Neural-Symbolic Integration: A Compositional Perspective
State of the art

Imposes restrictions on the syntax and the semantics of the logical theories:

- [5] translates acyclic or propositional theories to neural networks.
- [6] replaces logical computations by differentiable functions.
- [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

\[ n(x) = \omega \]

\[ r(\omega) = A \]

\[ \text{deduce}(T, A) \]
Setting: training

Goal: given training samples of the form \((x, o)\), train the \textit{neural component}. 

\begin{center}
\begin{tikzpicture}

\node[rectangle, draw] (n) at (0,0) {Neural module};
\node[rectangle, draw] (r) at (3,0) {Translator};
\node[rectangle, draw] (s) at (6,0) {Symbolic module};

\draw[->] (n) -- (r) node[midway, above] {\(n\)};
\draw[->] (r) -- (s) node[midway, above] {\(r\)};
\draw[->] (s) -- (s) node[midway, above] {\(o\)};
\end{tikzpicture}
\end{center}
Example: chess

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:

\[
\text{at}(b(k), (2,3)) \\
\text{at}(empty, (3,2)) \\
\text{at}(w(q), (1,1)) \\
\text{at}(w(b), (3,1)) \\
\text{at}(b(k), (2,3)) \land \text{at}(w(q), (1,1)) \land \text{at}(w(b), (3,1)) \land \text{at}(empty, (1,2)) \land \ldots \land \text{at}(empty, (3,2)) \\
\text{at}(w(b), (2,3)) \land \text{at}(w(r), (1,1)) \land \text{at}(w(n), (3,1)) \land \text{at}(empty, (1,2)) \land \ldots \land \text{at}(empty, (3,2)) \\
\text{at}(w(b), (2,3)) \land \text{at}(w(p), (1,1)) \land \text{at}(w(n), (2,2)) \land \text{at}(empty,(1,2)) \land \ldots \land \text{at}(empty, (3,2))
\]
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 $\Delta$ over of facts over $A$, such that

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

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

- We use *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

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

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

\( \phi = X \lor \neg Y \)
\( w = \{X \mapsto 0.9, Y \mapsto 0.1\} \)

| X | Y | \( \phi \) | Weight of assignment |
|---|---|---|---------------------|
| 0 | 0 | 1 | \((1 - w(X)) \times (1 - w(Y)) = 0.1 \times 0.9\) |
| 0 | 1 | 0 |  |
| 1 | 0 | 1 | \(w(X) \times (1 - w(Y)) = 0.9 \times 0.9\) |
| 1 | 1 | 1 | \(w(X) \times w(Y) = 0.9 \times 0.1\) |

\( \phi = X \lor \neg Y \)
\( w = \{X \mapsto 0.1, Y \mapsto 0.9\} \)

| X | Y | \( \phi \) | Weight of assignment |
|---|---|---|---------------------|
| 0 | 0 | 1 | \((1 - w(X)) \times (1 - w(Y)) = 0.1 \times 0.1\) |
| 0 | 1 | 0 |  |
| 1 | 0 | 1 | \(w(X) \times (1 - w(Y)) = 0.1 \times 0.1\) |
| 1 | 1 | 1 | \(w(X) \times w(Y) = 0.1 \times 0.9\) |
Training: an example

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

Background knowledge

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

Abduction

safe

\[ \nabla L \]

Differentiation

\[ L \]

Loss computation

\[ \omega \]
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.
- Competitors: DeepProbLog [2], ABL [12] and NeurASP [13].

\[
\begin{align*}
5 &+ 3 = 8 \\
2 &+ 8 = 10 \\
6 &/ 3 = 2 \\
7 &\times 1 + 4 = 11
\end{align*}
\]
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

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

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

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

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

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

[1] Tsamoura, E. et al. 2021. Neural-Symbolic Integration: A Compositional Perspective. In AAAI, to appear.
[2] Manhaeve, R. et al. 2018. DeepProbLog: Neural Probabilistic Logic Programming. In NeurIPS, 3749–3759.
[3] Van Krieken, E. et al. 2019. Semi-Supervised learning using differentiable reasoning. Journal of Applied Logics, Vol. 6 No. 4.
[4] Donadello, I. et al. 2017. Logic tensor networks for semantic image interpretation. In IJCAI, 1596–1602.
[5] d’Avila Garcez, A. S. et al. 2002. Neural-symbolic learning systems: foundations and applications. Perspectives in neural computing.
[6] Gaunt, A. L. et al. 2017. Differentiable Programs with Neural Libraries. In ICML, 1213–1222.
[7] Zhu, Y. et al. 2014. Reasoning about Object Affordances in a Knowledge Base Representation. In ECCV, 408–424.
[8] Valiant, L. G. 2000. Robust logics. Artificial Intelligence, Vol. 117, 231–253.
[9] Juba, B. 2013. Implicit Learning of Common Sense for Reasoning. In IJCAI, 939–946.
[10] Kakas, A. C. 2017. Abduction. Encyclopedia of Machine Learning and Data Mining, 1–8. Boston, MA: Springer US.
[11] Chavira, M. et al. 2008. On probabilistic inference by weighted model counting. Artificial Intelligence, Vol. 172, No. 6, 772–799.
[12] Dai, W.-Z. et al. 2019. Bridging Machine Learning and Logical Reasoning by Abductive Learning. In NeurIPS, 2815–2826.
[13] Yang, Z. et al. 2020. NeurASP: Embracing Neural Networks into Answer Set Programming. In IJCAI, 1755–1762.