NeurASP = Neural Network + ASP
Introduction: Neuro-Symbolic AI

- Neuro-symbolic AI combines the complementary strengths of neural networks and symbolic AI.

**Neural Network**
- raw and big data
- scalability
- modeling capability
- robustness against faults

**Symbolic AI**
- domain knowledge
- data efficiency
- interpretability
- provable correctness
Some neuro-symbolic approaches

• We have some knowledge about the domain that we want to incorporate to our neural network based learning and reasoning framework. How do we do it?
  • Distillation based approach
  • Compile into the loss function
  • **Have a separate symbolic reasoning and filtering layer**
• Neural networks that generate code (or a collection of structured facts) which then gets implemented by a program lead to another kind of neuro-symbolic integration.
• Neural models where each neuron has a symbolic meaning and the overall network can learn various aspects from noisy data
  • capture logical contradiction
  • learn explainable rules
• Neuro-symbolic approaches in planning and acting domains:
  • partially neural reinforcement learning (RL) framework for the continuous state and action space domain;
  • neuro-symbolic architecture that learns state transitions from images
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 make use of ASP rules to train a neural network better 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.

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. 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
Answer Set Programs

- Answer Set Programming (ASP) is a logic programming paradigm.
  - combinatorial search
  - knowledge intensive tasks
- ASP has well-developed foundations, efficient reasoning systems, and a methodology of use tested on a number of industrial applications.
- ASP supports a rich set of KR constructs that allow for convenient representation of complex knowledge.
  - Aggregates
  - Choices rules
  - Default
  - Optimization rules
Example Answer Set Program

ASP Program II for Digit Addition

\[
\begin{align*}
\{ \text{digit}(d_1)=0 \ ; \ldots \ ; \text{digit}(d_1)=9 \} &= 1. \\
\{ \text{digit}(d_2)=0 \ ; \ldots \ ; \text{digit}(d_2)=9 \} &= 1. \\
\text{addition}(A, B, N) &\leftarrow \text{digit}(A)=N_1, \\
&\quad \text{digit}(B)=N_2, \\
&\quad N = N_1+N_2.
\end{align*}
\]

Example Stable Model

\[ M = \{ \text{digit}(d_1)=3, \text{digit}(d_2)=0, \text{addition}(d_1,d_2,3) \} \]
Examples of ASP programs

Graph coloring

An \( n \)-coloring of a graph \( G = (V, E) \) is a function \( \text{color} : V \rightarrow \{1, \ldots, n\} \) such that \( \text{color}(x) \neq \text{color}(y) \) for every pair of adjacent vertices \( (x, y) \in E \). We would like to use ASP to find an \( n \)-coloring of a given graph (or determine that it does not exist).

This can be accomplished using the following Lparse program:

```
1  c(1..n).
2  l {color(X,I) : c(I)} l :- v(X).
3  :- color(X,I), color(Y,I), e(X,Y), c(I).
```

Line 1 defines the numbers \( 1, \ldots, n \) to be colors. According to the choice rule in Line 2, a unique color \( i \) should be assigned to each vertex \( x \). The constraint in Line 3 prohibits assigning the same color to vertices \( x \) and \( y \) if there is an edge connecting them.

If we combine this file with a definition of \( G \), such as

```
v(1..100). % 1,...,100 are vertices
e(1,55). % there is an edge from 1 to 55
```

and run smodels on it, with the numeric value of \( n \) specified on the command line, then the atoms of the form \( \text{color}(\ldots,\ldots) \) in the output of smodels will represent an \( n \)-coloring of \( G \).
NeurASP = NN + ASP, but why?

• NeurASP allows one to train a NN under weak supervision.

• Perception and reasoning are separated so as to achieve higher accuracy with less data.

• Reasoning can help identify perception mistakes that violate semantic constraints, which in turn can make perception more robust.

• NeurASP is more elaboration tolerant on visual tasks.

• NeurASP extends classification to context relational classification.

• A neural network can be trained together with rules so that it not only learns from implicit correlations from the data but also learns from explicit complex semantic constraints expressed by ASP rules.
How does NeurASP work?
Neural Atom

Link neural network outputs to atoms in an ASP program.

We use neural network digit to classify the image \( d_1 \), and the output \( p_{1,i} \) for \( i \) in \( \{0, \ldots, 9\} \) defines the probability of \( \text{digit}(d_1) = i \).

neural atom: \( \text{nn}( \text{digit}(1,d_1) , [0,1,2,3,4,5,6,7,8,9] ) \)
Syntax of NeurASP

In NeurASP, a neural network \( M \) is represented by a *neural atom* of the form

\[
\text{nn}(m(e, t), [v_1, \ldots, v_n])
\]

where

- **nn** is a reserved keyword to denote a neural atom;
- **m** is an identifier (symbolic name) of the neural network \( M \);
- **t** is a list of terms that serves as a “pointer” to an input tensor to \( M \);
- **\( v_1, \ldots, v_n \)** represent all \( n \) possible outcomes of each of the \( e \) random events.

A *NeurASP program* consists of ASP rules and neural atoms.

**EX.**

Let \( M_{\text{digit}} \) be a neural network that classifies an MNIST digit image. This neural network can be represented by

\[
\text{nn}(\text{digit}(1,d), [0,1,2,3,4,5,6,7,8,9])
\]
NeurASP Inference

Infer the probabilities of stable models and formulas.

Suppose $M_{i,j} = \{\text{digit}(d_1)=i, \text{digit}(d_2)=j, \text{addition}(d_1,d_2,i+j)\}$

- $P_n(M_{i,j}) = p_{1,i} \times p_{2,j}$ (for $i, j \in \{0, \ldots, 9\}$)

- $P_n(\text{addition}(d_1,d_2,3)) = P_n(M_{0,3}) + P_n(M_{1,2}) + P_n(M_{2,1}) + P_n(M_{3,0})$
Semantics of NeurASP — Stable Model

The semantics of NeurASP defines a **stable model** and its **probability** orienting from the NN outputs. For each NeurASP program $\Pi$, we obtain its ASP counterpart $\Pi'$ by replacing each neural atom

$$nn(m(e, t), [v_1, ..., v_n])$$

with the set of choice rules

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

• The **stable models** of $\Pi$ are the stable models of $\Pi'$.

**EX.**  

Let $M_{\text{digit}}$ be a neural network that classifies an MNIST digit image. Consider the NeurASP program $\Pi$ below.

$$nn( \text{digit}(1,d) , \{0,1,2,3,4,5,6,7,8,9\} )$$

Its ASP counterpart $\Pi'$ is

$$\{ \text{digit1}(d)=0 ; ... ; \text{digit1}(d)=9 \} = 1$$

Thus the stable models of $\Pi$ are the 10 stable models of $\Pi'$ below.

$$\{\text{digit1}(d)=0\} \quad \{\text{digit1}(d)=1\} \quad ... \quad \{\text{digit1}(d)=9\}$$
Semantics of NeurASP — Probability

Recall that we turn each neural atom \( nn(m(e, t), [v_1, ..., v_n]) \) into the set of choice rules
\[
\{ m_i(t) = v_1; \ldots; m_i(t) = v_n \} = 1 \quad \text{for} \ i \in \{1, \ldots, e\}.
\]

• Let \( \sigma^{nn} \) denote the set of atoms \( m_i(t) = v_j \) obtained from neural atoms as described above.

• With the input tensor (identified by) \( t \), we assume neural network \( M \) (identified by \( m \)) outputs a matrix in \( \mathbb{R}^{e \times n} \). The \( n \) numbers in the \( i \)-th row define the probability distribution of the following \( n \) atoms.
\[
m_i(t) = v_1, m_i(t) = v_2, \ldots, m_i(t) = v_n
\]

• The probability of a stable model \( I \) of \( \Pi \) is the product of the probabilities of all atoms in \( I \cap \sigma^{nn} \).

EX.

Consider the NeurASP program \( \Pi \) below.
\[
nn(digit(1,d), [0,1,2,3,4,5,6,7,8,9])
\]
Suppose the output of \( M \) given input tensor \( d \) is \( [0, 0, 0.3, 0, 0, 0, 0.6, 0, 0.1] \), then
the probability of \( I_0 = \{ digit(d) = 0 \} \) is \( P_n(I_0) = 0 \)

...the probability of \( I_5 = \{ digit(d) = 9 \} \) is \( P_n(I_5) = 0.1 \)
NeurASP Learning

Back-propagate gradients to the neural network through the chain rule.

ASP Program II for Digit Addition

\[
\text{addition}(A, B, N) \leftarrow \text{digit}(A) = N_1, \text{digit}(B) = N_2, N = N_1 + N_2.
\]

\[
\frac{\partial P_{\Pi}(\text{addition}(d_1, d_2, 3))}{\partial \theta} = \sum_{i \in \{1, 2\}} \sum_{j \in \{0, \ldots, 9\}} \frac{\partial P_{\Pi}(\text{addition}(d_1, d_2, 3))}{\partial p_{i,j}} \times \frac{\partial p_{i,j}}{\partial \theta}
\]
Gradients Computation

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

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

1. find all stable models $I$ of $\Pi \cup O$
2. compute the probability $P_{\Pi}(I)$
3. use Prop 1 to compute $\frac{\partial \log(P_{\Pi(\theta)}(O))}{\partial p}$
4. use the chain rule to further backward to NN parameters

$$
\frac{\partial}{\partial \theta} \sum_{O \in O} \log(P_{\Pi(\theta)}(O)) = \sum_{O \in O} \frac{\partial \log(P_{\Pi(\theta)}(O))}{\partial p} \times \frac{\partial p}{\partial \theta}
$$
Example NeurASP Program: Sudoku

Task: given an image of Sudoku board, predict the solution.

• Use NN identify to identify the digits in each of the 81 grid cells.

  \[ \text{nn}(\text{identify}(81, \text{img}), [\text{empty}, 1, 2, 3, 4, 5, 6, 7, 8, 9]). \]

• Assign one number to each cell \( i \) for \( i \in \{1, \ldots, 81\} \).

  \[ \text{a}(R, C, N) \leftarrow \text{identify}_i(\text{img})=N, R=i/9, C=i\mod 9, \text{N}\neq \text{empty}. \]
  \[ \{\text{a}(R, C, 1); \ldots; \text{a}(R, C, 9)\}=1 \leftarrow \text{identify}_i(\text{img})=\text{empty}, R=i/9, C=i\mod 9. \]

• No number repeats in the same row, column, and 3\( \times \)3 box.

  \[ \leftarrow \text{a}(R, C_1, N), \text{a}(R, C_2, N), C_1\neq C_2. \]
  \[ \leftarrow \text{a}(R_1, C, N), \text{a}(R_2, C, N), R_1\neq R_2. \]
  \[ \leftarrow \text{a}(R_1, C_1, N), \text{a}(R_2, C_2, N), R_1\neq R_2, C_1\neq C_2, ((R_1/3)\times 3+C_1/3) = ((R_2/3)\times 3+C_2/3). \]
What’s the benefit of NeurASP?
Higher Accuracy with Less Data

1. Perception and reasoning are separated so as to achieve higher accuracy with less data. (Neuro-symbolic Concept Learner by Mao et al. 2019)

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>Number of Data for Training</th>
<th>Accuracy of Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>NeurASP</td>
<td>Convolutional Neural Network + ASP</td>
<td>Image of Sudoku</td>
<td>25</td>
</tr>
<tr>
<td>(Park 2018)</td>
<td>Convolutional Neural Network</td>
<td>Text Representation of Sudoku (9×9 numbers)</td>
<td>1 Million</td>
</tr>
<tr>
<td>(Palm et al. 2018)</td>
<td>Graph Neural Network</td>
<td>Text Representation of Sudoku (9×9 numbers)</td>
<td>180,000</td>
</tr>
</tbody>
</table>
Prune out Perception Errors

2. Reasoning can help identify perception mistakes that violate semantic constraints, which in turn can make perception more robust.
Prune out Perception Errors

2. Reasoning can help identify perception mistakes that violate semantic constraints, which in turn can make perception more robust.

<table>
<thead>
<tr>
<th>Num of Train Data</th>
<th>Acc_identify of ( M_{\text{identify}} )</th>
<th>Acc_identify of NeurASP w/ ( \Pi_{\text{sudoku}} \setminus r )</th>
<th>Acc_identify of NeurASP w/ ( \Pi_{\text{sudoku}} )</th>
<th>Acc_sol of NeurASP w/ ( \Pi_{\text{sudoku}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>15%</td>
<td>49%</td>
<td>71%</td>
<td>71%</td>
</tr>
<tr>
<td>17</td>
<td>31%</td>
<td>62%</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>19</td>
<td>72%</td>
<td>90%</td>
<td>95%</td>
<td>95%</td>
</tr>
<tr>
<td>21</td>
<td>85%</td>
<td>95%</td>
<td>98%</td>
<td>98%</td>
</tr>
<tr>
<td>23</td>
<td>93%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>25</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Intuitively, \( \Pi_{\text{sudoku}} \setminus r \) only checks whether the identified numbers (by neural network \( M_{\text{identify}} \)) satisfy the three constraints (the last three rules of \( \Pi_{\text{sudoku}} \)), while \( \Pi_{\text{sudoku}} \) further checks whether there exists a solution given the identified numbers.

\[
\{a(R,C,1); \ldots; a(R,C,9)\}=1 \leftarrow \text{identify}_{i}(\text{img})=\text{empty}, \ R=i/9, \ C=i \setminus 9.
\]
3. NeurASP can be easily applied to elaborations of a task.

**Offset Sudoku**

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

\[- a(R_1, C_1, N), a(R_2, C_2, N), R_1 \backslash 3 = R_2 \backslash 3, \]
\[- C_1 \backslash 3 = C_2 \backslash 3, R_1 \neq R_2, C_1 \neq C_2. \]

**Anti-knight Sudoku** No number repeats at a knight move

\[- a(R_1, C_1, N), a(R_2, C_2, N), |R_1-R_2|+|C_1-C_2|=3. \]

**Sudoku-X** No number repeats at the diagonals

\[- a(R_1, C_1, N), a(R_2, C_2, N), R_1=C_1, R_2=C_2, R_1!\neq R_2. \]
\[- a(R_1, C_1, N), a(R_2, C_2, N), R_1+C_1=8, R_2+C_2=8, R_1!=R_2. \]
Weak Supervision

4. NeurASP allows one to train a NN under weak supervision.

<table>
<thead>
<tr>
<th></th>
<th>add2x2</th>
<th>apply2x2</th>
<th>member(3)</th>
<th>member(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy(%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepProbLog</td>
<td>88.4±0.7</td>
<td>100±0</td>
<td>96.3±0.3</td>
<td>timeout</td>
</tr>
<tr>
<td>NeuroLog</td>
<td>97.5±0.4</td>
<td>100±0</td>
<td>94.5±1.5</td>
<td>93.9±1.5</td>
</tr>
<tr>
<td>NeurASP</td>
<td>97.6±0.2</td>
<td>100±0</td>
<td>93.5±0.9</td>
<td>timeout</td>
</tr>
<tr>
<td>CL-STE</td>
<td>98.0±0.2</td>
<td>100±0</td>
<td>95.5±0.7</td>
<td>95.0±0.5</td>
</tr>
<tr>
<td>time(s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepProbLog</td>
<td>1035±71</td>
<td>586±9</td>
<td>2218±211</td>
<td>timeout</td>
</tr>
<tr>
<td>NeuroLog</td>
<td>2400±46</td>
<td>2428±29</td>
<td>427±12</td>
<td>682±40</td>
</tr>
<tr>
<td>NeurASP</td>
<td>142±2</td>
<td>47±1</td>
<td>253±1</td>
<td>timeout</td>
</tr>
<tr>
<td>CL-STE</td>
<td>54±4</td>
<td>112±2</td>
<td>43±3</td>
<td>49±4</td>
</tr>
</tbody>
</table>
Weak Supervision — Example Programs

nn(digit(4, i), [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).
add2x2(R1, R2, C1, C2) :- digit(0, i, N1), digit(1, i, N2), digit(2, i, N3), digit(3, i, N4),
                     R1 = N1+N2, R2 = N3+N4, C1 = N1+N3, C2 = N2+N4.

nn(op(4, i), [0, 1, 2]).
aply(D1, 01, D2, 02, D3, R) :- digits(D1, D2, D3), O1 = 0, O2 = 0, R = (D1*D2)+D3.
aply(D1, 01, D2, 02, D3, R) :- digits(D1, D2, D3), O1 = 0, O2 = 1, R = (D1*D2)-D3.
aply(D1, 01, D2, 02, D3, R) :- digits(D1, D2, D3), O1 = 0, O2 = 2, R = (D1*D2)*D3.
aply(D1, 01, D2, 02, D3, R) :- digits(D1, D2, D3), O1 = 1, O2 = 0, R = (D1-D2)+D3.
aply(D1, 01, D2, 02, D3, R) :- digits(D1, D2, D3), O1 = 1, O2 = 2, R = (D1-D2)*D3.
aply(D1, 01, D2, 02, D3, R) :- digits(D1, D2, D3), O1 = 2, O2 = 0, R = (D1*D2)+D3.
aply(D1, 01, D2, 02, D3, R) :- digits(D1, D2, D3), O1 = 2, O2 = 2, R = (D1*D2)*D3.
aply2x2(D1, D2, D3, R1, R2, C1, C2) :- digits(D1, D2, D3),
                        op(0, i, O1), op(1, i, O2), op(2, i, O3), op(3, i, O4),
aply(D1, 01, D2, 02, D3, R1),
aply(D1, 01, D2, 02, D3, R2),
aply(D1, 01, D2, 03, D3, C1),
aply(D1, 02, D2, 04, D3, C2).
Context Relational Classification

5. NeurASP extends classification to context relational classification

Q: What are the cars and toy-cars in these images?

• By default, we believe person is smaller than car.
  \[ \text{smaller}(B, B') \leftarrow \text{label}(B) = \text{person}, \text{label}(B') = \text{car}, \text{not } \sim \text{smaller}(B, B'). \]

• On the other hand, there are some exceptions.
  \[ \sim \text{smaller}(B, B') \leftarrow \text{box}(B, X_1, Y_1, X_2, Y_2), \text{box}(B', X_1', Y_1', X_2', Y_2'), \]
  \[ Y_2 \leq Y_2', \quad |X_1 - X_2| \times |Y_1 - Y_2| > |X_1' - X_2'| \times |Y_1' - Y_2'|. \]
  \[ \text{toy}(B') \leftarrow \text{label}(B) = \text{person}, \text{label}(B') = \text{car}, \text{smaller}(B', B). \]
Semantic Regularizer

6. A neural network can be trained together with rules so that it not only learns from implicit correlations from the data but also learns from explicit complex semantic constraints expressed by ASP rules.

Multi-Layer Perception (Cross-Entropy)

Multi-Layer Perception (NeurASP)
Semantic Regularizer

6. A neural network can be trained together with rules so that it not only learns from implicit correlations from the data but also learns from explicit complex semantic constraints expressed by ASP rules.

% [nr] 1. No removed edges should be predicted
\[ :\text{sp}(X,g,\text{true}), \text{removed}(X). \]

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

% [r] 3. Every 2 nodes in the prediction must be % reachable
\[ \text{reachable}(X,Y) :- \text{sp}(X,Y). \]
\[ \text{reachable}(X,Y) :- \text{reachable}(X,Z), \text{sp}(Z,Y). \]
\[ :- \text{sp}(X,A), \text{sp}(Y,B), \text{not reachable}(X,Y). \]

% [o] 4. Predicted path should contain least edges
\[ :\text{sp}(X,g,\text{true}). [1, X] \]

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

<table>
<thead>
<tr>
<th>Predictions satisfying</th>
<th>MLP Only</th>
<th>MLP (p)</th>
<th>MLP (p-r-o)</th>
<th>MLP (p-r-o-nr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>28.3%</td>
<td>96.6%</td>
<td>100%</td>
<td>30.1%</td>
</tr>
<tr>
<td>r</td>
<td>88.5%</td>
<td>100%</td>
<td>100%</td>
<td>87.3%</td>
</tr>
<tr>
<td>nr</td>
<td>32.9%</td>
<td>36.3%</td>
<td>45.7%</td>
<td>70.5%</td>
</tr>
<tr>
<td>p-r</td>
<td>28.3%</td>
<td>96.6%</td>
<td>100%</td>
<td>30.1%</td>
</tr>
<tr>
<td>p-r-o-nr</td>
<td>23.0%</td>
<td>33.2%</td>
<td>45.7%</td>
<td>24.2%</td>
</tr>
</tbody>
</table>

*label (ground truth)*

<table>
<thead>
<tr>
<th>Predictions satisfying</th>
<th>MLP Only</th>
<th>MLP (p)</th>
<th>MLP (p-r-o)</th>
<th>MLP (p-r-o-nr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>label (ground truth)</td>
<td>22.4%</td>
<td>28.9%</td>
<td>40.1%</td>
<td>22.7%</td>
</tr>
</tbody>
</table>
Future work & codes

• Training with NeurASP takes much more time than pure NN training due to exact inference — NeurASP uses clingo to ground the whole program and enumerate all stable models.

• We will improve the scalability of NeurASP using approximate inference instead of exact inference in our future work.

• The codes for NeurASP and all experiments are available at
  • https://github.com/azreasoners/NeurASP
Follow-up works by others

Neural-Symbolic Integration: A Compositional Perspective*

Efthymia Tsamoura¹, Timothy Hospedales¹, Loizos Michael²
¹ Samsung AI Research
² Open University of Cyprus &
Research Center on Interactive Media,
Smart Systems, and Emerging Technologies
efi.tsamoura@samsung.com, thospedales@samsung.com, loizos@ouc.ac.cy

Abstract
Despite significant progress in the development of neural-symbolic frameworks, the question of how to integrate a neural and a symbolic system in a compositional manner remains open. Our work seeks to fill this gap by treating these two systems as black boxes to be integrated as modules into a single architecture, without making assumptions on their internal structure and semantics. Instead, we expect only that each module exposes certain methods for accessing the functions that the module implements: the symbolic module exposes a deduction method for computing the function’s output on a given input, and an abduction method for computing the function’s inputs for a given output; the neural module exposes a deduction method for computing the function’s output on a given input, and an induction method for updating the function given input-output training instances. We are, then, able to show that a symbolic module — with any choice for syntax and semantics, as long as the deduction and abduction methods are exposed — can be cleanly integrated with a neural module, and facilitate the latter’s efficient training, achieving empirical performance that exceeds that of previous work.

Parisotto et al. (2017), and open question answering (Sun et al. 2018) settings. In these cases, the training of the neural module is regulated by the logic theory (and its integrity constraints or other constructs), which is far from straightforward since logical inference cannot be, in general, captured via a differentiable function.

To accommodate the integration of neural modules with logical theories, the majority of neural-symbolic frameworks restrict the type of the theories (e.g., to non-recursive or acyclic propositional ones), and they either translate them into neural networks (d’Avila Garcez, Broda, and Gabbay 2002; Hölldobler, Störr, and Kalinke 1999; Towell and Shavlik 1994), or they replace logical computations by differentiable functions (Bošnjak et al. 2017; Gaunt et al. 2017). A second line of work abandons the use of classical logic altogether and adopts theories whose interpretations take continuous values, such as fuzzy logic (Donadello, Serafini, and d’Avila Garcez 2017; Marra et al. 2019; Serafini and d’Avila Garcez 2016; Sourek et al. 2015; van Krieken, Acar, and van Harmelen 2019), or probabilistic logic (Manhaeve et al. 2019), which can support the uniform application of both.
Thank you!