Neural-Symbolic Integration: A Compositional Perspective

Feb 16, 2020  Efi Tsamoura, SAIC-Cambridge
State of the art

- Imposes restrictions on the syntax and the semantics of the logical theories:
  - [2,4] adopt theories with interpretations taking continuous values, e.g., fuzzy logic, probabilistic logic.

- Depends on the semantics and the complexity of the specific theory.
Objectives

- Develop a compositional framework in which users can plug in *any* logical theory and *any* neural component of interest.

**Benefits:**

- Control the inference cost.
- Control the expressive power of the theory (e.g., support for non-monotonic theories not supported by PLP-based neural-symbolic frameworks as [2]).
- Support for techniques coming from the learning theory community (e.g., implicit learning [9]).
Contributions

- A framework supporting those properties [1].

- Beyond the benefits mentioned before, compositionality allows integrating in a natural way the predictions of the neural component during the training process as opposed to prior art, e.g., [2].

- Compositionality is achieved via symbolic modules offering the following interfaces:
  - *deduction*, or forward inference; and
  - *abduction*, through which one computes (i.e., abduces) the inputs to the symbolic module that would deduce a given output.
Integrate a symbolic module adopting a theory $T$ and computing a function $s(\cdot)$ on top of a neural module computing a function $n(\cdot)$.

The translator respects the semantics of the theory, e.g., if $T$ is probabilistic, then each fact is provided along with its confidence/probability.

Assumptions:
- closed-world assumption;
- the semantics of the neural outputs is known.
Setting: inference

\[ n(x) = \omega \]  

\[ r(\omega) = A \]

\[ \text{deduce}(T, A) \]
Goal: given training samples of the form \((x, o)\), train the \textit{neural component}. 

\[ \begin{array}{c}
\text{Neural module} \\
\quad x \rightarrow n \\
\quad n \rightarrow r \\
\text{Translator} \\
\quad r \rightarrow s \\
\text{Symbolic module} \\
\quad s \rightarrow o
\end{array} \]
Given an image of a chessboard and the status of the black king, learn the weights of the neural component.
Training: high-level idea

- Given the target label $o$ compute a formula representing what the neural component should output in order to get the desired output after reasoning.

- The computation of the formula is done via abduction

- Use the computed formula to train the neural component.
Training: how do the training formulas look like?

If we want the output to be safe, the logical component should be provided with the following chessboards:

\[
\begin{align*}
\text{at}(b(k), (2,3)) \\
\text{at}(w(q), (1,1)) \\
\text{at}(w(b), (3,1)) \\
\text{at}(empty, (3,2)) \\
\end{align*}
\]
Abduction

Given:

- a set of rules $P$
- a set of abducible predicates $A$—data that is given as part of the input to the theory—
- a set of integrity constraints $IC$
- a user query $Q$

find a formula $Δ$ over of facts over $A$, such that

- $P \cup Δ \models Q$
- $P \cup Δ \models IC$
Training the neural component using formulas

- The loss function must show how close –\textit{semantically} – are the outputs of the nets to the formula we found via abduction.

- We use \textit{weighted model counting} [11].
Weighted model counting

Consider a propositional formula $\phi$, where each variable $X$ in $\phi$ is associated with a weight $w(X)$ in $[0,1]$.

A *satisfying assignment* $\sigma$ of $\phi$ is a mapping of the variables in $\phi$ to $\top$ or $\bot$, that makes $\phi$ true.

The weight of a satisfying assignment $\sigma$ is defined as

$$\prod_{X \in \phi \mid X = \top} w(X) \times \prod_{X \in \phi \mid X = \bot} 1 - w(X)$$

The weighted model count of $\phi$ is the *sum of the weights of all* satisfying assignments of $\phi$. 
Weighted model counting

\[ \phi = X \lor \neg Y \]

\[ w = \{ X \mapsto 0.9, Y \mapsto 0.1 \} \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>\phi</th>
<th>Weight of assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>((1 - w(X)) \times (1 - w(Y)) = 0.1 \times 0.9)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(w(X) \times (1 - w(Y)) = 0.9 \times 0.9)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(w(X) \times w(Y) = 0.9 \times 0.1)</td>
</tr>
</tbody>
</table>

\[ \phi = X \lor \neg Y \]

\[ w = \{ X \mapsto 0.1, Y \mapsto 0.9 \} \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>\phi</th>
<th>Weight of assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>((1 - w(X)) \times (1 - w(Y)) = 0.9 \times 0.1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(w(X) \times (1 - w(Y)) = 0.1 \times 0.1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(w(X) \times w(Y) = 0.1 \times 0.9)</td>
</tr>
</tbody>
</table>
Consider the formula \( \text{at}(b(k), (2,3)) \land \text{at}(w(q), (1,1)) \land \text{at}(w(b), (3,1)) \).

- Virtually create one network for each cell.
- Associate each net output with a unique Boolean variable.
- The formula becomes \( X_1 \land Y_2 \land Z_9 \).
- Set the weight of each net output as the weight of the corresponding Boolean variable.
- The loss is the negative logarithm of the weighted model count of \( X_1 \land Y_2 \land Z_9 \).
Training: overview

\[ \phi = \text{abduce}(T, o) \]

\[ \nabla L \]

\[ \omega \]

Loss computation

Differentiation

Background knowledge

safe

Abduction
Training: neural-guided abduction

- Abduction was done so far based only on the target label.

- We could consider the neural predictions to narrow down the abductive proofs.

- Benefits: improve training efficiency.
Neural-guided abduction: example

- Recall that when provided with the training pair, the proofs were computed based only on the training label (i.e., safe):

- However, if the neural component is confident in recognizing non-empty cells, i.e., it “sees”:

  then we can exclude all the abductive proofs not abiding this pattern.
To support neural-guided abduction, we need to:

- establish a communication channel between the neural and the logical components;
- extend abduction to deal with noisy or inconsistent neural predictions via proximity functions.
Empirical evaluation

- Benchmarks from [6], [2] and chess scenario.
State of the art

- **DeepProbLog.**
  - Reduces the problem to learning the parameters of probabilistic logic programs.

- **NeurASP**
  - Reduces the problem to learning the parameters of probabilistic answer set programs.

- **ABL**
  - Computes the neural predictions for each element.
  - Obscures subsets of the neural predictions.
  - Abduces the obscured predictions so that the resulting predictions are consistent with the background knowledge.
  - Trains the neural component using obscured and abduced neural predictions.
## Empirical evaluation

<table>
<thead>
<tr>
<th></th>
<th>ADD2x2</th>
<th>OPERATOR2x2</th>
<th>APPLY2x2</th>
<th>DBA(5)</th>
<th>MATH(3)</th>
<th>MATH(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>accur % NLOG</td>
<td>91.7 ± 0.7</td>
<td>90.8 ± 0.8</td>
<td>100 ± 0</td>
<td>95.0 ± 0.2</td>
<td>95.0 ± 1.2</td>
<td>92.2 ± 0.9</td>
</tr>
<tr>
<td>accur % DLOG</td>
<td>88.4 ± 2.5</td>
<td>86.9 ± 1.0</td>
<td>100 ± 0</td>
<td>95.6 ± 1.8</td>
<td>93.4 ± 1.4</td>
<td>timeout</td>
</tr>
<tr>
<td>accur % ABL</td>
<td>75.5 ± 34</td>
<td>timeout</td>
<td>88.9 ± 13.1</td>
<td>79 ± 12.8</td>
<td>69.7 ± 6.2</td>
<td>6.1 ± 2.8</td>
</tr>
<tr>
<td>accur % NASP</td>
<td>89.5 ± 1.8</td>
<td>timeout</td>
<td>76.5 ± 0.1</td>
<td>94.8 ± 1.8</td>
<td>27.5 ± 34</td>
<td>18.2 ± 33.5</td>
</tr>
<tr>
<td>time (s) NLOG</td>
<td>531 ± 12</td>
<td>565 ± 36</td>
<td>228 ± 11</td>
<td>307 ± 51</td>
<td>472 ± 15</td>
<td>900 ± 71</td>
</tr>
<tr>
<td>time (s) DLOG</td>
<td>1035 ± 71</td>
<td>8982 ± 69</td>
<td>586 ± 9</td>
<td>4203 ± 8</td>
<td>1649 ± 301</td>
<td>timeout</td>
</tr>
<tr>
<td>time (s) ABL</td>
<td>1524 ± 100</td>
<td>timeout</td>
<td>1668 ± 30</td>
<td>1904 ± 92</td>
<td>1903 ± 17</td>
<td>2440 ± 13</td>
</tr>
<tr>
<td>time (s) NASP</td>
<td>356 ± 4</td>
<td>timeout</td>
<td>454 ± 652</td>
<td>193 ± 2</td>
<td>125 ± 6</td>
<td>217 ± 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PATH(4)</th>
<th>PATH(6)</th>
<th>MEMBER(3)</th>
<th>MEMBER(5)</th>
<th>CHESS-BSV(3)</th>
<th>CHESS-ISK(3)</th>
<th>CHESS-NGA(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>accur % NLOG</td>
<td>97.4 ± 1.4</td>
<td>97.2 ± 1.1</td>
<td>96.9 ± 0.4</td>
<td>95.4 ± 1.2</td>
<td>94.1 ± 0.8</td>
<td>93.9 ± 1.0</td>
<td>92.7 ± 1.6</td>
</tr>
<tr>
<td>accur % DLOG</td>
<td>timeout</td>
<td>timeout</td>
<td>96.3 ± 0.3</td>
<td>timeout</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>accur % ABL</td>
<td>timeout</td>
<td>timeout</td>
<td>55.3 ± 3.9</td>
<td>49.0 ± 0.1</td>
<td>0.3 ± 0.2</td>
<td>44.3 ± 7.1</td>
<td>n/a</td>
</tr>
<tr>
<td>accur % NASP</td>
<td>timeout</td>
<td>timeout</td>
<td>94.8 ± 1.3</td>
<td>timeout</td>
<td>timeout</td>
<td>19.7 ± 6.3</td>
<td>n/a</td>
</tr>
<tr>
<td>time (s) NLOG</td>
<td>958 ± 89</td>
<td>2576 ± 14</td>
<td>333 ± 23</td>
<td>408 ± 18</td>
<td>3576 ± 28</td>
<td>964 ± 15</td>
<td>2189 ± 86</td>
</tr>
<tr>
<td>time (s) DLOG</td>
<td>timeout</td>
<td>timeout</td>
<td>2218 ± 211</td>
<td>timeout</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>time (s) ABL</td>
<td>timeout</td>
<td>timeout</td>
<td>1392 ± 8</td>
<td>1862 ± 28</td>
<td>9436 ± 169</td>
<td>7527 ± 322</td>
<td>n/a</td>
</tr>
<tr>
<td>time (s) NASP</td>
<td>timeout</td>
<td>timeout</td>
<td>325 ± 3</td>
<td>timeout</td>
<td>timeout</td>
<td>787 ± 307</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Results using 3000 training samples and 3 epochs.
NeuroLog: efficient caching mechanism

Efficient caching:
- Compute an circuit for each abductive formula.
- Use the compute circuit to compute the loss.
- The number of different circuits equals the number of different labels.
NeuroLog vs DeepProbLog and NeurASP

ADD2x2  OPERATOR2x2  APPLY2x2  DBA(n)

MATH(5)  MEMBER(n)  PATH(n)  CHESS-?(n)

Results using 3000 training samples and 3 epochs.
NeuroLog vs ABL

ADD2x2

APPLY2x2

DBA(n)

MATH(3)

MATH(5)

MEMBER(n)

CHESS-BSV(n)

CHESS-ISK(n)

Results using 3000 training samples and 3 epochs.
Summary

- Compositional: users can plug in nets and logic theories of interest, e.g., non-monotonic, probabilistic, action.
- Natural integration of neural predictions during the training process.
- Outperforms state of the art in terms of training time and efficiency.
References