NeurASP: Embracing Neural Networks into Answer Set Programming

Zhun Yang[1] , **Adam Ishay**[1] and **Joohyung Lee**[1 2 *]

[1] Arizona State University, Tempe, AZ, USA
[2] Samsung Research, Seoul, South Korea

{zyang90,aishay,joolee}@asu.edu

Abstract

We present NeurASP, a simple extension of answer set programs by embracing neural networks. By treating the neural network output as the probability distribution over atomic facts in answer set programs, NeurASP provides a simple and effective way to integrate sub-symbolic and symbolic computation. We demonstrate how NeurASP can make use of a pre-trained neural network in symbolic computation and how it can improve the neural network's perception result by applying symbolic reasoning in answer set programming. Also, NeurASP can be used to train a neural network better by training with ASP rules so that a neural network not only learns from implicit correlations from the data but also from the explicit complex semantic constraints expressed by the rules.

1 Introduction

The integration of low-level perception with high-level reasoning is one of the oldest problems in Artificial Intelligence. Today, the 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., [Cohen *et al.*, 2018; Rocktäschel and Riedel, 2017; Donadello *et al.*, 2017; Kazemi and Poole, 2018; Šourek *et al.*, 2015; Palm *et al.*, 2018; Lin *et al.*, 2019]. However, it is still not clear how complex and high-level reasoning, such as default reasoning [Reiter, 1980], ontology reasoning [Baader *et al.*, 2003], and causal reasoning [Pearl, 2000], 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) [Lifschitz, 2008; Brewka *et al.*, 2011], 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.

In this paper, we present a simple extension of answer set programs by embracing neural networks. Following the idea

of DeepProbLog [Manhaeve *et al.*, 2018], 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.

We demonstrate how NeurASP can be useful for some tasks where both perception and reasoning are required. Reasoning can help identify perception mistakes that violate semantic constraints, which in turn can make perception more robust. For 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. Or when it is unclear whether a round object attached to the car is a wheel or a doughnut, the reasoner could conclude that it is more likely to be a wheel by applying commonsense knowledge. In the case of a neural network that recognizes digits in a given Sudoku board, 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.

Another benefit of this hybrid approach is that it 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, when we need to solve some variation of Sudoku, such as Anti-knight or Offset Sudoku, the modification is simpler than training another large neural network from scratch to solve the new puzzle.

NeurASP can also be used to train a neural network together with rules so that a neural network not only learns from implicit correlations from the data but also from explicit complex semantic constraints expressed by ASP rules. The *semantic loss* [Xu *et al.*, 2018] obtained from the reasoning module can be backpropagated into the rule layer and then further into neural networks via neural atoms. This sometimes makes a neural network learn better even with fewer data.

Compared to DeepProbLog, NeurASP supports a rich set

of KR constructs supported by answer set programming that allows for convenient representation of complex knowledge. It utilizes an ASP solver in computation instead of constructing circuits as in DeepProbLog.

The paper is organized as follows. Section 2 introduces the syntax and the semantics of NeurASP. Section 3 illustrates how reasoning in NeurASP can enhance the perception result by considering relations among objects perceived by pre-trained neural networks. Section 4 presents learning in NeurASP where ASP rules work as a semantic regularizer for training neural networks so that neural networks are trained not only from data but also from rules. Section 5 examines related works and Section 6 concludes.

The implementation of NeurASP, as well as codes used for the experiments, is publicly available online at

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

2 NeurASP

We present the syntax and the semantics of NeurASP.

2.1 Syntax

We assume that neural network M allows an arbitrary tensor as input whereas the output is a matrix in $\mathbb{R}^{e \times n}$, where e is the number of random events predicted by the neural network and n is the number of possible outcomes for each random event. Each row of the matrix represents the probability distribution of the outcomes of each event. For example, if M is a neural network for MNIST digit classification, then the input is a tensor representation of a digit image, e is 1, and n is 10. If M is a neural network that outputs a Boolean value for each edge in a graph, then e is the number of edges and n is 2. Given an input tensor $\mathbf{t}$, by $M(\mathbf{t})$, we denote the output matrix of M. The value $M(\mathbf{t})[i,j]$ (where $i \in \{1, \dots, e\}$, $j \in \{1, \dots, n\}$) is the probability of the j-th outcome of the i-th event upon the input $\mathbf{t}$.

In NeurASP, the neural network M above can be represented by a *neural atom* of the form

$$nn(m(e, t), [v_1, \dots, v_n]), \qquad (1)$$

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 data; related to it, there is a mapping $\mathbf{D}$ (implemented by an external Python code) that turns t into an input tensor; (iv) $v_1, \dots, v_n$ 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 $c \in \{m_1(t), \dots, m_e(t)\}$ and $v \in \{v_1, \dots, v_n\}$. The output of the neural network provides the probabilities of the introduced atoms (defined in Section 2.2).

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 $digit_1(d) = 0$, $digit_1(d) = 1, \dots, digit_1(d) = 9$.

Example 2 *Let M_{sp} be another neural network for finding the shortest path in a graph with 24 edges. The input is a tensor encoding the graph and the start/end nodes of the path, and the output is a matrix in $\mathbb{R}^{24 \times 2}$. This neural network can be represented by the neural atom*

$$nn(sp(24, g), [\text{TRUE}, \text{FALSE}]).$$

A NeurASP *program* Π is the union of Π^{asp} and Π^{nn}, where Π^{asp} is a set of propositional rules (standard rules as in ASP-Core 2 [Calimeri *et al.*, 2020]) and Π^{nn} is a set of neural atoms. Let σ^{nn} be the set of all atoms $m_i(t) = v_j$ that is obtained from the neural atoms in Π^{nn} as described above. We require that, in each rule *Head* ← *Body* in Π^{asp}, no atoms in σ^{nn} appear in *Head*.

We could allow schematic variables into Π, which are understood in terms of grounding as in standard answer set programs. We find it convenient to use rules of the form

$$nn(m(e, t), [v_1, \dots, v_n]) \leftarrow Body \qquad (2)$$

where *Body* is either identified by $\top$ or $\bot$ during grounding so that (2) can be viewed as an abbreviation of multiple (variable-free) neural atoms (1).

Example 3 *An example NeurASP program Π_{digit} is as follows, where d_1 and d_2 are terms representing two images. Each image is classified by neural network M_{digit} as one of the values in $\{0, \dots, 9\}$. The addition of two digit-images is the sum of their values.*

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

$$(3)$$

The neural network M_{digit} outputs 10 probabilities for each image. The addition is applied once the digits are recognized and its probability is induced from the perception as we explain in the next section.

2.2 Semantics

For any NeurASP program Π, we first obtain its ASP counterpart Π' where each neural atom (1) is replaced with the set of rules

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

The above rule (in the language of CLINGO) means to choose exactly one atom in between the set braces.[1] We define the *stable models* of Π as the stable models of Π'.

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

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

---

[1] In practice, each atom $m_i(t) = v$ is written as $m(i, t, v)$.

For instance, recall that the output matrix of $M_{digit}(\mathbf{D}(d))$ in Example 3 is in $\mathbb{R}^{1\times 10}$. The probability of atom $digit_1(d)=k$ is $M_{digit}(\mathbf{D}(d))[1, k+1]$.

Given an interpretation I, by $I|_{\sigma^{nn}}$, we denote the projection of I onto σ^{nn}. By $Num(I|_{\sigma^{nn}}, \Pi)$, we denote the number of stable models of Π that agree with $I|_{\sigma^{nn}}$ on σ^{nn}.

The probability of a stable model I of Π 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 Π that agree with $I|_{\sigma^{nn}}$ on σ^{nn}. That is, for any interpretation I,

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

An *observation* is a set of ASP constraints (i.e., rules of the form $\bot \leftarrow Body$). The probability of an observation O is defined as

$$P_{\Pi}(O) = \sum_{I\models O} P_{\Pi}(I)$$

($I \models O$ denotes that I satisfies O).

The probability of the set $\mathbf{O} = \{O_1, \ldots, O_o\}$ of observations is defined as the product of the probability of each O_i:

$$P_{\Pi}(\mathbf{O}) = \prod_{O_i\in\mathbf{O}} P_{\Pi}(O_i).$$

Example 3 Continued The ASP program Π'_{digit}, which is the ASP counterpart of Π_{digit}, is obtained from (3) by replacing the third rule with

$$\{digit_1(d_1)=0;\ \ldots\ ; digit_1(d_1)=9\} = 1.$$
$$\{digit_1(d_2)=0;\ \ldots\ ; digit_1(d_2)=9\} = 1.$$

The following are the stable models of Π_{digit}, i.e., the stable models of Π'_{digit}.

$I_1 = \{digit_1(d_1)=0, digit_1(d_2)=0, addition(d_1, d_2, 0), \ldots\}$,
$I_2 = \{digit_1(d_1)=0, digit_1(d_2)=1, addition(d_1, d_2, 1), \ldots\}$,
$I_3 = \{digit_1(d_1)=1, digit_1(d_2)=0, addition(d_1, d_2, 1), \ldots\}$,
$\ldots$,
$I_{100} = \{digit_1(d_1)=9, digit_1(d_2)=9, addition(d_1, d_2, 18), \ldots\}$.

Their probabilities are as follows:

$P_{\Pi}(I_1) = M_{digit}(\mathbf{D}(d_1))[1,1] \times M_{digit}(\mathbf{D}(d_2))[1,1]$,
$\ldots$,
$P_{\Pi}(I_{100}) = M_{digit}(\mathbf{D}(d_1))[1,10] \times M_{digit}(\mathbf{D}(d_2))[1,10]$.

The probability of $O = \{\leftarrow not\ addition(d_1, d_2, 1)\}$ is

$$P_{\Pi}(O) = P_{\Pi}(I_2) + P_{\Pi}(I_3).$$

3 Inference with NeurASP

We implemented NeurASP by integrating PyTorch [Adam *et al.*, 2017] and CLINGO [Gebser *et al.*, 2011]. PyTorch takes care of neural network processing including data loading and mapping $\mathbf{D}$ that maps pointer terms in neural atoms to input tensors. Computing the probability of a stable model is done by calling CLINGO and post-processing in Python. This section illustrates how this integration can be useful in reasoning about relations among objects recognized by neural networks.

3.1 Commonsense Reasoning about Image

Suppose we have a neural network M_{label} that outputs classes of objects in the bounding boxes that are already detected. The following rule asserts that the neural network M_{label} classifies the bounding box B into one of $\{car, cat, person, truck, other\}$, where B is at location (X_1, Y_1, X_2, Y_2) in image I:

$$nn(label(1, I, B), [car, cat, person, truck, other]) \leftarrow$$
$$box(I, B, X_1, Y_1, X_2, Y_2).$$

Figure 1: Reasoning about relations among perceived objects

(i₁) (i₂)
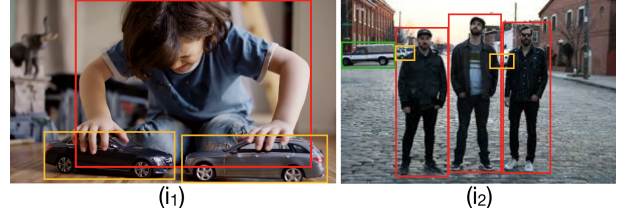
Consider the two images i_1 and i_2 in Figure 1. The bounding boxes can be represented by the following facts.

$$box(i_1, b_1, 100, 0, 450, 350).$$
$$box(i_1, b_2, 300, 300, 500, 400).$$
$$\ldots$$

The first rule says that there is a bounding box b_1 (i.e., the red box with a child) in image i_1, and the coordinates of its left-top and right-bottom corners are $(100, 0)$ and $(450, 350)$.

Below we describe rules that allow for reasoning about the recognized objects. The following rules describe the general size relation between objects.

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

Next is the rule asserting that by default we conclude the same size relationship as above.

$$smaller(I, B_1, B_2) \leftarrow not \sim smaller(I, B_1, B_2),$$
$$label_1(I, B_1)=L_1, label_1(I, B_2)=L_2, smaller(L_1, L_2).$$

(The $\sim$ symbol stands for strong negation in ASP, which asserts explicit falsity.)

On the other hand, there are some exceptions, for instance,

$$\sim smaller(I, B_2, B_1) \leftarrow box(I, B_1, X_1, Y_1, X_2, Y_2),$$
$$box(I, B_2, X_1', Y_1', X_2', Y_2'), Y_2 \geq Y_2',$$
$$|X_1 - X_2| \times |Y_1 - Y_2| < |X_1' - X_2'| \times |Y_1' - Y_2'|.$$

$$smaller(I, B_1, B_2) \leftarrow \sim smaller(I, B_2, B_1).$$

$$toy(I, B_1) \leftarrow label_1(I, B_1) = L_1,\ label_1(I, B_2) = L_2,$$
$$smaller(I, B_1, B_2),\ smaller(L_2, L_1).$$

The first rule says that "B_2 is not smaller than B_1 if (i) B_1 and B_2 are objects in image I, (ii) B_1 is closer to the camera (i.e., B_1's bottom boundary is closer to the bottom of I), and (iii) the box in the image for B_1 is smaller than B_2." [2]

---

[2] We assume that the camera is at the same height as the objects.

The neural network model M_{label} outputs that the red boxes are persons, the yellow boxes are cars, and the green box is a truck. Upon this input and the rules above, NeurASP allows us to derive that the two cars in image i_1 are *toy* cars, whereas the two cars in image i_2 are not: although they are surrounded by smaller boxes than those of humans, their boxes are not closer to the camera.

3.2 Example: Solving Sudoku Puzzle in Image

Consider the task of solving a Sudoku puzzle given as an image. In NeurASP, we could use a neural network to recognize the digits in the given puzzle and use an ASP solver to compute the solution instead of having a single network that accounts for both perception and solving.

We use the following NeurASP program Π_{sudoku} to first identify the digits in each grid cell on the board and then find the solution by assigning digits to all empty grid cells. [3]

```
% identify the number in each of the 81 positions
nn(identify(81, img), [empty,1,2,3,4,5,6,7,8,9]).

% assign one number N to each position (R,C)
a(R,C,N) :- identify(Pos,img,N), R=Pos/9, C=Pos\9,
            N!=empty.
{a(R,C,N): N=1..9}=1 :- identify(Pos, img, empty),
                       R=Pos/9, C=Pos\9.

% no number repeats in the same row
:- a(R,C1,N), a(R,C2,N), C1!=C2.

% no number repeats in the same column
:- a(R1,C,N), a(R2,C,N), R1!=R2.

% no number repeats in the same 3*3 box
:- a(R,C,N), a(R1,C1,N), R!=R1, C!=C1,
   ((R/3)*3 + C/3) = ((R1/3)*3 + C1/3).
```

The neural network model $M_{identify}$ is rather simple. It is composed of 5 convolutional layers with dropout, a max pooling layer, and a 1×1 convolutional layer followed by softmax. Given a Sudoku board image (.png file), neural network $M_{identify}$ outputs a matrix in $\mathbb{R}^{81 \times 10}$, which represents the probabilities of the values (empty, 1, ..., 9) in each of the 81 grid cells. The network $M_{identify}$ is pre-trained using $\langle image, label \rangle$ pairs, where each $image$ is a Sudoku board image generated by *OpenSky Sudoku Generator* (http://www.opensky.ca/~jdhildeb/software/sudokugen/) and each $label$ is a vector of length 81 in which 0 is used to represent an empty cell at that position.

Let $Acc_{identify}$ denote the accuracy of identifying all empty cells and the digits on the board given as an image without making a single mistake in a grid cell. Let Acc_{sol} denote the accuracy of solving a given Sudoku board without making a single mistake in a grid cell. Let r be the following rule in Π_{sudoku}:

```
{a(R,C,N): N=1..9}=1 :- identify(Pos, img, empty),
                       R=Pos/9, C=Pos\9.
```

Table 1 compares $Acc_{identify}$ of each of $M_{identify}$, NeurASP program $\Pi_{sudoku} \setminus r$ with $M_{identify}$, NeurASP program Π_{sudoku} with $M_{identify}$, as well as Acc_{sol} of Π_{sudoku} with $M_{identify}$.

Table 1: Sudoku: Accuracy on Test Data

Num of Train Data	$Acc_{identify}$ of $M_{identify}$	$Acc_{identify}$ of NeurASP w/ $\Pi_{sudoku} \setminus r$	$Acc_{identify}$ of NeurASP w/ Π_{sudoku}	Acc_{sol} of NeurASP w/ Π_{sudoku}
15	15%	49%	71%	71%
17	31%	62%	80%	80%
19	72%	90%	95%	95%
21	85%	95%	98%	98%
23	93%	99%	100%	100%
25	100%	100%	100%	100%

Intuitively, $\Pi_{sudoku} \setminus r$ only checks whether the identified numbers (by neural network $M_{identify}$) satisfy the three constraints (the last three rules of Π_{sudoku}), while Π_{sudoku} further checks whether there exists a solution given the identified numbers. As shown in Table 1, the use of reasoning in NeurASP program $\Pi_{sudoku} \setminus r$ improves the accuracy $Acc_{identify}$ of the neural network $M_{identify}$ as explained in the introduction. The accuracy $Acc_{identify}$ is further improved by trying to solve Sudoku completely using Π_{sudoku}. Note that the solution accuracy Acc_{sol} of Π_{sudoku} is equal to the perception accuracy $Acc_{identify}$ of Π_{sudoku} since the ASP yields a 100% correct solution once the board is correctly identified.

Palm *et al.* [2018] use a Graph Neural Network to solve Sudoku but the work restricts attention to textual input of the Sudoku board, not images as we do. Their work achieves 96.6% accuracy after training with 216,000 examples. In comparison, even with the more challenging task of accepting images as input, the number of training examples we used is $15 - 25$, which is much less than the number of training examples used in [Palm *et al.*, 2018]. Our work takes advantage of the fact that in a problem like Sudoku, where the constraints are explicitly given, a neural network only needs to focus on perception tasks, which is simpler than learning the perception and reasoning together.

Furthermore, using the same trained perception neural network $M_{identify}$, we can solve some elaborations of Sudoku problems by adding the following rules:

[Anti-knight Sudoku] No number repeats at a knight move

```
:- a(R1,C1,N), a(R2,C2,N), |R1−R2|+|C1−C2|=3.
```

[Sudoku-X] No number repeats at the diagonals

```
:- a(R1,C1,N), a(R2,C2,N), R1=C1, R2=C2, R1!=R2.
:- a(R1,C1,N), a(R2,C2,N), R1+C1=8, R2+C2=8, R1!=R2.
```

With neural network only approach, since the neural network needs to learn both perception and reasoning, each of the above variations would require training a complex and different model with a big dataset. However, with NeurASP, the neural network only needs to recognize digits on the board. Thus solving each Sudoku variation above uses the same pre-trained model for the image input and we only need to add the aforementioned rules to Π_{sudoku}.

Some Sudoku variations, such as Offset Sudoku, are in colored images. In this case, we need to increase the number of

---

[3]The expression $\{a(R, C, N) : N = 1..9\} = 1$ is a shorthand for $\{a(R, C, 1); \ldots ; a(R, C, 9)\} = 1$ in the language of CLINGO.

channels of $M_{identify}$ from 1 to 3, and need to retrain the neural network with the colored images. Although not completely elaboration tolerant, compared to the pure neural network approach, this is significantly simpler. For instance, the number of training data needed to get 100% perception accuracy for Offset Sudoku ($Acc_{identify}$) is 70, which is still much smaller than what the end-to-end Sudoku solver would require. Using the new network trained, we only need to add the following rule to Π_{sudoku}.

[Offset Sudoku] No number repeats at the same relative position in 3*3 boxes

```
:- a(R1,C1,N), a(R2,C2,N), R1\3 = R2\3,
   C1\3 = C2\3, R1 != R2, C1 != C2.
```

4 Learning in NeurASP

We show how the semantic constraints expressed in NeurASP can be used to train neural networks better.

4.1 Gradient Ascent with NeurASP

In this section, we denote a NeurASP program by $\Pi(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is the set of the parameters in the neural network models associated with Π. Assume a NeurASP program $\Pi(\boldsymbol{\theta})$ and a set $\mathbf{O}$ of observations such that $P_{\Pi(\boldsymbol{\theta})}(O) > 0$ for each $O \in \mathbf{O}$. The task is to find $\hat{\boldsymbol{\theta}}$ that maximizes the log-likelihood of observations $\mathbf{O}$ under program $\Pi(\boldsymbol{\theta})$, i.e.,

$$\hat{\boldsymbol{\theta}} \in \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ log(P_{\Pi(\boldsymbol{\theta})}(\mathbf{O})),$$

which is equivalent to

$$\hat{\boldsymbol{\theta}} \in \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ \sum_{O \in \mathbf{O}} log(P_{\Pi(\boldsymbol{\theta})}(O)).$$

Let $\mathbf{p}$ denote the probabilities of the atoms in σ^{nn}. Since $\mathbf{p}$ is indeed the outputs of the neural networks in $\Pi(\boldsymbol{\theta})$, we can compute the gradient of $\mathbf{p}$ w.r.t. $\boldsymbol{\theta}$ through backpropagation. Then the gradient of $\sum_{O \in \mathbf{O}} log(P_{\Pi(\boldsymbol{\theta})}(O))$ w.r.t. $\boldsymbol{\theta}$ is

$$\frac{\partial \sum_{O \in \mathbf{O}} log(P_{\Pi(\boldsymbol{\theta})}(O))}{\partial \boldsymbol{\theta}} = \sum_{O \in \mathbf{O}} \frac{\partial log(P_{\Pi(\boldsymbol{\theta})}(O))}{\partial \mathbf{p}} \times \frac{\partial \mathbf{p}}{\partial \boldsymbol{\theta}}$$
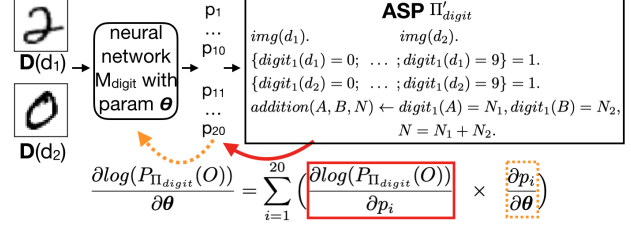
where $\frac{\partial \mathbf{p}}{\partial \boldsymbol{\theta}}$ can be computed through the usual neural network backpropagation, while $\frac{\partial log(P_{\Pi(\boldsymbol{\theta})}(O))}{\partial p}$ for each $p \in \mathbf{p}$ can be computed as follows.

Proposition 1 *Let $\Pi(\boldsymbol{\theta})$ be a NeurASP program and let O be an observation such that $P_{\Pi(\boldsymbol{\theta})}(O) > 0$. Let p denote the probability of an atom $c = v$ in σ^{nn}, i.e., p denotes $P_{\Pi(\boldsymbol{\theta})}(c = v)$. We have that[4]*

$$\frac{\partial log(P_{\Pi(\boldsymbol{\theta})}(O))}{\partial p} = \frac{\displaystyle\sum_{\substack{I: \ I \models O \\ I \models c = v}} \frac{P_{\Pi(\boldsymbol{\theta})}(I)}{P_{\Pi(\boldsymbol{\theta})}(c=v)} - \sum_{\substack{I, v': \ I \models O \\ I \models c = v', v \neq v'}} \frac{P_{\Pi(\boldsymbol{\theta})}(I)}{P_{\Pi(\boldsymbol{\theta})}(c=v')}}{\displaystyle\sum_{I: \ I \models O} P_{\Pi(\boldsymbol{\theta})}(I)}$$

---

[4] $\frac{P_{\Pi(\boldsymbol{\theta})}(I)}{P_{\Pi(\boldsymbol{\theta})}(c=v)}$ and $\frac{P_{\Pi(\boldsymbol{\theta})}(I)}{P_{\Pi(\boldsymbol{\theta})}(c=v')}$ are still well-defined since the denominators have common factors in $P_{\Pi(\boldsymbol{\theta})}(I)$.

Figure 2: NeurASP Gradient Propagation

$$\frac{\partial log(P_{\Pi_{digit}}(O))}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{20} \left(\frac{\partial log(P_{\Pi_{digit}}(O))}{\partial p_i} \times \frac{\partial p_i}{\partial \boldsymbol{\theta}} \right)$$

Intuitively, the proposition tells us that each interpretation I that satisfies O tends to increase the value of p if $I \models c = v$, and decrease the value of p if $I \models c = v'$ such that $v' \neq v$. NeurASP internally calls CLINGO to find all stable models I of $\Pi(\boldsymbol{\theta})$ that satisfy O and uses PyTorch to obtain the probability of each atom $c = v$ in σ^{nn}.

4.2 Experiment 1: Learning Digit Classification from Addition

All experiments in Section 4 were done on Ubuntu 18.04.2 LTS with two 10-cores CPU Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz and four GP104 [GeForce GTX 1080].

The digit addition problem is a simple example used in [Manhaeve *et al.*, 2018] 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 problem can be represented by NeurASP program Π_{digit} in Example 3. For comparison, we use the same dataset and the same structure of the neural network model used in [Manhaeve *et al.*, 2018] to train the digit classifier M_{digit} in Π_{digit}. For each pair of images denoted by d_1 and d_2 and their sum n, we construct the ASP constraint $\leftarrow not \ addition(d_1, d_2, n)$ as the observation O. The training target is to maximize $log(P_{\Pi_{digit}}(O))$.

Figure 2 shows how the forward and the backward propagations are done for NeurASP program Π_{digit} in Example 3. The left-to-right direction is the forward computation of the neural network extended with the rule layer, whose output is the probability of the observation O. The right-to-left direction shows how the gradient from the rule layer is backpropagated further into the neural network by the chain rule to update all neural network parameters so as to find the parameter values that maximize the probability of the given observation.

Figure 3 shows the accuracy on the test data after each training iteration. The method CNN denotes the baseline used in [Manhaeve *et al.*, 2018] 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 (Sequential Decision Diagram) at each iteration for each training instance (i.e., each pair of images). This example

Figure 3: NeurASP v.s. DeepProbLog
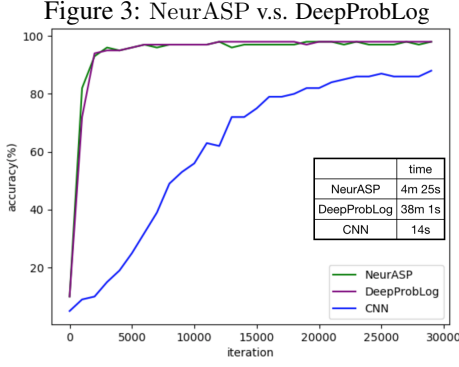
illustrates that generating many SDDs could be more time-consuming than enumerating stable models in NeurASP computation. In general, there is a trade-off between the two methods and other examples may show the opposite behavior.

4.3 Experiment 2: Learning How to Solve Sudoku

In section 3.2, we used a neural network $M_{identify}$ to identify the numbers on a Sudoku board and used ASP rules to solve the Sudoku problem. In this section, we use a neural network to learn to solve Sudoku problems. The task is, given the *textual representation* of an unsolved Sudoku board (in the form of a 9×9 matrix where an empty cell is represented by 0), to let a neural network learn to predict the solution of the Sudoku board.

We use the neural network M_{sol} from [Park, 2018] as the baseline. M_{sol} is composed of 9 convolutional layers and a 1x1 convolution layer followed by softmax. Park trained M_{sol} using 1 million examples and achieved 70% accuracy using an "inference trick": instead of predicting digits for all empty cells at once, which leads to a poor accuracy, the most probable grid-cell value was predicted one by one.

Since the current NeurASP implementation is not as scalable as neural network training, training on 1 million examples takes too long. Thus, we construct a dataset of 63,000 + 1000 $\langle config, label \rangle$ pairs for training and testing. Using Park's method on this relatively small dataset, we observe that M_{sol}'s highest whole-board accuracy Acc_{sol}[5] is only 29.1% and M_{sol}'s highest grid-cell accuracy[6] is only 89.3% after 63 epochs of training.

We get a better result by training M_{sol} with the NeurASP program Π_{sol}. The program is almost the same as $\Pi_{identify}$ in Section 3.2 except that it uses M_{sol} in place of $M_{identify}$ and the first three rules of $\Pi_{identify}$ are replaced with

```
nn(sol(81, img), [1,2,3,4,5,6,7,8,9]).
a(R,C,N) :- identify(sol, img, N), R=Pos/9, C=Pos\9.
```

because we do not have to assign the value `empty` in solving Sudoku.

We trained M_{sol} using NeurASP where the training target is to maximize the probability of all stable models that satisfy the observation. On the same test data, after 63 epochs

of training, the highest whole-board accuracy of M_{sol} trained this way is 66.5% and the highest grid-cell accuracy is 96.9% (In other words, we use rules only during training and not during testing). This indicates that including such structured knowledge sometimes helps the training of the neural network significantly.

4.4 Experiment 3: Learning Shortest Path (SP)

The experiment is about, given a graph and two points, finding the shortest path between them. We use the dataset from [Xu *et al.*, 2018], which was used to demonstrate the effectiveness of semantic constraints for enhanced neural network learning. Each example is a 4 by 4 grid $G = (V, E)$, where $|V| = 16, |E| = 24$. The source and the destination nodes are randomly picked up, as well as 8 edges are randomly removed to increase the difficulty. The dataset is divided into 60/20/20 train/validation/test examples.

The following NeurASP program [7]

```
nn(sp(24, g), [true, false]).
sp(0,1) :- sp(1,g,true).
...
sp(X,Y) :- sp(Y,X).
```

together with the union of the following 4 constraints defines the shortest path.

```
% [nr] 1. No removed edges should be predicted
:- sp(X,g,true), removed(X).

% [p] 2. 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] 3. 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] 4. 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 [Xu *et al.*, 2018], 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 [Xu *et al.*, 2018]); [8] (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**). [9]

Table 2 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 short-

---

[5] The percentage of Sudoku examples that are correctly solved.

[6] The percentage of grid cells having correct digits regardless whether the Sudoku solution is correct.

[7] $sp(X, g, true)$ means edge X is in the shortest path. $sp(X, Y)$ means there is a path between nodes X and Y in the shortest path.

[8] A path is *simple* if every node in the path other than the source and the destination has only 1 incoming edge and only 1 outgoing edge.

[9] Other combinations are either meaningless (e.g., **o**) or having similar results (e.g. **p-r** is similar to **p**).

est path constraint (i.e., **p-r-o-nr**), and the accuracy w.r.t. the ground truth.

The accuracies for the first experiment (MLP Only) show that M_{sp} was not trained well only by minimizing the cross-entropy loss of its prediction: 100-28.3 = 71.7% of the predictions are not even a simple-path.

In the remaining experiments (MLP (x)), instead of minimizing the cross-entropy loss, our training target is changed to maximizing the probability of all stable models under certain constraints. The accuracies under the 2nd and 3rd experiments (MLP (p) and MLP (p-r-o) columns) are increased significantly, showing that (i) including such structured knowledge helps the training of the neural network and (ii) the more structured knowledge included, the better M_{sp} is trained under NeurASP. Compared to the results from [Xu *et al.*, 2018], M_{sp} trained by NeurASP with the simple-path constraint **p** (in the 2nd experiment MLP (p) column) obtains a similar accuracy on predicting the label (28.9% v.s. 28.5%) but a higher accuracy on predicting a simple-path (96.6% v.s. 69.9%).

In the 4th experiment (MLP (p-r-o-nr) column) where we added the constraint **nr** saying that "no removed edges can be predicted", the accuracies go down. This is because the new constraint **nr** is about randomly removed edges, changing from one example to another, which is hard to be generalized.

Table 2: 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%

5 Related Work

Recent years have observed the rising interests of combining perception and reasoning. As mentioned, the work on DeepProbLog [Manhaeve *et al.*, 2018] is closest to our work. Some differences are: (i) The computation of DeepProbLog relies on constructing circuits such as sequential decision diagrams (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.

Also related is using the semantic constraints to train neural networks better [Xu *et al.*, 2018], but the constraints used in that work are simple propositional formulas whereas we use answer set programming language, in which it is more convenient to encode complex KR constraints. Logic Tensor Network [Donadello *et al.*, 2017] 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) [Rocktäschel and Riedel, 2017] adopts the idea of dynamic neural module networks [Andreas *et al.*, 2016] 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 [Cohen *et al.*, 2018], 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) [Kipf and Welling, 2017] 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 [Palm *et al.*, 2018], it is shown that GNN can do well on Sudoku, but the input there is not an image but a textual representation. However, this is still restrictive compared to the more complex reasoning that KR formalisms provide.

Neuro-Symbolic Concept Learner [Mao *et al.*, 2019] 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 results are similar in the sense that using symbolic reasoning, we could use fewer data to achieve a high accuracy.

NeurASP is similar to LP$^{\text{MLN}}$ [Lee and Wang, 2016] in the sense that they are both probabilistic extensions of ASP and their semantics are defined by translations into ASP [Lee and Yang, 2017]. LP$^{\text{MLN}}$ allows any rules to be weighted, whereas NeurASP uses standard ASP rules.

6 Conclusion

We showed that NeurASP can improve the neural network's perception result by applying reasoning over perceived objects and also can help neural network learn better by compensating the small size data with knowledge and constraints. Since NeurASP is a simple integration of ASP with neural networks, it retains each of ASP and neural networks in individual forms, and can directly utilize the advances in each of them.

The current implementation is a prototype and not highly scalable due to a naive computation of enumerating stable models. The future work includes how to make learning faster, and also analyzing the effects of the semantic constraints more systematically.

Acknowledgments

References

[Adam *et al.*, 2017] Paszke Adam, Gross Sam, Chintala Soumith, Chanan Gregory, Yang Edward, D Zachary, Lin Zeming, Desmaison Alban, Antiga Luca, and Lerer Adam. Automatic differentiation in PyTorch. In *Proceedings of Neural Information Processing Systems*, 2017.

[Andreas *et al.*, 2016] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 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*, pages 1545–1554, 2016.

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

[Brewka *et al.*, 2011] Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

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

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

[Donadello *et al.*, 2017] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1596–1602. AAAI Press, 2017.

[Gebser *et al.*, 2011] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.

[Kazemi and Poole, 2018] Seyed Mehran Kazemi and David Poole. Relnn: A deep neural model for relational learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017*, 2017.

[Lee and Wang, 2016] Joohyung Lee and Yi Wang. Weighted rules under the stable model semantics. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 145–154, 2016.

[Lee and Yang, 2017] Joohyung Lee and Zhun Yang. LPMLN, weak constraints, and P-log. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1170–1177, 2017.

[Lifschitz, 2008] Vladimir Lifschitz. What is answer set programming? In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1594–1597. MIT Press, 2008.

[Lin *et al.*, 2019] Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. 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)*, pages 2822–2832, 2019.

[Manhaeve *et al.*, 2018] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3749–3759, 2018.

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

[Palm *et al.*, 2018] Rasmus Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3368–3378, 2018.

[Park, 2018] Kyubyong Park. Can convolutional neural networks crack sudoku puzzles? https://github.com/Kyubyong/sudoku, 2018.

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

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Rocktäschel and Riedel, 2017] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3788–3800, 2017.

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

[Xu *et al.*, 2018] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, July 2018.