A Simple Extension of Answer Set Programs to Embrace Neural Networks (Extended Abstract)

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

[1] Arizona State University, USA
[2] Samsung Research, S. Korea
{zyang90, aishay, joolee}@asu.edu

This is a brief report on NeurASP [2], 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. NeurASP can make use of a pre-trained neural network in symbolic computation and 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.

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. However, it is still not clear how complex and high-level reasoning, such as default reasoning, ontology reasoning, and causal reasoning, 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), are logic-oriented and do not incorporate high-dimensional feature space as in deep learning, which limits the applicability of KR in many practical applications.

In [2], we present a simple extension of answer set programs by embracing neural networks. Following the idea of DeepProbLog [1], by treating the neural network output as the probability distribution over atomic facts in answer set programs, the proposed formalism called 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 **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 $\mathbb{R}^{1\times 10}$. This 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 $digit_1(d)=0$, "$digit_1(d)=1$, ..., $digit_1(d)=9$."*

Submitted to:

A NeurASP *program* Π is the union of Π^{asp} and Π^{nn}, where Π^{asp} is a set of propositional ASP rules 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 $Head \leftarrow Body$ in Π^{asp}, no atoms in σ^{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 Π, we first obtain its ASP counterpart Π' where each neural atom (1) is replaced with the set of "choice" rules

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

The *stable models* of Π are defined as the stable models of Π'.

To define the probability of a stable model, we first define the probability $P_\Pi(m_i(t)=v_j)$ of an atom $m_i(t)=v_j$ in σ^{nn} as the probability of the j-th outcome of the i-th event outputted by the neural network M upon the input tensor pointed by t. Based on this, the probability of an interpretation I is defined as follows.

$$P_\Pi(I)=\begin{cases}\dfrac{\prod_{i,j\,:\,m_i(t)=v_j\in I|_{\sigma^{nn}}} P_\Pi(m_i(t)=v_j)}{Num(I|_{\sigma^{nn}},\Pi)} & \text{if } I \text{ is a stable model of } \Pi;\\ 0 & \text{otherwise}\end{cases}$$

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

The paper [2] illustrates 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. 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 NeurASP could conclude that it cannot be 1 by applying the constraints for Sudoku.

NeurASP also 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.

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 is to improve computational methods.

References

[1] 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.

[2] 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:10.24963/ijcai.2020/243.