Neural-Symbolic Integration: A Compositional Perspective

Efthymia Tsamoura[1], **Timothy Hospedales**[1], **Loizos Michael**[2,3]
[1] Samsung AI Research
[2] Open University of Cyprus
[3] CYENS Center of Excellence
efi.tsamoura@samsung.com, t.hospedales@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[1].

Introduction

Neural-symbolic frameworks (d'Avila Garcez, Broda, and Gabbay 2002; Rocktäschel and Riedel 2017; Wang et al. 2019) vow to bring a new computational paradigm in which symbolic systems can tolerate noisy or unstructured data, and neural systems can learn with fewer data and offer interpretable outcomes. The potential of integrating a symbolic, typically logic-based, module on top of a neural one has been well-demonstrated in semi-supervised learning (Donadello, Serafini, and d'Avila Garcez 2017; Marra et al. 2019; Serafini and d'Avila Garcez 2016; van Krieken, Acar, and van Harmelen 2019), program induction (Kalyan et al. 2018; 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. 2018), which can support the uniform application of backpropagation on both the symbolic and the neural module.

We consider the problem of integrating a symbolic module that computes a function $s(\cdot)$ on top of a neural module that computes a function $n(\cdot)$, so that together the two modules implement the composition $s \circ n$. We argue that this integration can be done fully compositionally, without the need to revamp the syntax or semantics of either module.

We borrow two well-known notions from mathematical logic to establish the interface that should be provided by the symbolic module to reach a transparent and "non-intrusive" integration: *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.

While abduction has been used in the past as the means to train a neural module feeding into a symbolic module (Dai et al. 2019), there are two key differences between our framework and prior art, over and above our high-level contribution in setting the basis for compositionality. The first difference is on the abduced inputs that are used to train the neural module. Our basic framework makes use of *all* such abduced inputs, while prior art restricts its attention on one of them. As also supported by the empirical evidence that we offer in this work, this restriction causes the learning process to suffer: learning is led to fall into local minima since the single abduced input offers lopsided feedback to the learning process, training faces weaker supervision signals due to

[1] Efthymia Tsamoura and Loizos Michael contributed to the conception and design of the framework, to the implementation of the architecture, to the analysis and interpretation of the results, and to the writing of the paper. Timothy Hospedales contributed to early discussions in the general area of neural-symbolic integration, and proposed some existing benchmarks for the empirical evaluation.

the *loss* of the semantic constraints among the different abduced inputs, and the learning process becomes vulnerable to random supervision signals on those parts of the single abuced input that are forced to take values when they should have semantically been treated as irrelevant.

The second difference is on the training process itself. Prior art uses an ad-hoc training procedure which requires training of the neural module multiple times for the same training sample. That training approach is not only computationally expensive, but it is also difficult to customize on different scenarios. Instead, our framework provides the means to control the training process in a customized manner by delegating to the symbolic module the encoding of any domain-specific training choices. In particular, there exist cases where one would wish to have the neural predictions guide the choice of abduced inputs — presumably the problem that also motivates prior art. We show that such *neural-guided abduction* can be done easily as an extension of our basic framework, by encoding in the symbolic module the knowledge of which abduced inputs are to be used for training, using declarative or procedural techniques to resolve any inconsistencies and to rank the abduced inputs in terms of compatibility with the current neural predictions.

Beyond the plugging in of theories with any semantics and syntax, and beyond the already-mentioned support for neural-guided abduction, the clean take of our proposed compositional architecture easily extends to support other features found in past works, including *program induction* and *domain-wide constraints*. To our knowledge, a uniform handling of all these features is not present in past works.

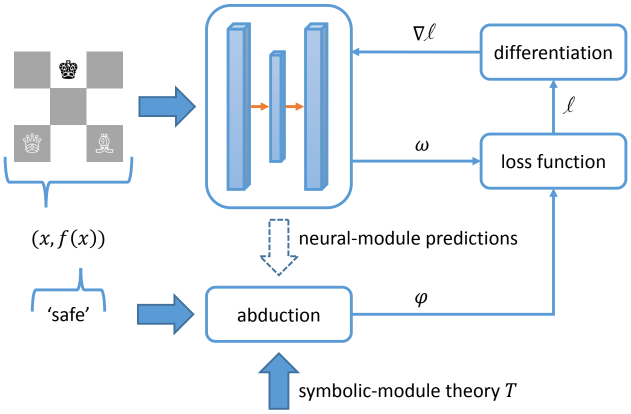
We empirically evaluate — in what we believe to be a more comprehensive manner than typically found in the relevant literature — the performance of our framework against three frameworks that share the same goals with ours: DEEPPROBLOG (Manhaeve et al. 2018), NEURASP (Yang, Ishay, and Lee 2020), and ABL (Dai et al. 2019). We demonstrate the superior performance of our framework both in terms of training efficiency and accuracy over a wide range of scenarios showing the features described above.

Preliminaries

For concreteness of exposition, and without excluding other syntax and semantics, we assume that the symbolic component encodes a logic theory using the standard syntax found in the abductive logic programming literature (Kakas 2017).

As typical in logic programming, the language comprises a set of relational ***predicates*** that hold over ***variables*** or ***constants***. An ***atom*** is a predicate with its arguments. A ***formula*** is defined as a logical expression over atoms, using the logical connectors of Prolog, e.g., conjunction, disjunction, negation. A ***theory*** is a collection of such formulas. Figure 1 shows a theory for determining the status of the game of a certain variant of chess played on a 3×3 board with three pieces: a black king, and two white pieces of different types.

As far as our proposed architecture is concerned, the precise syntax and semantics of the theory are inconsequential. We will, therefore, not delve into a detailed analysis of the aforementioned theory T, except as needed to highlight certain features. What is of only importance is that T is accompanied by an entailment operator $\models$ that allows exposing: a deduction method $\mathtt{deduce}$ that takes as input a set of atoms A and produces a set of atoms $O = \mathtt{deduce}(T, A)$ such that $T \cup A \models O$; an abduction method $\mathtt{abduce}$ that takes as input a set of atoms O and produces a set (out of possibly many) of atoms $A \in \mathtt{abduce}(T, O)$ such that $T \cup A \models O$.



```
safe :- placed(Z1), movable(Z1).
draw :- placed(Z1), \+attacked(Z1), \+movable(Z1).
mate :- placed(Z1), attacked(Z1), \+movable(Z1).

placed(Z1) :- pos(Z1), at(b(k),Z1), pos(Z2), pos(Z3), Z2\=Z3,
              piece(w(P2)), at(w(P2),Z2), piece(w(P3)),
              at(w(P3),Z3).

movable(Z1)  :- pos(Z2), reached(Z2,k,Z1), \+attacked(Z2).
attacked(Z2) :- pos(Z3), piece(w(P)), at(w(P),Z3),
                reached(Z2,P,Z3).

reached((X,Y),k,(PX,PY)) :- abs(X,PX,DX), 1>=DX, abs(Y,PY,DY),
                            1>=DY, sum(DX,DY,S), 0<S.
reached((X,Y),q,(PX,PY)) :- reached((X,Y),r,(PX,PY)).
reached((X,Y),q,(PX,PY)) :- reached((X,Y),b,(PX,PY)).
...

ic :- piece(P), at(P,Z1), at(P,Z2), Z1\=Z2.
ic :- piece(P1), piece(P2), at(P1,Z), at(P2,Z), P1\=P2.
ic :- at(b(k),Z1), at(w(k),Z2), reached(Z1,k,Z2).
ic :- piece(b(P1)), at(b(P1),Z1), piece(b(P2)), at(b(P2),Z2),
      Z1\=Z2.
ic :- piece(w(P1)), at(w(P1),Z1), piece(w(P2)), at(w(P2),Z2),
      piece(w(P3)), at(w(P3),Z3), Z1\=Z2, Z2\=Z3, Z3\=Z1.
```

Figure 1: Snippet of a theory for an example chess domain being used to train a neural module through abduction.

As part of exposing a method one needs to define its input and output spaces. We will assume that $A \subseteq \mathcal{A}$ and call $\mathcal{A}$ the set of ***symbolic inputs*** or ***abducibles***; and that $O \subseteq \mathcal{O}$ and call $\mathcal{O}$ the set of ***symbolic outputs*** or ***outcomes***. We will also assume that atoms in $\mathcal{A}$ and $\mathcal{O}$ are grounded and disjoint. When convenient, we will represent a subset of atoms as a formula: the conjunction of the subset's members. An ***abductive proof*** for a given outcome $O \subseteq \mathcal{O}$ is any formula $A \in \mathtt{abduce}(T, O)$. Observe that for any fixed outcome there might exist zero, one, or multiple abductive proofs.

Example 1 *In our example chess domain, the set $\mathcal{A}$ of abducibles comprises all atoms of the form $\mathsf{at}(P, (X, Y))$, corresponding to the concept of a chess piece of type P being on the chess board at coordinates (X, Y); P takes one of the values in $\{\mathsf{b}(k), \mathsf{w}(k), \mathsf{w}(q), \mathsf{w}(r), \mathsf{w}(b), \mathsf{w}(n), \mathsf{w}(p)\}$, where $\mathsf{w}(\cdot)$ and $\mathsf{b}(\cdot)$ stand for white or black pieces, and k, q, r, b, n, and p denote the king, queen, rook, bishop,* *knight and pawn, respectively; each of X and Y take one of the values in* $\{1, 2, 3\}$. *The set $\mathcal{O}$ of outcomes is equal to* $\{\mathsf{safe}, \mathsf{draw}, \mathsf{mate}\}$, *corresponding to the concepts that the black king has a valid move, is stalemated, or is mated.*

Deduction receives as input a subset of $\mathcal{A}$ that describes the state of the chess board, and produces as output a (singleton) subset of $\mathcal{O}$ on the status of the black king. Conversely, abduction receives as input a (singleton) subset of $\mathcal{O}$ that describes the desired status of the black king, and produces as output subsets of $\mathcal{A}$, each describing a state of the chess board where the black king has the desired status.

A theory may be extended with ***integrity constraints***, special formulas that restrict the possible inferences that can be drawn when applying the methods of deduction and abduction, by constraining which subsets of $\mathcal{A}$ are considered acceptable. A subset $A \subseteq \mathcal{A}$ violates the integrity constraints if and only if $\texttt{deduce}(T, A)$ is a special symbol $\perp \notin \mathcal{O}$. Analogously, a subset $A \subseteq \mathcal{A}$ violates the integrity constraints if and only if $A \notin \texttt{abduce}(T, O)$ for each subset $O \subseteq \mathcal{O}$. Thus, integrity constraints in a theory need to be respected by every abductive proof for each outcome $O \subseteq \mathcal{O}$.

Example 2 *In our example chess domain, the integrity constraints are encoded as rules with an* `ic` *head. The five integrity constraints in Figure 1 capture, in order, the following requirements: the same piece type is not at more than one position; no two pieces are at the same position; the black and white kings are not attacking each other; there is at most one black piece on the chess board; there are at most two white pieces on the chess board. The requirement for the existence of at least one black king and at least two white pieces is captured through the rule with the* `placed(Z1)` *head. If the set $\mathcal{A}$ of abducibles is extended to include all atoms of the form* $\mathsf{empty}((X, Y))$ *to denote explicitly the coordinates of the board cells that are empty, then additional integrity constraints and rules can be added in the theory to ensure that no piece can be placed at an empty cell, an that every non-empty cell should hold some piece.*

Framework

We consider a neural-symbolic system built by composing a neural module feeding into a symbolic module.

Module Compositionality

We let $\mathcal{X}$ and $\Omega = [0, 1]^k$ be, respectively, the space of possible inputs and the space of possible outputs of the neural module. At any given training iteration t, the neural module effectively implements a function $n_t : \mathcal{X} \rightarrow \Omega$. For notational simplicity, we will overload the use of the symbol n_t to denote both the function and the underlying neural network itself. We assume that there is a ***translator*** function r that maps each $\omega \in \Omega$ to a set of abducibles $r(\omega) \in \mathcal{A}$.

Given a symbolic module with a theory T, the ***end-to-end reasoning*** of the neural-symbolic system at iteration t is the process that maps an input in $\mathcal{X}$ to an outcome subset of $\mathcal{O}$ as follows: the system receives an input $x \in \mathcal{X}$; the neural module computes the vector $\omega = n_t(x)$; the translator maps ω to the abducibles $A = r(\omega) \subseteq \mathcal{A}$; the symbolic module computes the outcome $O = \texttt{deduce}(T, A) \subseteq \mathcal{O} \cup \{\perp\}$. Thus, *inference* in our framework proceeds by running the inference mechanism of the symbolic module over the inferences of the neural module on a given neural input. To simplify our notation, and when there is no confusion, we will write $h_t^T(x)$ to mean $\texttt{deduce}(T, r(n_t(x)))$ for $x \in \mathcal{X}$.

Example 3 *In our example chess domain, consider a neural module n_t that receives as input $x \in \mathcal{X}$ a 3×3 grid of images representing a chess board. The neural module outputs a vector $\omega = n_t(x) \in \Omega^k$ that corresponds to what the neural module predicts. One possible implementation is for the neural module to have eight output nodes for each cell at coordinates (X, Y) of the chess board (hence, $k = 8 \times 9$). These eight output nodes represent, respectively, whether their associated cell includes no piece, the black king, the white king, the white queen, the white rook, the white bishop, the white knight, or the white pawn. ω assigns, effectively, confidence values on each of these predictions for each cell.*

The translator function r could simply turn ω into a set of abducibles A by considering for each cell the most confident prediction and including the corresponding atom in A. Thus, if the first eight components of ω, which correspond to predictions for cell $(1, 1)$, were such the third value was the maximum one, then A would include $\mathsf{at}(\mathsf{w}(k), (1, 1))$.

A is provided as input to the symbolic component, which deduces whether the chess board is in a safe, draw, or mate state (or in $\perp$ in case A violates the integrity constraints).

In certain cases, the input $x \in \mathcal{X}$ to a neural-symbolic system might be associated with explicit input-specific knowledge provided in the form of a symbolic formula $\overline{x}$. This side-information does not go through the usual pipeline as x, but can be readily accommodated by extending the theory T to include it, and by computing $\texttt{deduce}(T \cup \{\overline{x}\}, r(n_t(x)))$ instead. Our compositional perspective affords us to remain agnostic, at the architecture level, on how side-information will be dealt with by the symbolic module (e.g., as integrity constraints or as weak preferences), and puts the burden on the theory itself to make this domain-specific determination.

Neural-Module Learning

As in standard supervised learning, consider a set of labeled samples of the form $\{\langle x_j, f(x_j)\rangle\}_j$, with f being the target function that we wish to learn, x_j corresponding to the features of the sample, and $f(x_j)$ being the label of the sample.

In the context of our neural-symbolic architecture, learning seeks to identify, after t iterations over a training subset of labeled samples, a hypothesis function $h_t^T(\cdot)$ that sufficiently approximates the target function $f(\cdot)$ on a testing subset of labeled samples. Given a fixed theory T for the symbolic module, the only part of the hypothesis function $h_t^T(\cdot) = \texttt{deduce}(T, r(n_t(\cdot)))$ that remains to be learned is the function n_t implemented by the neural module.

We put forward Algorithm 1 to achieve this goal. In line with our compositional treatment, the algorithm does not delve into the internals of the neural and the symbolic module, but accesses them only through the methods that they expose: inference and backpropagation for the neural module; deduction and abduction for the symbolic module.

Algorithm 1 $\texttt{TRAIN}(x, f(x), n_t) \rightarrow n_{t+1}$

```
$\omega := n_t(x)$
$\varphi := \bigvee \texttt{abduce}(T, f(x))$                    ▷ basic form *or*
$\varphi := \bigvee \texttt{abduce}(T \cup r(\omega), f(x))$         ▷ NGA form
$\ell := \mathsf{loss}(\varphi, r, \omega)$                       ▷ using WMC
$n_{t+1} := \mathsf{backpropagate}(n_t, \bigtriangledown \ell)$
**return** $n_{t+1}$
```

The algorithm considers the label $f(x)$ of a given sample, viewed as a (typically singleton) subset of $\mathcal{O}$, and abduces *all* abductive proofs $A \in \texttt{abduce}(T, f(x)) \subseteq \mathcal{A}$. Taking the disjunction of all abductive proofs, the algorithm computes the ***abductive feedback*** formula φ that captures all the acceptable outputs of the neural module that would lead, through the theory T, the system to correctly infer $f(x)$.

The abductive feedback acts as a supervision signal for the neural module. Combining that signal with the actual output ω of the neural module (through the use of the translator function r), we can compute the loss of the neural module. *Critically, the resulting loss function is differentiable, even if the theory T of the symbolic module is not!* By differentiating the loss function we can use backpropagation to update the neural module to implement function n_{t+1}.

Rather than requiring for the theory to be differentiable, as done in certain past works (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; Manhaeve et al. 2018), the use of abduction for neural-symbolic integration poses no a priori constraints on the form of the theory, but proceeds to extract its "essence" in a differentiable form, albeit in an outcome-specific manner. Fortuitously, the space of possible outcomes is usually considerably restricted, which readily allows the caching of the abductive proofs, or even their precomputation prior to the training phase. Put differently, the use of abduction allows replacing any arbitrary theory T by the set of its abductive feedbacks $\{\varphi_O \mid \varphi_O = \bigvee \texttt{abduce}(T, O), O \subseteq \mathcal{O}\}$.

Example 4 *In our example chess domain, consider a training sample* $(x, f(x))$*, where* x *is a* 3×3 *grid of images representing a chess board with a white queen at cell* $(1, 1)$*, a white bishop at cell* $(3, 1)$*, and a black king at cell* $(2, 3)$*, and* $f(x)$ *labels the chess board as being in a safe state. Starting from the label, we compute the abductive feedback* $\ldots \vee [\mathsf{at}(\mathsf{w}(q), (1, 1)) \wedge \mathsf{at}(\mathsf{w}(b), (3, 1)) \wedge \mathsf{at}(\mathsf{b}(k), (2, 3)) \wedge \ldots \wedge \mathsf{empty}((3, 3))] \vee [\mathsf{at}(\mathsf{w}(r), (1, 1)) \wedge \mathsf{at}(\mathsf{w}(n), (3, 1)) \wedge \mathsf{at}(\mathsf{b}(k), (2, 3)) \wedge \ldots \wedge \mathsf{empty}((3, 3))] \vee [\mathsf{at}(\mathsf{b}(k), (1, 1)) \wedge \mathsf{at}(\mathsf{w}(p), (3, 1)) \wedge \mathsf{at}(\mathsf{w}(r), (2, 3)) \wedge \ldots \wedge \mathsf{empty}((3, 3))] \vee [\mathsf{at}(\mathsf{w}(p), (1, 1)) \wedge \mathsf{at}(\mathsf{w}(n), (2, 2)) \wedge \mathsf{at}(\mathsf{b}(k), (2, 3)) \wedge \ldots \wedge \mathsf{empty}((3, 3))] \vee \ldots$*. Among the shown disjuncts, the first one represents the input chess board, the next two represent chess boards that are safe and have pieces only at cells* $(1, 1)$*,* $(3, 1)$ *and* $(2, 3)$*, and the last represents a chess board that is safe, but has pieces at cells* $(1, 1)$*,* $(2, 2)$ *and* $(2, 3)$*.*

Neural-Guided Abduction

Although computing the entire abductive feedback is generally the appropriate choice of action, there might exist circumstances where it might be beneficial to prune some of its parts. Caution should, however, be exercised, as pruning might end up removing the part of the abductive feedback that corresponds to the true state of affairs (cf. Example 4), and might, thus or otherwise, misdirect the learning process.

One case worth considering is ***neural-guided abduction*** (NGA), where the prediction of the neural module is used as a focus point, and only abductive proofs that are proximal perturbations of that point find their way into the abductive feedback. What counts as a perturbation, how proximity is determined, and other such considerations are ultimately domain-specific, and are not specified by the framework.

Example 5 *In our example chess domain, consider a neural module that is highly confident in distinguishing empty from non-empty cells, but less confident in determining the exact types of the pieces in the non-empty cells. Consider, further, a particular training sample* $\langle x, f(x) \rangle$ *on which the neural component identifies the non-empty cells as being* $(1, 1)$*,* $(3, 1)$*, and* $(2, 3)$*. It is then natural for the symbolic module to attempt to utilize the predictions of the neural module to prune and focus the abductive feedback that it will provide for the further training of the neural module.*

If, for example, $f(x)$ *labels the chess board as being in a safe state, then the abductive feedback will exclude the last disjunct from Example 4, since it represents a chess board with pieces at cells other than* $(1, 1)$*,* $(3, 1)$*, and* $(2, 3)$*, and will maintain the first three disjuncts as they respect the neural predictions in terms of the positions of the three pieces.*

To support neural-guided abduction, we must, first, establish a communication channel between the neural module and the abduction mechanism, in order for the neural module to provide its predictions to the abduction mechanism.

Our proposed architecture can seamlessly implement this communication channel by treating the communicated information as input-specific knowledge. Given, therefore, a training sample $(x, f(x))$, we can simply call the abduction method not by providing only the theory T and the outcome $f(x)$ as inputs, but by first extending the theory T with the neural predictions $\omega = n_t(x)$ as translated by the translator function r. Thus, the abductive feedback in Algorithm 1 is now computed as $\varphi := \bigvee \texttt{abduce}(T \cup r(\omega), f(x))$.

As we have already mentioned, the treatment of this side-information is not determined by the framework, but is left to the theory itself. Although the side-information might, in some domains, provide confident predictions that could act as hard constraints for the theory (cf. Example 5), our treatment allows also the handling of domains where the side-information might be noisy, incorrect, or even in direct violation of the existing integrity constraints of the theory.

Such neural predictions might still offer some useful guidance to the abduction process. Depending on the syntactic and semantic expressivity of the symbolic module, the theory can provide a declarative or a procedural way to resolve the inconsistencies that arise in a domain-specific manner.

Example 6 *In our example chess domain, consider a particular training sample* $\langle x, f(x) \rangle$ *on which the prediction of the neural module, as translated by the translator into* *symbolic inputs, corresponds to the subset* $\{\mathsf{at}(\mathsf{w}(q),(1,1)),$ $\mathsf{at}(\mathsf{w}(b),(3,1)),$ $\mathsf{at}(\mathsf{b}(k),(2,3)),\ldots,\mathsf{empty}((3,3))\}$.

Assume, first, that $f(x)$ labels the chess board as being in a safe state. Then, there exists exactly one abductive proof that matches the neural prediction perfectly. As this corresponds to a zero-cost perturbation of the neural prediction, only it ends up in the abductive feedback. As a result, the neural module ends up reinforcing exactly what it predicted.

Assume, now, that $f(x)$ labels the chess board as being in a draw state. Then, there is no abductive proof that matches the neural prediction perfectly. Rather, there is an abductive proof $[\mathsf{at}(\mathsf{w}(q),(1,1)) \wedge \mathsf{at}(\mathsf{w}(r),(3,1)) \wedge \mathsf{at}(\mathsf{b}(k),(2,3)) \wedge \ldots \wedge \mathsf{empty}((3,3))]$ *that differs from the neural prediction only in changing the type of an already predicted white piece, while maintaining its position, and also maintaining the types and positions of the other two pieces. This abductive proof could be evaluated to have a minimal-cost among the perturbations of the neural prediction, and only it ends up in the abductive feedback. As a result, the neural module ends up reinforcing parts of what it sees, while helping revise locally one of its mistakes (perhaps because it is still unable to fully differentiate between rooks and bishops).*

Assume, finally, that $f(x)$ labels the chess board as being in a mate state. Then, there is no abductive proof that matches the neural prediction perfectly. In fact, there are no abductive proofs that respect the positions of the pieces as predicted by the neural module. Abduction will then seek to identify perturbations that, if possible, move a single piece with respect to the predicted ones, or move and change the type of a single piece, etc., that would respect the label $f(x)$. Depending on how one costs the various perturbations, one or more abductive proofs can be evaluated to have minimal-cost, and all those will end up in the abductive proof.

Evaluation

We have empirically assessed the training time and test accuracy of our proposed compositional framework, hereafter abbreviated as NEUROLOG, against three prior approaches that share the same goals with us: DEEPPROBLOG (Manhaeve et al. 2018), NEURASP (Yang, Ishay, and Lee 2020) and ABL (Dai et al. 2019). Comparing with other architectures, such as (Gaunt et al. 2017), which are concerned not only with neural-module learning, but also with symbolic-module learning, is beyond the scope of the current paper.

The code and data to reproduce the experiments are available at: https://bitbucket.org/tsamoura/neurolog/src/master/.

Implementation

Abductive feedback in NEUROLOG was computed using the A-system (Nuffelen and Kakas 2001) running over SICStus Prolog 4.5.1. Each abductive feedback φ was grounded (and, hence, effectively propositional) by construction, which facilitated the use of semantic loss (Xu et al. 2018) for training the neural module. The semantic loss of φ was computed by treating each atom in φ as a Boolean variable, weighted by the activation value of the corresponding output neuron of the neural module, and by taking the negative logarithm of its *weighted model count* (WMC) (Chavira and Darwiche 2008). For the purposes of computing WMC, φ was first compiled into an arithmetic circuit (Darwiche 2011).

In order to avoid recomputing the same models or the same abductive feedbacks during training, we used caching across all the systems that we evaluated. Furthermore, we encoded the theories of the symbolic modules with an eye towards minimizing the time to perform abduction, grounding, or inference. Experiments were ran on an Ubuntu 16.04 Linux PC with Intel i7 64-bit CPU and 94.1 GiB RAM.

Scenarios

Benchmark datasets have been used to provide inputs to the neural module as follows: MNIST (LeCun et al. 1998) for images of digits; HASY (Thoma 2017) for images of math operators; GTSRB (Stallkamp et al. 2011) for images of road signs. Below we describe each experimental scenario:

ADD2x2 (Gaunt et al. 2017): The input is a 2×2 grid of images of digits. The output is the four sums of the pairs of digits in each row / column. The symbolic module computes the sum of pairs of digits.

OPERATOR2x2 (new; ADD2x2 with program induction): The input is a 2×2 grid of images of digits. The output is the four results of applying the math operator *op* on the pairs of digits in each row / column. The math operator *op* in $\{+,-,\times\}$ is fixed for each row / column but *unknown*. The symbolic module computes the sum, difference, and product of pairs of digits. The neural module seeks to induce the unknown operator and to recognize the digits.

APPLY2x2 (Gaunt et al. 2017): The input is three digits d_1, d_2, d_3 and a 2×2 grid of images of math operators $op_{i,j}$. The output is the four results of applying the math operators in each row / column on the three digits (e.g., $d_1\ op_{11}\ d_2\ op_{12}\ d_3$). The symbolic module computes results of applying pairs of math operators on three digits.

DBA(n) (Dai et al. 2019): The input is a mathematical expression comprising n images of $\{0,1\}$ digits and math operators (including the equality operator). The output is a truth value indicating whether the mathematical expression is a valid equation. The symbolic module evaluates the validity of an equation. Our DBA scenario extends that from (Dai et al. 2019) by allowing math operators to appear on both sides of the equality sign.

MATH(n) (Gaunt et al. 2017): The input is a mathematical expression comprising n images of digits and math operators. The output is the result of evaluating the mathematical expression. The symbolic module computes results of math operators on integers.

PATH(n) (Gaunt et al. 2017): The input is an $n \times n$ grid of images of road signs and two symbolically-represented grid coordinates. The output is a truth value indicating whether there exists a path from the first to the second coordinate. The symbolic module determines valid paths between coordinates given as facts.

MEMBER(n) (new): The input is a set of n images of digits and a single symbolically-represented digit. The output is a truth value indicating whether the single digit appears

	ADD2x2	OPERATOR2x2	APPLY2x2	DBA(5)	MATH(3)	MATH(5)
NLog	91.7 ± 0.7	90.8 ± 0.8	100 ± 0	95.0 ± 0.2	95.0 ± 1.2	92.2 ± 0.9
DLog	88.4 ± 2.5	86.9 ± 1.0	100 ± 0	95.6 ± 1.8	93.4 ± 1.4	timeout
ABL	75.5 ± 34	timeout	88.9 ± 13.1	79 ± 12.8	69.7 ± 6.2	6.1 ± 2.8
NASP	89.5 ± 1.8	timeout	76.5 ± 0.1	94.8 ± 1.8	27.5 ± 34	18.2 ± 33.5
NLog	531 ± 12	565 ± 36	228 ± 11	307 ± 51	472 ± 15	900 ± 71
DLog	1035 ± 71	8982 ± 69	586 ± 9	4203 ± 8	1649 ± 301	timeout
ABL	1524 ± 100	timeout	1668 ± 30	1904 ± 92	1903 ± 17	2440 ± 13
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)
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
DLog	timeout	timeout	96.3 ± 0.3	timeout	n/a	n/a	n/a
ABL	timeout	timeout	55.3 ± 3.9	49.0 ± 0.1	0.3 ± 0.2	44.3 ± 7.1	n/a
NASP	timeout	timeout	94.8 ± 1.3	timeout	timeout	19.7 ± 6.3	n/a
NLog	958 ± 89	2576 ± 14	333 ± 23	408 ± 18	3576 ± 28	964 ± 15	2189 ± 86
DLog	timeout	timeout	2218 ± 211	timeout	n/a	n/a	n/a
ABL	timeout	timeout	1392 ± 8	1862 ± 28	9436 ± 169	7527 ± 322	n/a
NASP	timeout	timeout	325 ± 3	timeout	timeout	787 ± 307	n/a

Table 1: Empirical results. NLog stands for NeuroLog, DLog for DeepProbLog and NASP for NeurASP. The first four rows in each table show the % testing accuracy, while the last four rows show the total training time in seconds.

in the set of digits. The symbolic module determines set membership of an element given as a fact.

We have also used the chess domain from our running example to highlight certain (new) features of our framework: *a richer class of theories*, *non-declarative theories*, and *neural-guided abduction*. We denote by **CHESS-BSV**(n) and **CHESS-NGA**(n), the scenarios corresponding, respectively, to Example 4 and Example 6: in the former scenario, the full abductive feedback is used to train the neural module, and in the latter scenario a *non-declarative* theory is used to enumerate and evaluate, against the neural predictions, the various abductive proofs to select which parts of the abductive feedback to retain. We also consider a third variant that sits between the former two, called **CHESS-ISK**(n), which roughly corresponds to Example 5, but rather than receiving the positions of the three pieces from a confident neural module, it receives them as externally-provided (and noiseless) information. In all scenarios, the chess pieces are represented by images of digits.

Results and Analysis

Results of our empirical evaluation are shown in Table 1, and in Figures 2, 3, and 4. Each system was trained on a training set of 3000 samples, and was ran independently 10 times per scenario to account for the random initialization of the neural module or other system stochasticity. Training was performed over 3 epochs for NeuroLog, DeepProbLog and NeurASP, while the training loop of ABL was invoked 3000 times. Note that there is no one-to-one correspondence between the training loop of ABL and that of the other three systems: in each iteration, ABL considers multiple training samples and based on them it trains the neural component multiple times. In all systems, the neural module was trained using the Adam algorithm with a learning rate of 0.001.

Results on running DeepProbLog on the CHESS-?(n) suite of scenarios are not available, since DeepProbLog's syntax does not readily support the integrity constraints (nor the procedural constructs for the CHESS-NGA(n) scenario) in the symbolic module. Since neural-guided abduction is not supported by any of the other three systems, we report results on CHESS-NGA only for NeuroLog.

The results offer support for the following conclusions:

(C1) The average accuracy of NeuroLog is comparable to, or better than, that of the other systems. NeuroLog performs similarly to DeepProbLog on those scenarios that are supported by the latter and in which DeepProbLog does not time out, while it may perform considerably better than NeurASP and ABL. For example, the average accuracy of NeuroLog is up to 70% higher than that of NeurASP in the MATH scenarios, and up to 40% higher than that of NeurASP in the MEMBER scenarios.

NeurASP and ABL are vulnerable to weak supervision signals, as their performance decreases when the number of abductive proofs per training sample increases. For example, the average accuracy of ABL drops from 69.7% in MATH(3) to 6.1% in MATH(5), while it drops from 44% in CHESS-ISK(n), where each training sample is provided with the coordinates of the non-empty cells, to less than 1% in CHESS-BSV(n) where no such information is provided.

With regards to ABL, this phenomenon may be attributed to the consideration of a *single* abductive proof per training sample instead of considering all the relevant abductive proofs as NeuroLog does. Considering a single abductive proof may result in excluding the correct one; i.e., the one corresponding to the true state of the sample input. Notice that when the number of abductive proofs per training sample increases, the probability of excluding the right abductive proof from consideration increases as well, resulting in very weak supervision signals, as seen in CHESS-BSV.

(C2) Compared to NeurASP and ABL, NeuroLog is

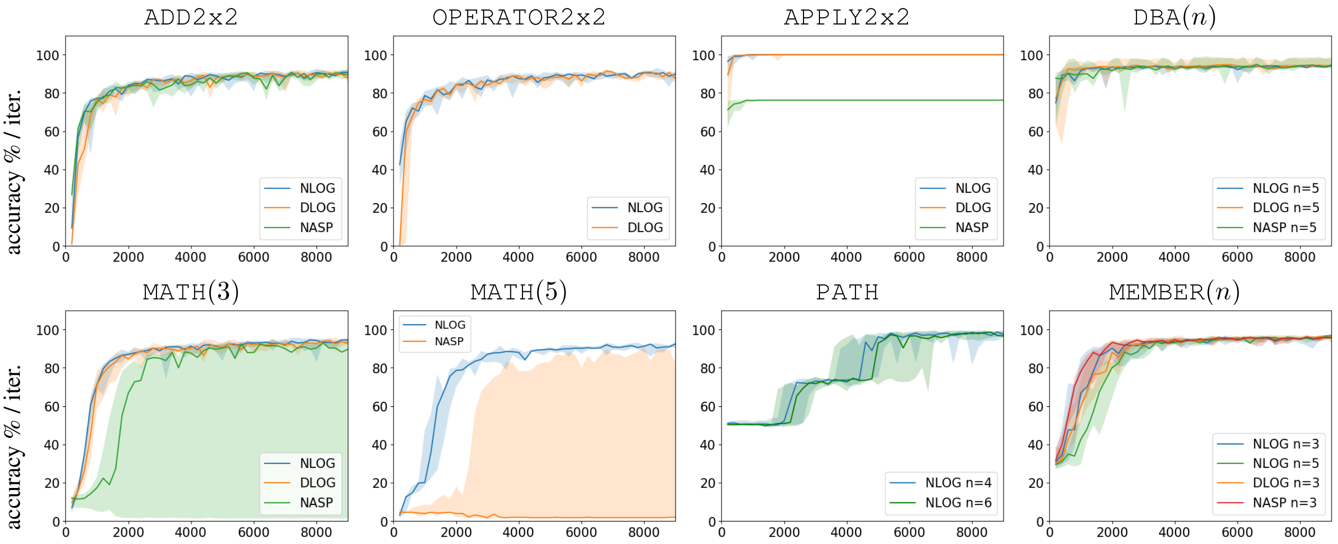


Figure 2: Empirical results for NEUROLOG, DEEPPROBLOG, and NEURASP. Solid lines and lightly-colored areas show, respectively, the average behavior and the variability of the behavior across different repetitions.

less sensitive to the initialization of the neural module. For example, the accuracy of NEURASP spans 15%–94% in MATH(3), and that of ABL spans 57%–94% in APPLY2x2.

With regards to ABL, this sensitivity may be, again, attributed to the consideration of a *single* abductive proof per training sample. The learning process of ABL obscures and abduces part of the neural predictions, so that the modified predictions are consistent with the theory and also lead to the entailment of the sample label (see the "Related Work" section). Considering a single abductive proof has high chances of missing the right one, and hence the training process ends up being biased on the obscuring process, which, in turn, depends upon the initial weights of the neural module.

(C3) The average training time of NEUROLOG may be significantly less than that of the other systems. For example, the average total training time is: 16m47s for NEUROLOG in MATH(5) versus 22m48s for DEEPPROBLOG in the simpler MATH(3) scenario; 42m93s for NEUROLOG in PATH(6) versus DEEPPROBLOG and NEURASP timing out in the simpler PATH(4) scenario; 16m for NEUROLOG in CHESS-ISK(3) versus 125m for ABL in the same scenario.

With regards to ABL, its high training time may be attributed to its trial-and-error use of abduction. At each training iteration, an optimization process obscures and performs abduction *multiple times* over different subsets of the training samples. It holds, in particular, that although ABL computes a single abductive proof per training sample, it may perform abduction multiple times for the same sample.

With regards to NEURASP, its high training time may be attributed to the grounding that NEURASP applies on the theory; i.e., computing all the consequences that are semantically entailed. Instead of computing all such forward-reasoning consequences, abduction is driven by the sample label, and evaluates (and grounds) only the relevant part of the theory. It is worth noting, however, that NEURASP achieves comparable accuracy to NEUROLOG in less training time in the ADD2x2 and DBA scenarios. Its training time is also lower than that of NEUROLOG in the two MATH(n) scenarios, however, for these cases its accuracy is very poor.

(C4) When compared to CHESS-BSV(3), the use of side-information in CHESS-ISK(3) and CHESS-NGA(3) leads to asymptotically faster training. The higher training time during the earlier iterations, which is particularly pronounced in the CHESS-NGA(3) scenario (see Figure 4), corresponds to the phase where new abductive proofs are still being computed. Recall that in CHESS-BSV(3) an abductive proof is distinct for each label (i.e., mate, draw, safe), whereas in CHESS-ISK(3) and CHESS-NGA(3) an abductive proof is distinct for each combination of label and side-information. Once the bulk of the distinct abductive proofs is computed and cached, the training time per iteration drops. Unsurprisingly, this initial phase is longer for the CHESS-NGA(3) scenario, where the distinct abductive proofs are more, as they depend on a more variable space of side-information.

The average end accuracy for the CHESS-?(3) scenarios is comparable; see Table 1. The average *interim* accuracy of CHESS-NGA(3) is, however, relatively lower during early training, where the neural module predictions are still highly noisy / random. Specifically, the average accuracy at 1000 iterations is: 73.9 ± 1.5 for CHESS-BSV(3), 73.4 ± 5.2 for CHESS-ISK(3), 51.1 ± 7.9 for CHESS-NGA(3).

Scalability: Computing abductive proofs is intractable (NP-hard to decide their existence; #P-hard to enumerate/count them). Neural-guided abduction reduces this cost in practice by excluding irrelevant proofs, but the problem remains worst-case hard. However, since abductive proofs are a function of only the sample label and side-information, NEUROLOG can cache and reuse them across different training samples, showing that in practice our approach can be more computationally efficient than prior art, e.g., ABL.

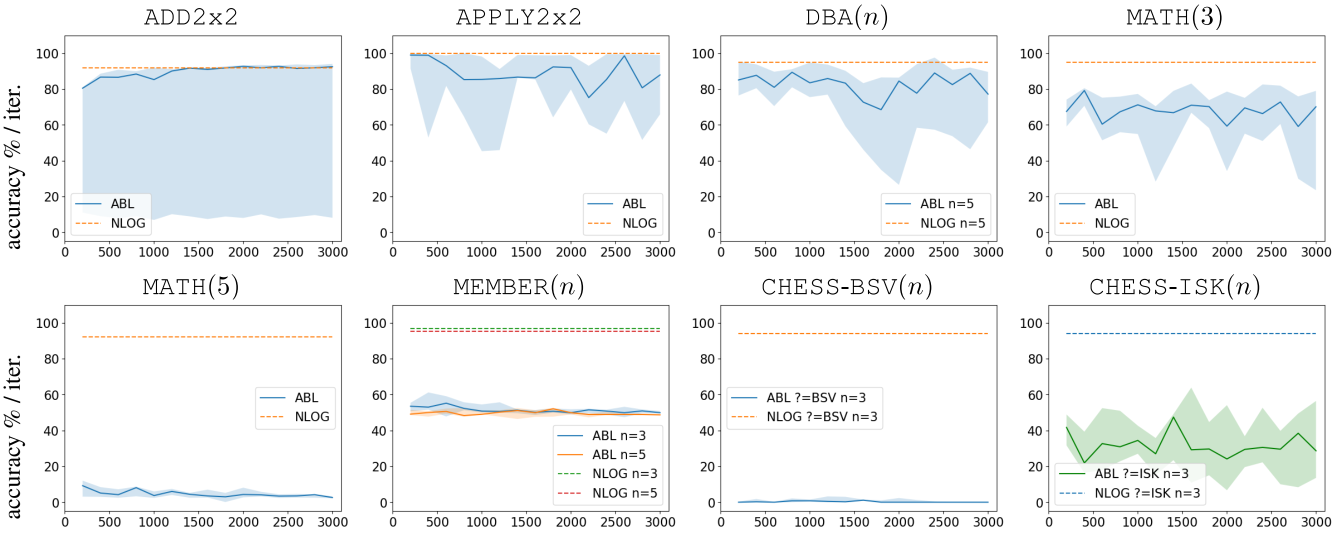


Figure 3: Empirical results for ABL. Solid lines and lightly-colored areas show, respectively, the average behavior and the variability of the behavior across different repetitions. Dashed lines show the final average accuracy of NEUROLOG as reported in Figure 2. Due to the different training regimes of ABL and NEUROLOG, an iteration-by-iteration comparison is not meaningful.

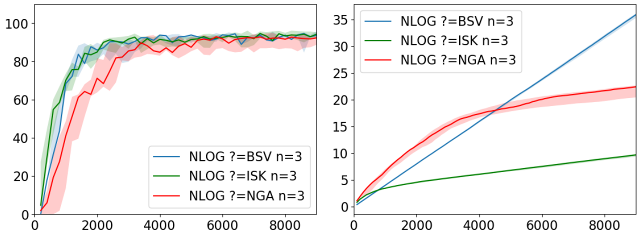


Figure 4: Test accuracy percentage (left) and training time in minutes (right) versus number of iterations for CHESS-?(n).

Related Work

Although ABL shares with NEUROLOG the high-level idea of using abduction, it does so by employing an ad hoc optimization procedure. In each training iteration over a given set $\{\langle x^j, f(x^j)\rangle\}_j$ of training samples, ABL first considers different subsets S^t of the training set and performs the following steps: *(i)* it gets the neural predictions for each element in S^t, *(ii)* it obscures a subset of the neural predictions (both within the same and across different training samples), and *(iii)* it abduces the obscured predictions so that the resulting predictions are consistent with the background knowledge. Let S^* be the largest S^t satisfying the theory after obscuring and abducing. For each $s_i \in S^*$, ABL trains multiple times the neural component using obscured and abduced neural predictions. As our empirical results show, the optimization procedure applied to obscure and abduce the neural predictions may be time-consuming and ineffective, even though a single abductive proof is computed each time.

The second system which shares NEUROLOG's objectives is DEEPPROBLOG (Manhaeve et al. 2018), which works by reducing the problem of learning and inference of a neural-symbolic system to the problem of learning and inference over a probabilistic logic program. The reduction works by treating the outputs of the neural module as probabilistic facts. The accuracy of DEEPPROBLOG is comparable to that of NEUROLOG, but it supports fewer semantic constructs (e.g., integrity constraints) and it requires significantly more training time. NEURASP shares the same high-level approach with DEEPPROBLOG, but reduces, instead, to the more expressive probabilistic answer set programs (ASP) (Yang, Ishay, and Lee 2020). As supported by our empirical evidence, its performance may be lower than that of NEUROLOG. Furthermore, using an ASP solver may be computationally expensive, since it involves computing all the consequences that are semantically entailed by the theory. Instead, our training approach is goal-driven, since the computation of the abductive feedback involves evaluating only the part of the theory that relates to the training label.

Our work can be seen as an extension to (Xu et al. 2018), where a given *fixed* propositional formula is used to define a loss function based on weighted model counting, and this loss function is used, in turn, to regulate the training of the neural module. In contrast to (Xu et al. 2018), our work computes a sample-specific formula to regulate the training of the neural component based on the label of each sample.

Broader area of neural-symbolic integration: The work in (Parisotto et al. 2017; Kalyan et al. 2018; Balog et al. 2017) uses ML to help perform faster and more data efficient program induction, while saving from designing heuristics. To tackle rule induction in the presence of noisy examples, the work in (Evans and Grefenstette 2018) reduces inductive logic programming to a problem of minimizing a differentiable loss. Other frameworks that deal with rule induction under noisy data are Neural Logic Programming (Yang, Yang, and Cohen 2017) and DRUM (Sadeghian et al. 2019).

Neural Theorem Prover (Rocktäschel and Riedel 2017) is an alternative to Prolog's QA engine to support noisy theories. It proceeds by embedding predicates and constants into a vector space and uses vector distance measures to compare them. Neural Logic Machines (Dong et al. 2019) implements rules inside a tensor network providing thus the ability to reason uniformly over neural modules and logical theories. However, its semantics is not connected to any logic semantics (e.g., Tarski, Sato, or fuzzy) and no soft or hard constraints are imposed at inference time.

Conclusion

We have introduced a compositional framework for neural-symbolic integration that utilizes abduction to support a uniform treatment of symbolic modules with theories beyond any specific logic, or a declarative representation altogether. Our empirical results have demonstrated not only the practical feasibility of this perspective, but also its superior performance over state-of-the-art approaches in terms of cross-domain applicability, testing accuracy, and training speed.

Two are the key directions for future work: *(i)* further consideration of the use of non-logic or non-declarative theories for the symbolic module; *(ii)* explicit treatment of symbolic-module learning, which, unlike program induction, will not delegate the burden of learning to the neural module. With respect to the latter direction, in particular, the consideration of human-in-the-loop learning paradigms (such as the *Machine Coaching* paradigm (Michael 2019), for example) would present an interesting challenge for neural-symbolic integration systems, bringing into focus the issue of learning in a manner that is cognitively-compatible with humans.

Acknowledgements

This work was supported by funding from the EU's Horizon 2020 Research and Innovation Programme under grant agreements no. 739578 and no. 823783, and from the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination, and Development. The authors would like to thank Antonis Kakas for help with the abduction system used in this work.

References

Balog, M.; Gaunt, A. L.; Brockschmidt, M.; Nowozin, S.; and Tarlow, D. 2017. DeepCoder: Learning to Write Programs. In *ICLR*.

Bošnjak, M.; Rocktäschel, T.; Naradowsky, J.; and Riedel, S. 2017. Programming with a Differentiable Forth Interpreter. In *ICML*, 547–556.

Chavira, M.; and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172(6): 772 – 799.

Dai, W.-Z.; Xu, Q.; Yu, Y.; and Zhou, Z.-H. 2019. Bridging Machine Learning and Logical Reasoning by Abductive Learning. In *NeurIPS*, 2815–2826.

Darwiche, A. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *IJCAI*, 819–826.

d'Avila Garcez, A. S.; Broda, K.; and Gabbay, D. M. 2002. *Neural-symbolic learning systems: foundations and applications*. Perspectives in neural computing. Springer.

Donadello, I.; Serafini, L.; and d'Avila Garcez, A. S. 2017. Logic Tensor Networks for Semantic Image Interpretation. *CoRR* abs/1705.08968.

Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; and Zhou, D. 2019. Neural Logic Machines. In *ICLR*.

Evans, R.; and Grefenstette, E. 2018. Learning Explanatory Rules from Noisy Data. *Journal of Artificial Intelligence Research* 61: 1–64.

Gaunt, A. L.; Brockschmidt, M.; Kushman, N.; and Tarlow, D. 2017. Differentiable Programs with Neural Libraries. In *ICML*, 1213–1222.

Hölldobler, S.; Störr, H.-P.; and Kalinke, Y. 1999. Approximating the Semantics of Logic Programs by Recurrent Neural Networks. *Applied Intelligence* 11: 45–58.

Kakas, A. C. 2017. Abduction. In Sammut, C.; and Webb, G. I., eds., *Encyclopedia of Machine Learning and Data Mining*, 1–8. Boston, MA: Springer US.

Kalyan, A.; Mohta, A.; Polozov, O.; Batra, D.; Jain, P.; and Gulwani, S. 2018. Neural-Guided Deductive Search for Real-Time Program Synthesis from Examples. In *ICLR*.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* 86(11): 2278–2324.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2018. DeepProbLog: Neural Probabilistic Logic Programming. In *NeurIPS*, 3749–3759.

Marra, G.; Giannini, F.; Diligenti, M.; and Gori, M. 2019. Integrating Learning and Reasoning with Deep Logic Models. *CoRR* abs/1901.04195.

Michael, L. 2019. Machine Coaching. In *IJCAI 2019 Workshop on Explainable Artificial Intelligence (XAI)*, 80–86.

Nuffelen, B. V.; and Kakas, A. 2001. A-system: Declarative Programming with Abduction. In *Logic Programming and Nonmotonic Reasoning*, 393–397.

Parisotto, E.; Mohamed, A.; Singh, R.; Li, L.; Zhou, D.; and Kohli, P. 2017. Neuro-Symbolic Program Synthesis. In *ICLR*.

Rocktäschel, T.; and Riedel, S. 2017. End-to-end Differentiable Proving. In *NIPS*.

Sadeghian, A.; Armandpour, M.; Ding, P.; and Wang, D. Z. 2019. DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs. In *NeurIPS*, 15321–15331.

Serafini, L.; and d'Avila Garcez, A. S. 2016. Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge. *CoRR* abs/1606.04422.

Sourek, G.; Aschenbrenner, V.; Zelezný, F.; and Kuzelka, O. 2015. Lifted Relational Neural Networks. *CoRR* abs/1508.05128.

Stallkamp, J.; Schlipsing, M.; Salmen, J.; and Igel, C. 2011. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, 1453–1460.

Sun, H.; Dhingra, B.; Zaheer, M.; Mazaitis, K.; Salakhutdinov, R.; and Cohen, W. 2018. Open Domain Question Answering Using Early Fusion of Knowledge Bases and Text. In *EMNLP*, 4231–4242.

Thoma, M. 2017. The HASYv2 dataset. *CoRR* abs/1701.08380.

Towell, G. G.; and Shavlik, J. W. 1994. Knowledge-based artificial neural networks. *Artificial Intelligence* 70(1): 119 – 165.

van Krieken, E.; Acar, E.; and van Harmelen, F. 2019. Semi-Supervised Learning using Differentiable Reasoning. *IF-CoLog Journal of Logic and its Applications* 6(4): 633–653.

Wang, P.; Donti, P. L.; Wilder, B.; and Kolter, J. Z. 2019. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*.

Xu, J.; Zhang, Z.; Friedman, T.; Liang, Y.; and Van den Broeck, G. 2018. A Semantic Loss Function for Deep Learning with Symbolic Knowledge. In *ICML*, 5502–5511.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *NeurIPS*, 2319–2328.

Yang, Z.; Ishay, A.; and Lee, J. 2020. NeurASP: Embracing Neural Networks into Answer Set Programming. In *IJCAI*, 1755–1762.