural networks when the constraints/knowledge are already given. Instead of building a large end-to-end neural network that learns to solve a Sudoku puzzle given as an image, we can let a neural network only do digit recognition and use ASP to find the solution of the recognized board. This makes the design of the neural network simpler and the required training dataset much smaller. Also, the neural network may get confused if a digit next to 1 in the same row is 1 or 2, but the reasoner can conclude that it cannot be 1 by applying the constraints for Sudoku. What's more, when we need to solve a variant of Sudoku, such as Anti-knight Sudoku, the modification is much simpler than training another large neural network from scratch to solve the new puzzle. Indeed, one can use the same pre-trained neural network and only need to add a rule in the NeurASP program saying that "no number repeats at a knight move".

- **Learning Shortest Path** We show how expressive reasoning originating from ASP, such as recursive definition, aggregates, and weak constraints, can be used in NeurASP learning. We aim at training a neural network to find a shortest path in a $4 \times 4$ grid with missing edges. Such a neural network can be used to simulate a plain ASP program to solve the same problem using less

inference time. The representation of the shortest path problem in NeurASP would be almost the same as a CLINGO program and as simple as follows.

```
nn(sp(24, g), [true, false]).
% if edge 1 in graph g is selected in the shortest path,
% then there is an edge between node 0 and node 1
sp(0,1) :- sp(1,g,true).
...
sp(X,Y) :- sp(Y,X).

% [nr] No removed edges should be predicted
:- sp(X,g,true), removed(X).

% [p] (aggregates) Prediction must form a simple path, i.e.,
%                  the degree of each node must be either 0 or 2
:- X=0..15, #count{Y: sp(X,Y)} = 1.
:- X=0..15, #count{Y: sp(X,Y)} >= 3.

% [r] (recursive definition) Every 2 nodes in the prediction must be reachable
reachable(X,Y) :- sp(X,Y).
reachable(X,Y) :- reachable(X,Z), sp(Z,Y).
:- sp(X,A), sp(Y,B), not reachable(X,Y).

% [o] (weak constraint) Predicted path should contain least edges
:~ sp(X,g,true). [1, X]
```

In this experiment, we trained the same neural network model $M_{sp}$ as in [26], a 5-layer Multi-Layer Perceptron (MLP), but with 4 different settings: (i) MLP only; (ii) together with NeurASP with the simple-path constraint **(p)** (which is the only constraint used in [26]); [1] (iii) together with NeurASP with simple-path, reachability, and optimization constraints **(p-r-o)**; and (iv) together with NeurASP with all 4 constraints **(p-r-o-nr)**. [2]

Table 1 shows, after 500 epochs of training, the percentage of the predictions on the test data that satisfy each of the constraints **p**, **r**, and **nr**, the path constraint (i.e., **p-r**), the shortest path constraint (i.e., **p-r-o-nr**), and the accuracy w.r.t. the ground truth. As we can see, NeurASP helps to train the same neural network such that it's more likely to satisfy the constraints. Besides, the last column shows that a neural network is not always trained better with more constraints

Table 1: Shortest Path: Accuracy on Test Data: columns denote MLPs trained with different rules; each row represents the percentage of predictions that satisfy the constraints

| Predictions satisfying | MLP Only | MLP (p) | MLP (p-r-o) | MLP (p-r-o-nr) |
|---|---|---|---|---|
| p | 28.3% | 96.6% | **100**% | 30.1% |
| r | 88.5% | **100**% | **100**% | 87.3% |
| nr | 32.9% | 36.3% | 45.7% | **70.5**% |
| p-r | 28.3% | 96.6% | **100**% | 30.1% |
| p-r-o-nr | 23.0% | 33.2% | **45.7**% | 24.2% |
| *label (ground truth)* | 22.4% | 28.9% | **40.1**% | 22.7% |

[1] A path is *simple* if every node in it other than the source and the destination has only 1 incoming and only 1 outgoing edge.
[2] Other combinations are either meaningless (e.g., **o**) or having similar results (e.g. **p-r** is similar to **p**).

## 6 Open Issues and Expected Achievements

The biggest issue we are encountering now is that training with NeurASP implementation still takes much more time than pure neural network training due to the internal use of a symbolic reasoning engine (i.e., CLINGO in our case). With our recent experience on encoding logic directly in neural networks, we expect that the scalability issue will be resolved by embedding ASP (or possibly a fragment of ASP) directly in neural networks. We expect to design a new formalism whose rules can be turned into neural network regularizers. We also expect to implement a prototype for the new formalism and apply it to the domains that NeurASP was applied to so that we can compare and analyze the pros and cons of both approach.

## References

[1] Jacob Andreas, Marcus Rohrbach, Trevor Darrell & Dan Klein (2016): *Learning to Compose Neural Networks for Question Answering*. In: *Proceedings of the 2016 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1545–1554, doi:10.18653/v1/n16-1181.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi & Peter F. Patel-Schneider, editors (2003): *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

[3] Gerhard Brewka, Ilkka Niemelä & Miroslaw Truszczynski (2011): *Answer Set Programming at a Glance*. *Communications of the ACM* 54(12), pp. 92–103, doi:10.1145/2043174.2043195.

[4] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca & Torsten Schaub (2020): *ASP-Core-2 input language format*. *Theory and Practice of Logic Programming* 20(2), pp. 294–309, doi:10.1017/S1471068419000450.

[5] William W Cohen, Fan Yang & Kathryn Rivard Mazaitis (2018): *TensorLog: Deep Learning Meets Probabilistic Databases*. *Journal of Artificial Intelligence Research* 1, pp. 1–15.

[6] Ivan Donadello, Luciano Serafini & Artur D'Avila Garcez (2017): *Logic tensor networks for semantic image interpretation*. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, AAAI Press, pp. 1596–1602, doi:10.24963/ijcai.2017/221.

[7] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick & Ross Girshick (2017): *Clevr: A diagnostic dataset for compositional language and elementary visual reasoning*. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2901–2910, doi:10.1109/CVPR.2017.215.

[8] Seyed Mehran Kazemi & David Poole (2018): *RelNN: A deep neural model for relational learning*. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

[9] Thomas N. Kipf & Max Welling (2017): *Semi-Supervised Classification with Graph Convolutional Networks*. In: *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017*.

[10] Doga Kisa, Guy Van den Broeck, Arthur Choi & Adnan Darwiche (2014): *Probabilistic sentential decision diagrams*. In: *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

[11] Joohyung Lee & Yi Wang (2016): *Weighted Rules under the Stable Model Semantics*. In: *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 145–154.

[12] Joohyung Lee & Yi Wang (2018): *Weight Learning in a Probabilistic Extension of Answer Set Programs*. In: *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 22–31.

[13] Joohyung Lee & Zhun Yang (2017): *LPMLN, Weak Constraints, and P-log*. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1170–1177.

[14] Yuliya Lierler & Marco Maratea (2004): *Cmodels-2: SAT-based answer set solver enhanced to non-tight programs*. In: *Proceedings of International Conference on Logic Programming and NonMonotonic Reasoning*, Springer, pp. 346–350, doi:10.1007/978-3-540-24609-1_32.

[15] Vladimir Lifschitz (2008): *What Is Answer Set Programming?* In: *Proceedings of the AAAI Conference on Artificial Intelligence*, MIT Press, pp. 1594–1597.

[16] Bill Yuchen Lin, Xinyue Chen, Jamin Chen & Xiang Ren (2019): *KagNet: Knowledge-Aware Graph Networks for Commonsense Reasoning*. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2822–2832, doi:10.18653/v1/D19-1282.

[17] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester & Luc De Raedt (2018): *Deepproblog: Neural probabilistic logic programming*. In: *Proceedings of Advances in Neural Information Processing Systems*, pp. 3749–3759.

[18] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum & Jiajun Wu (2019): *The neuro-symbolic concept learner: interpreting scenes, words, and sentences from natural supervision*. In: *Proceedings of International Conference on Learning Representations*.

[19] Rasmus Palm, Ulrich Paquet & Ole Winther (2018): *Recurrent relational networks*. In: *Proceedings of Advances in Neural Information Processing Systems*, pp. 3368–3378.

[20] Judea Pearl (2000): *Causality: models, reasoning and inference*. 29, Cambridge Univ Press.

[21] Raymond Reiter (1980): *A logic for default reasoning*. *Artificial Intelligence* 13, pp. 81–132, doi:10.1016/0004-3702(80)90014-4.

[22] Tim Rocktäschel & Sebastian Riedel (2017): *End-to-end differentiable proving*. In: *Proceedings of Advances in Neural Information Processing Systems*, pp. 3788–3800.

[23] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura & David L. Dill (2019): *Learning a SAT Solver from Single-Bit Supervision*. In: *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

[24] Gustav Šourek, Vojtech Aschenbrenner, Filip Železny & Ondřej Kuželka (2015): *Lifted relational neural networks*. In: *Proceedings of the 2015th International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches-Volume 1583*, CEUR-WS. org, pp. 52–60.

[25] Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai & Yoav Artzi (2019): *A Corpus for Reasoning about Natural Language Grounded in Photographs*. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL)*, pp. 6418–6428, doi:10.18653/v1/p19-1644.

[26] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang & Guy Van den Broeck (2018): *A Semantic Loss Function for Deep Learning with Symbolic Knowledge*. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Available at `http://starai.cs.ucla.edu/papers/XuICML18.pdf`.

[27] Zhun Yang, Adam Ishay & Joohyung Lee (2020): *NeurASP: Embracing Neural Networks into Answer Set Programming*. In: *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1755–1762, doi:10.1017/S1471068419000450.

[28] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba & Joshua B. Tenenbaum (2020): *CLEVRER: Collision Events for Video Representation and Reasoning*. In: *Proceedings of the 8th International Conference on Learning Representations (ICLR)*.