Neuro-Symbolic Artificial Intelligence: The State of the Art

Pascal Hitzler [a] and Md Kamruzzaman Sarker [b]
[a] *Kansas State University*
[b] *University of Hartford*

Contents

Chapter 7

Neuro-Symbolic AI = Neural + Logical + Probabilistic AI

Robin Manhaeve, KU Leuven, Dept. of Computer Science; Leuven.AI

Giuseppe Marra, KU Leuven, Dept. of Computer Science; Leuven.AI

Thomas Demeester, Ghent University - imec, Dept. of Information Technology

Sebastijan Dumančić, KU Leuven, Dept. of Computer Science; Leuven.AI

Angelika Kimmig, KU Leuven, Dept. of Computer Science; Leuven.AI

Luc De Raedt, KU Leuven, Dept. of Computer Science; Leuven.AI; Örebro University

[book title, editor(s), print ISBN or online ISBN, pages, year, and DOI or URL].

There is a broad consensus that both learning and reasoning are essential to achieve true (artificial) intelligence [1]. This explains why the quest for neuro-symbolic artificial intelligence (NeSy) [2, 3, 4, 5], which combines high-level reasoning with low-level perception, is high on the research agenda.

The two most prominent frameworks for reasoning are logic and probability. While in the past, they were studied by separate communities in artificial intelligence, a significant number of researchers has been working towards their integration, and aiming at combining probability with logic and statistical learning; cf. the areas of statistical relational artificial intelligence (StarAI) [6, 7] and probabilistic logic programming [8]. The reasoning abilities of statistical relational artificial intelligence approaches are complementary to the strong pattern-recognition abilities of deep learning.

Generally, neuro-symbolic systems integrate logic with neural networks. Probability theory has already been integrated with logic (cf. statistical relational AI) and neural networks. It therefore makes sense to consider the integration of logic, neural networks and probabilities. This effectively leads to an integration of probabilistic logics with neural networks and opens up new abilities. Furthermore, although at first sight, including

probability may appear as a complication, it actually greatly simplifies the integration of neural networks with logic. The reason for this is that probabilistic frameworks provide clear optimisation criteria, such as the likelihood of the training examples. Real-valued probabilistic quantities are also well-suited for gradient-based training procedures, as opposed to their discrete logic counterparts.

In this chapter, we first look at the three base paradigms (i.e. neural, logical and probabilistic methods) separately. Then, we look at the well established integrations, NeSy and StarAI. Next, we consider the integration of all three paradigms as Neural Probabilistic Logic Programming, and exemplify it with the DeepProbLog framework. Finally, we discuss the limitations of the state of the art, and consider future directions based on the parallels between StarAI and NeSy.

7.1. Base paradigms

We first consider the base elements of the integrations: neural, logical and probabilistic methods.

7.1.1. Logical methods

We now introduce the necessary background on the logical methods for this chapter. For more details, we refer to [9]. In this chapter, we focus on the subset of logic based on definite clauses. More formally, a *definite clause* is an expression of the form $h \leftarrow b_1 \wedge ... \wedge b_n$ where h and the b_i are logical atoms of the form $p(t_1,...,t_m)$, with p a predicate of arity m and the t_i terms. Terms then are either constants, logical variables or structured terms of the form $f(t_1,...,t_k)$ with f a functor and the t_j terms. Definite clauses for which $n = 0$ are called *facts*. By using the standard Prolog notation (i.e. commas for conjunctions and $:-$ for the implication), the following definite clause states that the alarm goes off if there is an earthquake:

```
alarm :- earthquake.
```

A logic program consists of a set of definite clauses. We show an encoding of the alarm example from [10] in Figure 1. In this example, your alarm can be triggered by either a burglar or an earthquake. If either of your neighbours, John and Mary, are home, the will call you if they hear the alarm. This program is *propositional* since every atom in the head or the body of the clauses has arity zero.

By introducing relations, we move from a propositional logic program to a first-order logic program. First-order programs are much more compact since definite clauses with variables behave as templates for many instantiations of the same clause. In Figure 2, we show a first order variant of the alarm program from Figure 1, where we compactly describe the behavior of multiple individuals (i.e. `mary` or `john`) using relations (i.e. `at_home`) and variables.

A substitution θ is an expression of the form $\{V_1 = t_1,...,V_n = t_n\}$ where the V_i are different variables and the t_i terms. Applying a substitution θ to an expression e (term or clause) yields the instantiated expression $e\theta$ where all variables V_i in e have been

```
# Facts
burglary.
earthquake.
at_home_mary.
at_home_john.

# Clauses
alarm :- earthquake.
alarm :- burglary.
calls_mary :- alarm,at_home_mary.
calls_john :- alarm,at_home_john.
```

Figure 1. A propositional logic program for the alarm example.

```
burglary.
earthquake.
at_home(mary).
at_home(john).

alarm :- earthquake.
alarm :- burglary.
calls(X) :- alarm,at_home(X).
```

Figure 2. A first-order program for the alarm example.

replaced by their corresponding terms t_i in e. For example, applying the substitution $\theta = \{X = \texttt{mary}\}$ to the term $g = \texttt{calls(X)}$ result in the substitution $g\theta = \texttt{calls(mary)}$.

The Herbrand base of a set of clauses is the set of all ground atoms that can be constructed using the predicates, functors and constants occurring in the clauses. Subsets of the Herbrand base are called Herbrand interpretations. A Herbrand interpretation is a model of a definite clause $h \leftarrow b_1 \wedge ... \wedge b_n$ if for every substitution θ such that $b_1\theta \wedge ... \wedge b_n\theta$ is in the interpretation, $h\theta$ is in the interpretation as well. An interpretation is a model of a definite clause program if it is a model of all its definite clauses. Finally, the semantics of a definite clause program is given by its unique Least Herbrand Model, i.e. the smallest model with respect to the set inclusion. This means that the Least Herbrand Model contains only the set of all ground facts that are logically entailed by the program.

Example 7.1.1 (Least Herbrand Model)**.** The Least Herbrand Model of the program in Figure 2 is:

$$\{\texttt{burglary}, \texttt{earthquake}, \texttt{at\_home(mary)}, \texttt{at\_home(john)},$$
$$\texttt{alarm}, \texttt{calls(mary)}, \texttt{calls(john)}\}$$

This definition of the semantics only holds for programs without negation. The negation of an atom in logic programming (denoted as $\backslash + p(t_1, ..., t_m)$) is defined as negation

as failure, meaning that the negation of an atom is true if the atom cannot be derived from the program. Programs with negation are not guaranteed to have a unique minimal Herbrand model. Several ways to define a canonical model have been studied (e.g. the well-founded semantics [11]).

7.1.2. Probabilistic methods

Probabilistic graphical models [12] are graphical models that compactly represent a (joint) probability distribution $P(X_1,...,X_n)$ over n discrete or continuous random variables $X_1,...,X_n$. The key idea is that the joint probability distribution factorizes over some factors f^i specified over subsets X^i of the variables $\{X_1,...,X_n\}$.

$$P(X_1,...,X_n) = \frac{1}{Z} f_1(X^1) \times ... \times f_k(X^k) \tag{1}$$

where Z is a normalization constant for P to represent a probability distribution.

The random variables correspond to the nodes in the graphical structure, and the factorization is determined by the edges in the graph. There are two main classes of probabilistic graphical models: Bayesian networks, where the underlying graph structure is a directed acyclic graph, and Markov random fields, where the graph is undirected. In this chapter, we will focus on Bayesian networks as we will establish several parallels with definite clause logic programs.

In a Bayesian network, we define a factor $f^i(X_i|parents(X_i))$ for each of the nodes X_i, where $parents(X_i)$ denotes the set of random variables that are a parent of X_i in the graph. The factors correspond to the conditional probability distributions $P(X_i|parents(X_i))$. In Figure 3 the *burglary alarm* Bayesian network [10] is shown. Here, for example, the random variable `alarm` is associated to the factor $P(\texttt{alarm}|\texttt{burglary},\texttt{earthquake})$. All the random variables are Boolean, i.e. they can take one of the values $\{true, false\}$ or $\{1,0\}$. For Bayesian networks, because one works with conditional probability distributions as factors, the joint distribution in Equation 1 is normalized and $Z = 1$. A Bayesian network represents a set of conditional independence assumptions: a variable is independent of the other variables that are not its descendants in the graph given its parents.

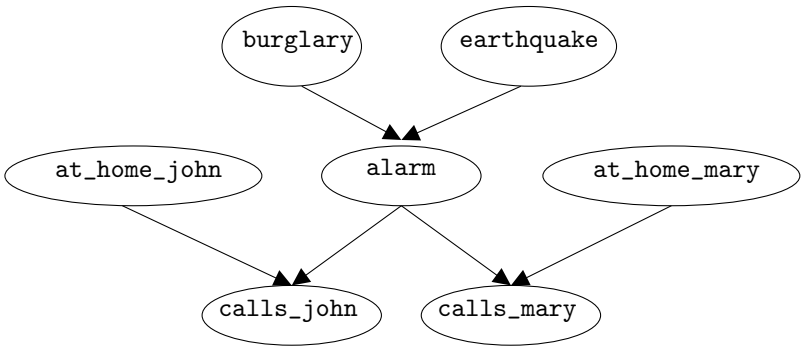

Figure 3. The alarm Bayesian network.

Table 1. Conditional probability distribution $P(\texttt{alarm}|\texttt{burglary},\texttt{earthquake})$ as a Conditional Probability Table, cf. Figure 3.

burglary	earthquake	$P(\texttt{alarm}\|\texttt{burglary},\texttt{earthquake})$
true	*true*	(0.95,0.05)
true	*false*	(0.95,0.05)
false	*true*	(0.29,0.71)
false	*false*	(0.001,0.999)

Probabilistic graphical models are an expressive formalism for learning and reasoning under uncertainty. However, inference in such models (e.g. computing specific marginal probabilities), is in general intractable as it is #P-complete. Nevertheless, there has been a lot of progress in devising various types of approximate and exact inference algorithms [13, 14, 15].

7.1.3. Neural methods

Neural networks [16] excel in machine learning tasks where the input data is high-dimensional and feature engineering is hard, for instance for analysing images, video or audio, or natural language. The key to their success lies in the combination of two principles. First, they are exceptionally good in automatically extracting features from raw inputs (e.g., pixel images, or textual data). To this end, neural networks learn multiple layers of nonlinear functions that project the input data onto relatively low-dimensional latent spaces. The latent representation of an input pattern is usually called an *embedding*. Second, the use of advanced gradient based optimization techniques [17] allows these nonlinear models to be efficiently trained.

In classification tasks, neural networks typically learn a conditional probability distribution $P(Y|X)$, where Y is a random variable representing the class and X is the input pattern. As an example, a neural network can model the probability $P(Y = 1|X = \blacksquare)$ that the MNIST image represents the digit 1. It is worth observing that such neural network models are quite similar to a conditional probability table, such as the one in Table 1, as they both represent conditional probability distributions over a set of random variables. However, it would be impractical to represent conditional distributions over complex inputs (e.g. images) using a table, as there would be a different row for any possible raw input, e.g., for every possible 28 x 28 gray scale image. Neural networks can thus be seen as compact approximations of such tables. The approximation is possible since it is assumed that, in high dimensional spaces, similar inputs have similar output distributions. This analogy allows for an intuitive understanding of the integration of neural networks with probabilistic reasoning models by replacing conditional probability tables by neural network building blocks (cf. Section 7.3.1).

Although deep learning has been successfully applied to a wide variety of domains, there is a growing awareness of the limitations of deep learning [18]. Deep learning requires large amounts of (the right kind of) data to train the network, it provides neither justifications nor explanations, and the models are black-boxes that can neither be understood nor modified by domain experts. Although there have been attempts to demonstrate reasoning-like behaviour with deep learning [19, 20, 21], their current reasoning abilities are nowhere close to what is possible with typical high-level reasoning approaches. Even when neural networks are used to perform reasoning, they can fail badly when they have to generalize to unseen reasoning patterns [22]. This failure is for example observed

when more reasoning steps are necessary during testing than during training. These types of generalizations are evaluated in datasets such as the CLUTRR dataset [23].

7.2. Integration

Let us now address the question as to how these different paradigms can be integrated in order to support both high-level reasoning and low-level perception.

We start by discussing how the field of Statistical Relational Artificial Intelligence (StarAI) [6] integrates the logic and probabilistic paradigms. This is then exemplified using the ProbLog [24] framework. Next, we will introduce how neuro-symbolic AI integrates sub-symbolic and symbolic systems.

7.2.1. Statistical Relational AI

First-order logic (programming), cf. Section 7.1.1, extends propositional logic by introducing relations. Bayesian networks and probability theory, cf. Section 7.1.2 can be seen as extending propositional logic with uncertainty. In fact, one can reason on the probability of a certain proposition given another one. Statistical Relational Artificial Intelligence (StarAI) brings the best of both worlds by extending first-order logic (programming) with uncertainty, or, equivalently, by extending probability theory with relations. One key idea, due to Poole [25] and Sato [26] is to unify the notion of a logical atom and a random variable.

StarAI models allow defining templated probabilistic models thanks to the use of first order clauses with variables. The parameters of these models can be trained on a specific set of individuals (e.g. `mary` and `john`)) and then they can be used to make predictions on new individuals (e.g. `bob`) . Indeed, one of the main properties of StarAI models is that they can be defined before knowing which individuals exist, by only reasoning about their potential relationships. Thus the same model can be applied to different populations and the learned regularities about the individuals in the training set can be transferred to new individuals. We will exemplify the domain of StarAI with ProbLog, a probabilistic extension of Prolog.

ProbLog ProbLog [24] lifts Prolog to a probabilistic model through the introduction of one concept: the probabilistic fact. Whereas a fact in Prolog is deterministically true, a probabilistic fact is of the form

$$p :: f$$

where f is a logical atom and p a probability. This allows us to encode the burglary alarm Bayesian network as a ProbLog program:

```
0.1::burglary.                  alarm:-earthquake.
0.5::at_home(mary).             alarm:-burglary.
0.2::earthquake.              calls(X):-alarm,at_home(X).
0.4::at_home(john).
```

A ProbLog program consists of a set of probabilistic facts $\mathscr{F}$ and a set of definite clauses $\mathscr{R}$. Each ground instance $f\theta$ of a probabilistic fact f corresponds to an *independent Boolean random variable* that is true with probability p and false with probability $1-p$. Let us denote the set of all ground instances of probabilistic facts in $\mathscr{F}$ as $\mathscr{F}\Theta$. Every subset $F \subseteq \mathscr{F}\Theta$ defines a possible world $w_F = F \cup \{h\theta | \mathscr{R} \cup F \models h\theta \text{ and } h\theta \text{ is ground}\}$, that is, the world w_F is the least Herbrand model of the logic program obtained by adding F to the set of clauses $\mathscr{R}$, i.e., the set of definite clauses. For instance

$$w_{\{\texttt{burglary},\texttt{at\_home(mary)}\}} = \{\texttt{burglary}, \texttt{at\_home(mary)}\} \cup \{\texttt{alarm}, \texttt{calls(mary)}\}$$

To keep the presentation simple, we focus on the case of finitely many ground probabilistic facts, but note that the semantics is also well-defined for the countably infinite case. The probability $P(w_F)$ of such a possible world w_F is given by the product of the probabilities of the truth values of the probabilistic facts:

$$P(w_F) = \prod_{f_i \in F} p_i \prod_{f_i \in \mathscr{F}\Theta \setminus F} (1-p_i) \tag{2}$$

For instance,

$$P(w_{\{\texttt{burglary},\texttt{at\_home(mary)}\}}) = 0.1 \times 0.5 \times (1-0.2) \times (1-0.4) = 0.024$$

The probability of a ground atom q, also called *success probability of* q, is then defined as the sum of the probabilities of all worlds containing q, i.e.,

$$P(q) = \sum_{F \subseteq \mathscr{F}\Theta : q \in w_F} P(w_F) \tag{3}$$

For ease of modeling, ProbLog supports non-ground probabilistic facts as a shortcut for introducing a set of ground probabilistic facts, as well as annotated disjunctions (ADs), which are expressions of the form

$$p_1 :: h_1 \ ; \ \ldots \ ; \ p_n :: h_n \ \text{:-} \ b_1, \ldots, b_m.$$

where the p_i are probabilities that sum to at most one, the h_i are atoms, and the b_j are literals. The meaning of an AD is that whenever all b_i hold, the AD causes one of the heads h_j to be true, or none of them with probability $1 - \sum p_i$. Note that several of the h_i may be true at the same time if they also appear as heads of other definite clauses or ADs. This is convenient to model choices between different categorical variables, e.g. different severities of the earthquake:

$$0.4 :: \texttt{no\_earthquake} \ ; \ 0.4 :: \texttt{mild\_earthquake} \ ; \ 0.2 :: \texttt{severe\_earthquake}.$$

or without explicitly representing the event of no earthquake:

$$0.4 :: \texttt{mild\_earthquake} \ ; \ 0.2 :: \texttt{severe\_earthquake}.$$

in which neither `mild_earthquake` nor `severe_earthquake` will be true with probability 0.4. Annotated disjunctions do not change the expressivity of ProbLog, as they

can alternatively be modeled through independent facts and definite clauses; cf. [8] for technical details.

To obtain some intuitions about the probabilistic logic program representation, it is instructive to see how they can represent Bayesian networks. Consider Bayesian networks involving only Boolean random variables. Each node without a parent then corresponds to a probabilistic fact. Observe that both nodes without any parents in a Bayesian network and probabilistic facts in ProbLog are marginally independent. Furthermore, each entry in a conditional probability table can be mapped onto an annotated disjunction. Assume the parents of the node n are x and y. Then there would be four annotated disjunctions of the form $p_i :: n :- x_v, y_v$, where x_v and y_v are the positive or negative literals corresponding to x, resp. y. This shows that annotated disjunctions can be used to specify conditional probability tables. For example, consider the node `alarm` in Figure 3 and its parents `burglary` and `earthquake`. The CPT in Table 1 can be encoded with the following ProbLog program:

```
0.95 :: alarm :- burglary, earthquake.
0.95 :: alarm :- burglary, \+ earthquake.
0.29 :: alarm :- \+ burglary, earthquake.
0.001 :: alarm :- \+ burglary, \+ earthquake.
```

7.2.2. Neuro-symbolic AI

We now discuss how neuro-symbolic AI (NeSy) systems differ from each other along two important aspects.

7.2.2.1. Two types of neuro-symbolic AI

Neuro-symbolic AI integrates neural networks with symbolic representations, often using symbolic logic. There are at least two ways of approaching such an integration. First, there are the neural networks approaches that use logic as a regularizer. Second, there are the logical approaches that are extended with neural constructs.

Neural networks with logical aspects. These methods extend neural network models with logical aspects. One way to do this is to define logical constraints on their output. These constraints are not hard constraints (i.e. they are not guaranteed to hold). Rather, they are only used during the training, where the degree in which these constraints are not satisfied serves as an additional loss term. Thus the logic acts as a kind of regularization function during the optimization. The effect is that the logic behaves as a kind of soft constraint on the outputs, which enforces the neural methods to make predictions that adhere better to the logic. This formalism effectively encodes the logic into the parameters of the network, so that even when the logic is not explicitly present, the model should still satisfy the logical constraints more than models that were trained without these constraints. Examples of such systems include Semantic Based Regularization [27] and the Semantic Loss function [28].

Another way to extend neural networks with logical aspects is to encode the logic into the structure. This can be done through a carefully designed architecture that can perform reasoning using common differentiable operations [20, 21, 22]. One can also use the logic to define the neural network structure in a templating approach [29, 30, 31].

Logical methods with neural constructs. Whereas the other line of work extends neural network with logic, this line of work tackles the integration from the other side. These methods extend logic-based frameworks with neural constructs. The neural constructs create an interface between the logic-based framework and the neural network. These constructs generally allow the logic to evaluate neural networks in a differentiable manner such that their parameters can be optimized together with any possible parameters in the logic. This type of extension similar to how Prolog was extended into the StarAI system ProbLog through the introduction of probabilistic facts. Example of systems in this line of work include DeepProbLog [32] and NeurASP [33].

7.2.2.2. Numbers vs Booleans

Neuro-symbolic AI systems need to connect the output of the neural networks to the Boolean truth values in logic. When we consider neural networks as classifiers, they output a confidence score for each class. If the neural network are sufficiently confident, it might suffice to only select the top prediction. However, in general, and especially when the neural network still needs to be trained, this is not feasible. There are several strategies for dealing with this.

One strategy is to choose one of the outputs of the neural network, based on the confidence score. When the choices turn out to be logically inconsistent, different outputs are chosen until they are consistent. These choices can then be used as pseudo-labels to train the neural network to make this output more likely. Systems that use this approach include ABL [34] and NGS [35].

Another strategy is to use the confidence scores directly in the logic. To do this, the logical operators are turned into real-valued functions, and hence, relax the Boolean truth values to the continuous $[0,1]$ interval. This introduces the semantics of *fuzzy logic* (or soft logic), which is mathematically grounded in the t-norm theory. For example, in the product t-norm, the truth degree of the conjunction, i.e. $t(a \wedge b)$, is equal to the product of the truth degrees of its arguments, i.e. $t(a \wedge b) = t(a)t(b)$. This approach is used in systems such as LRNN [30] and DiffLog [36].

Finally, it is also possible to interpret the confidence scores as a probability distribution, and to use a probabilistic logic to deal with the uncertainty. We will argue in the next section that this integration is very natural and has benefits over the use of a fuzzy logic. A probabilistic approach is used in systems such as DeepProbLog [32], NeurASP [33] and Semantic Loss [28].

The difference in how the probabilistic and the fuzzy methods relax the discrete logical semantics allows, in general, a computationally cheaper inference for the fuzzy approaches. However, this comes at the cost of a misalignment with the original Boolean theory, whose semantics is instead preserved by probabilistic approaches. In particular, in probabilistic methods, one defines a probability distribution over possible worlds or models. The probability of an atom or formula is computed as a sum over the probabilities of the worlds in which they are true. Their probability is as such defined only semantically and independent of the syntax. On the contrary, by relaxing the operators to continuous functions, fuzzy logics represent an alternative semantics and the truth degree of atoms and formulas may become a function of the syntax.

Example 7.2.1. Consider for example the following annotated program.

```
0.3::b.
a1 :- b.
a2 :- b,b.
```

Here, 0.3 is the label of b without any particular semantics yet. Given the idempotency of the Boolean conjunction, we would expect that the scores of both a1 and a2 to be identical since both b and $\mathtt{b} \wedge \mathtt{b}$ are true when b is true. In a probabilistic approach, the score is interpreted as a probability, i.e. $p(\mathtt{b}) = 0.3$. The probability of any atom is the sum of the probabilities of all the worlds where that atom is true. It is straightforward to see that, for both a1 and a2, they are the worlds where b is true, thus $p(\mathtt{a1}) = p(\mathtt{a2}) = p(\mathtt{b}) = 0.3$. On the other side, in the fuzzy settings, the score of b is interpreted as its truth degree, i.e. $t(\mathtt{b}) = 0.3$. Let us consider the product t-norm, where $t(x \wedge y) = t(x)t(y)$, then $t(\mathtt{a1}) = t(\mathtt{b}) = 0.3$ while $t(\mathtt{a2}) = t(\mathtt{b})t(\mathtt{b}) = 0.09$. While this issue could be solved by choosing a different t-norm (e.g. the minimum t-norm), similar issues arise due to the relaxed logical semantics.

7.3. Neural Probabilistic Logic Programming

Based on our experience in upgrading machine learning systems towards the use of (probabilistic) logical and relational representations [37, 38], we argue that *a first desirable property of frameworks that integrate two other frameworks A and B, is to have the original frameworks A and B as a special case of the integrated one*. If A or B cannot be fully reconstructed, one clearly loses certain abilities, which is not only undesirable but which also implies that there is no true unification. Applying this property to neuro-symbolic computation implies that the existing frameworks should have both the neural and the symbolic representations as special case. When this is the case, one retains both the learning abilities of the neural component as well as the reasoning and learning abilities and the semantics of the symbolic representations. Unfortunately, this property is not satisfied by the vast majority of neuro-symbolic approaches in that they either push the symbolic representation inside the neural network (from which the logic cannot be recovered), or vice versa, apply neural learning principles to symbolic representations (and risk loosing both the pure neural component and the logical semantics), as we shall show in the next section.

We also advocate a second desirable property for neuro-symbolic computation: models that learn from observed samples should be able to deal with uncertainty. Therefore, *one should not only integrate logic with neural networks in neuro-symbolic computation, but also probability*.

Although at first sight this may appear as a complication, it actually can greatly simplify the integration of neural networks with logic. The reason for this is that the probabilistic framework provides a clear optimisation criterion, namely the probability of the training examples. Real-valued probabilistic quantities are also well-suited for gradient-based training procedures, as opposed to discrete logic quantities.

We will now show how DeepProbLog achieves both of the desired properties by introducing the neural predicate as the interface between neural networks. Although the examples below are focused on reasoning with image classification, neural probabilistic

logic programming can be applied to a wide variety of domains, ranging from nature language reasoning to robotics.

7.3.1. DeepProbLog

DeepProbLog [32, 39] is a neuro-symbolic extension of ProbLog. In ProbLog, the probabilities of all random choices are explicitly specified as part of probabilistic facts or annotated disjunctions. DeepProbLog extends ProbLog to basic random choices whose probabilities are parameterized by neural networks. This is realized through the neural predicate, the interface between the probabilistic logic and the neural networks.

In [39], two types of neural predicates are introduced. We will only introduce the most general definition here, namely the neural annotated disjunction.

Definition 1 (Neural annotated disjunction)**.** *A neural annotated disjunction (nAD) is an expression of the form*

$$nn(m_r,[X_1,...,X_k],O,[y_1,...,y_n])::r(X_1,...,X_k,O)$$

where nn is a reserved functor, m_r uniquely identifies a neural network model (i.e., its architecture as well as its trainable parameters) that defines a probability distribution $p_{m_r}(O|X=x)$ over the domain $O \in \{y_1,...,y_n\}$ given the input $x=[x_1,..,x_k]$. The $X_1,...,X_k$ are variables representing the inputs to the neural network, O is the output variable, the ground terms $y_1,...,y_n$ define the domain of the output distribution, and r is a predicate symbol. Note that the arguments of predicate r can be in an arbitrary order.

Formally, such a neural AD represents a set of ground neural ADs of the following form, one for every sequence of ground terms $x_1,...,x_k$ representing inputs to the neural network:

$$\begin{aligned} &p_{m_r}(O=y_1|X_1=x_1,\ldots,X_k=x_k)::r(x_1,...,x_k,y_1)\ ;\\ &\ldots\ ;\\ &p_{m_r}(O=y_n|X_1=x_1,\ldots,X_k=x_k)::r(x_1,...,x_k,y_n) \end{aligned}$$

The neural network thus represents a discriminative classifier, which naturally maps onto an annotated disjunction for each input. For instance, in the MNIST addition example, we would specify the nAD

$$\texttt{nn}(\texttt{m\_digit},[\texttt{X}],\texttt{Y},[0,\ldots,9])::\texttt{digit}(\texttt{X},\texttt{Y}).$$

where `m_digit` is a network that classifies MNIST digits. For input image  , the ground nAD is

$$\texttt{p}_{\texttt{m\_digit}}(\texttt{Y}=0|\texttt{X}=\texttt{3})::\texttt{digit}(\texttt{3},0)\ ;\ \ldots\ ;\ \texttt{p}_{\texttt{m\_digit}}(\texttt{Y}=9|\texttt{X}=\texttt{3})::\texttt{digit}(\texttt{3},9).$$

The neural network could take any shape, e.g., a convolutional network for image encoding, a recurrent network for sequence encoding, etc. However, its output layer, which feeds the corresponding neural predicate, needs to be normalized.

In Section 7.1.3, we discussed how neural network classifiers naturally encode conditional probability distributions and that these could represent efficient approximations of conditional probability tables over high-dimensional inputs (like images). Moreover, in Section 7.2.1, we described a one-to-one mapping between ProbLog annotated disjunctions and entries of conditional probability tables in Bayesian networks. Neural annotated disjunctions in DeepProbLog exploit these two analogies to introduce neural network capabilities (i.e. image recognition, natural language processing) into (probabilistic) logic programming.

The semantics of a DeepProbLog program with respect to a fixed set of possible inputs to every neural network used in the neural facts and neural ADs is the semantics of the ProbLog program that replaces each neural fact and neural AD by the corresponding sets of ground neural facts and ground neural ADs for these inputs.

Inference in DeepProbLog We now briefly describe how inference happens in DeepProbLog. It is largely identical to inference in ProbLog and only needs to be extended with the evaluation of neural networks. ProbLog inference proceeds in four steps. The first step is the grounding step, in which the logic program is grounded with respect to the query. This step uses backward reasoning to determine which ground clauses are relevant to derive the truth value of the query, and may perform additional logical simplifications that do not affect the query's probability.

The second step rewrites the ground logic program into a formula in propositional logic that defines the truth value of the query in terms of the truth values of probabilistic facts. We can calculate the query success probability by performing *weighted model counting* (WMC) on this logic formula (cf. [40]). However, performing WMC on this logical formula directly is not efficient.

The third step is knowledge compilation [41]. During this step, the logic formula is transformed into a compiled structure that allows for efficient weighted model counting.

The fourth and final step transforms this compiled structure into an arithmetic circuit (AC). This is done by putting the probabilities of the probabilistic facts or their negations on the leaves, replacing the OR nodes with addition and the AND nodes by multiplication. The WMC is then calculated with an evaluation of the AC.

The only change required for DeepProbLog inference is that we need to instantiate nADs and neural facts to regular ADs and probabilistic facts. This is done in two steps. During grounding, we obtain ground nADs and ground neural facts with a symbolic representation of the probabilities. In a separate step after grounding, the concrete parameters are determined by making a forward pass on the relevant neural network with the ground input.

We exemplify inference with the following program that defines the addition of two MNIST digits.

```
nn(m_digit, [X], Y, [0...9]) :: digit(X,Y).
addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.
```

We now pose DeepProbLog the following query: `addition(`  `,`  `,1)`. First, the program is grounded on this query, which results in the following relevant ground program.

```
nn(m_digit,[ ],0)::digit( ,0);nn(m_digit,[ ], 1)::digit( ,1).
```

```
nn(m_digit,[ ,0)::digit( ,0);nn(m_digit,[ ], 1)::digit( ,1).
addition( , ,1) :- digit( ,0), digit( ,1).
addition( , ,1) :- digit( ,1), digit( ,0).
```

This ground program gets rewritten to the following formula : $(\texttt{digit}(,0) \wedge \texttt{digit}(,1)) \vee (\texttt{digit}(,1) \wedge \texttt{digit}(,0))$. This formally is then compiled and transformed into an arithmetic circuit, shown in Figure 4. The bottom of the diagram shows the neural network evaluation on the two images and . The output of these probabilities are then combined through a bottom-up evaluation of the AC to derive the probability of the query.

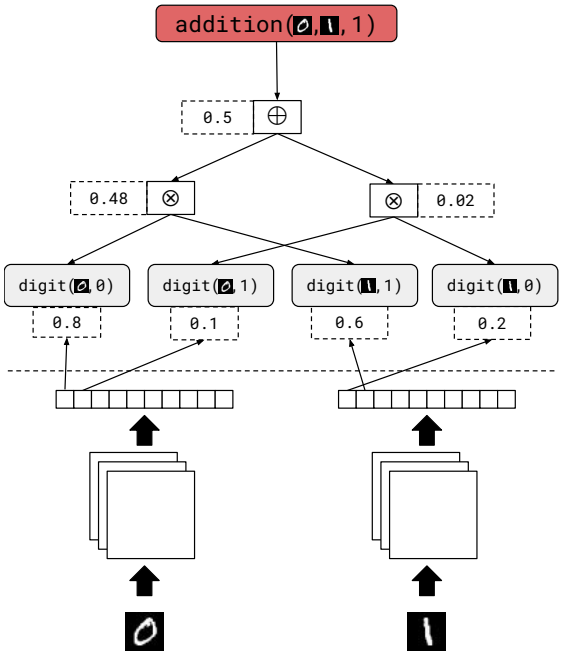

Figure 4. The arithmetic circuit for the MNIST addition example on the query `addition`(, ,1). Figure adapted from [39].

Learning in DeepProbLog In contrast to the earlier approach for ProbLog parameter learning in this setting by [42], DeepProbLog uses gradient based learning instead of expectation maximization. This allows for seamless integration with the training of neural network parameters. The key insight here is that the arithmetic circuit used for inference can be used for gradient computations as well. This AC is a differentiable structure, as it is composed of addition and multiplication operations.

DeepProbLog relies on the automatic differentiation capabilities of ProbLog to derive these gradients. More specifically, to compute the gradient with respect to the probabilistic logic program part, DeepProbLog uses aProbLog [43], a generalization of the ProbLog language and inference to arbitrary commutative semirings, including the gradient semiring [44]. Whereas ProbLog is confined to only calculating probabilities, the use of this gradient semiring in aProbLog allows the system to calculate the gradient

alongside the probabilities. This done by replacing the normal operations on probabilities in the arithmetic circuit:

$$p_1 \oplus p_2 = p_1 + p_2$$
$$p_1 \otimes p_2 = p_1 p_2$$

by simultaneous operations on probabilities and gradients, effectively calculating the gradients alongside the query probability.

$$(p_1, \nabla p_1) \oplus (p_2, \nabla p_2) = (p_1 + p_2, \nabla p_1 + \nabla p_2)$$
$$(p_1, \nabla p_1) \otimes (p_2, \nabla p_2) = (p_1 p_2, p_2 \nabla p_1 + p_1 \nabla p_2)$$

Preserving the base paradigms. We now show how the different base paradigms are preserved. ProbLog itself retains its base paradigms (probabilistic graphical models and logic). If no probabilistic facts are included in ProbLog, it essentially becomes Prolog and supports purely logical reasoning. As discussed in Section 7.2.1, probabilistic graphical models such as Bayesian networks can easily be represented in ProbLog. As DeepProbLog is an extension of ProbLog that adds the neural predicate, yet doesn't remove anything from this base framework, it becomes plain ProbLog when no neural predicates are present in the program. As such, it retains the same base paradigms as ProbLog.

The neural base paradigm is also preserved in DeepProbLog. For example, a classifier can be represented in DeepProbLog as a program with only a single neural predicate. Other neural methods such as embedding based methods (e.g. TransE, DistMult, ...) can also be represented as shown below in Example 3.

7.3.2. DeepProbLog examples

We will now use the DeepProbLog framework to showcase several desirable properties of neuro-symbolic integrations.

Example 1: Logical reasoning with DeepProbLog To show that DeepProbLog supports both logical reasoning and deep learning, we extend the classic learning task on the MNIST dataset [45] to a more complex problem that requires reasoning. Instead of using labeled single digits, we train on pairs of images, labeled with the sum of the individual labels. The DeepProbLog program consists of the clause

$$\texttt{addition(X,Y,Z)} :- \texttt{digit(X,X2)}, \texttt{digit(Y,Y2)}, \texttt{Z is X2} + \texttt{Y2}$$

and a neural AD for the `digit/2` predicate, which classifies an MNIST image. We compare to a CNN baseline classifying the two images into the 19 possible sums. Results shown in Table 2 indicate that performing reasoning and learning at the same time outperforms methods that can only learn, namely a neural baseline.

Example 2: Probabilistic reasoning with DeepProbLog To show the benefit of integrating the probabilistic paradigm, we consider an extension to the previous experiment. The extension is that a fraction of the labels is replaced by a number uniformly selected between 0 and 18. If we model this label noise, DeepProbLog can reason probabilistically about each experiment whether its label is correct or not, and train the neural network

Table 2. The accuracy of a neural baseline and DeepProbLog on the addition task for different number of training examples.

	Number of training examples		
Model	30 000	3 000	300
Neural Baseline	93.46 ± 0.49	78.32 ± 2.14	23.64 ± 1.75
DeepProbLog	97.20 ± 0.45	92.18 ± 1.57	67.19 ± 25.05

Table 3. The accuracy on the noiseless test set for MNIST addition experiment trained with label noise.

	Fraction of noisy labels					
	0.0	0.2	0.4	0.6	0.8	1.0
Baseline	93.46	87.85	82.49	52.67	8.79	5.87
DeepProbLog	97.20	95.78	94.50	92.90	46.42	0.88
DeepProbLog w/ explicit noise	96.64	95.96	95.58	94.12	73.22	2.92
p(`noisy`)	0.000	0.212	0.415	0.618	0.803	0.985

accordingly. In the code below, we have a probabilistic fact `noisy`. This fact is used to split the `addition/3` predicate into two cases. If `noisy` is false, then the addition happens as in Example 1. If it's true, then we assume that `Z` is uniformly sampled between 0 and 19. The probability of the `noisy` fact is a learnable parameter, which allows the model to estimate the fraction of label noise in the training data.

```
nn(classifier, [X], Y, [0 ..9]) :: digit(X,Y).
t(0.2) :: noisy.

1/19 :: uniform(X,Y,0) ; ...; 1/19 :: uniform(X,Y,18).

addition(X,Y,Z) :- noisy, uniform(X,Y,Z).
addition(X,Y,Z) :- \+noisy, digit(X,N1), digit(Y,N2), Z is N1+N2.
```

We compare three models: a CNN baseline, the DeepProbLog model from Example 1, and the DeepProbLog model shown above. Table 3 shows the final accuracy on the noiseless test set. For the DeepProbLog model that explicitly models the noise, we also report the value of this parameter, which indicates how much noise the DeepProbLog model believes there is. As we can see, probabilistic reasoning makes DeepProbLog robust and also gives it the ability to detect the amount of label noise in the dataset.

Example 3: DeepProbLog as a neuro-symbolic programming language In this experiment, we showcase that DeepProbLog is a neuro-symbolic programming language by manipulating embeddings in a declarative, yet differentiable way. A straightforward way to obtain image embeddings, is by optimizing a clustering objective. That is, we minimize the distance between the embeddings of images of the same number, while maximizing the distance between the embedding of images of different numbers. As a similarity metric, the radial basis function $\varphi(x,y)$ (RBF) can be used (cf. the neural theorem prover [31]). This can be included in DeepProbLog as a predicate `rbf/2` that succeeds with probability $p = \varphi(x,y) = e^{-||x-y||_2}$ where x and y are embeddings that are the ar-

guments of the predicate. This can be used to implement a predicate that determines the similarity between two images:

```
similar(I1,I2) :-
    %Encode images I1 and I2 into E1 and E2
    cnn_encode(I1,E1), cnn_encode(I2,E2),
    rbf(E1,E2).
```

Here, `cnn_encode/2` unifies the second argument with an embedding of the first argument calculated by a convolutional neural network. The `rbf/2` acts as a constraint that leads to clustering in embedding space.

As a proof of concept of the wider possibilities of implicitly training embeddings through DeepProbLog, we extend the clustering objective towards inducing an order relation in embedding space. The successor relationship between two MNIST images is defined as `successor(I1,I2,R)` where `I1` and `I2` are MNIST images, and `R` is the difference of the digits represented by the two images (e.g. `successor(`  `,`  `,`-2`)`). We model the successor relationship as a translation (cf. TransE [46]). We learn an embedding r_s of the successor relationship in this embedding space such that we minimize the distance between $e_x + nr_s$ and e_y, where e_x and e_y are the embeddings of the two images and n is the difference of the image labels.

```
successor(I1,I2,N) :-
    %Encode images I1 and I2 into E1 and E2
    cnn_encode(I1,E1), cnn_encode(I2,E2),
    %Embed the successor relation into embedding S
    embed(successor, S),
    %The relation should not be trivial (i.e. 0)
    \+rbf(S,0).
    %E = E1 + N*S should be similar to E2
    mul(S,N,S2), add(E1,S2,E),
    rbf(E,E2).
```

If $n = 0$, then the `successor/3` relationship becomes identical to the `similarity/2` relationship. We also specify that r_s should not be zero to avoid collapsing on a trivial solution. Note that this constraint on the size of the embedding can be seen as a form of regularization defined in the logic. We define a predicate `embed/2` that unifies the second argument with a learnable embedding for the first argument. `mul/3` is the product between a scalar and a vector and `add/3` an element-wise addition. These create the mathematical definition of the successor relationship. The number 0 is used as a zero vector of the same dimension as the embeddings.

Figure 5 shows the embeddings of the MNIST test set (crosses) and the mean of each of these images, grouped by label (solid dots). It shows that the model has learned to embed images into clusters corresponding to the labels of these images, which are positioned sequentially along the embedding dimension. As can be seen, the cluster centers can be (approximately) linearly mapped to the actual image labels.

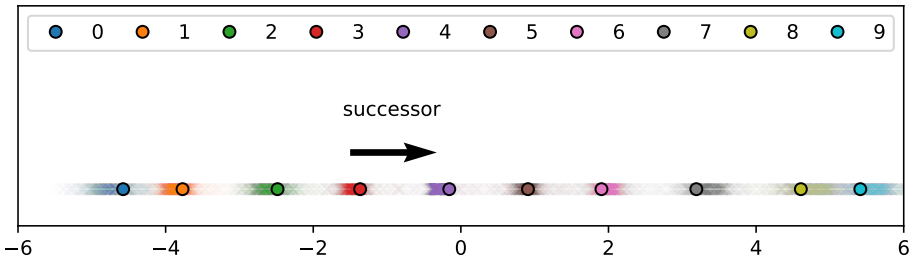

Figure 5. A visualization of the embedding space learned in Example 3. Individual MNIST image embedding are shown in crosses, with the mean for each label as a solid dot. The direction of the successor relation is marked by the arrow.

7.4. Conclusions and open challenges

In this chapter, we discussed the base paradigms that make up the vast majority of neuro-symbolic systems. We then discussed the similarities and differences of two paradigms that integrate reasoning and learning, namely statistical relational AI and neuro-symbolic AI. Next, we discussed the desirable properties of neuro-symbolic systems. Firstly, a framework that is the integration of two frameworks should retain either framework as a special case. Otherwise, certain abilities have been lost in the integration. Secondly, neuro-symbolic systems should integrate a logical, neural and a probabilistic base paradigm. We have showcased this for the neural probabilistic logic programming paradigm with the DeepProbLog framework.

However, there are still open challenges that need to be solved by the NeSy community. One challenge to overcome is the scalability of neuro-symbolic systems. Especially the inference of systems that have a separate logical inference step can be prohibitively expensive. Including probabilistic inference can make this problem even worse. Although some progress has been made in this direction by investigating approximate inference, there is not yet a solution that can be applied to arbitrary settings. Furthermore, a lot of effort has gone into making the training of (large) neural networks as efficient as possible, maximally utilizing the parallellization capabilities of hardware. The same considerations will have to be made for neuro-symbolic systems if they are to be made universally applicable.

Another open challenge is performing structure learning in neuro-symbolic systems. Although some progress has been made in this direction, the possibilities of these systems are still not on the level of structure learning seen in pure logic programming / StarAI. Introducing neural networks into the structure learning problem further complicates this setting as these neural networks might also have to be trained themselves.

A final challenge directly related to the previous challenges is that the field is not yet mature enough to be applied to a showcase application. This is partly due to the limited scalability of certain neuro-symbolic methods, but also due to the fact that many systems assume that the logical structure is given by the user. However, this is not viable for all areas of application, and a robust system that can discover (part of) the logical structure would be essential.

References

[1] Kahneman D. Thinking, fast and slow. Farrar, Straus and Giroux New York; 2011.

[2] Garcez ASd, Broda KB, Gabbay DM. Neural-symbolic learning systems: foundations and applications. Springer Science & Business Media; 2012.

[3] Garcez Ad, Besold TR, De Raedt L, et al. Neural-symbolic learning and reasoning: contributions and challenges. In: 2015 AAAI Spring Symposium Series; 2015.

[4] Hammer B, Hitzler P. Perspectives of neural-symbolic integration. Vol. 8. Springer Heidelberg:; 2007.

[5] De Raedt L, Dumancic S, Manhaeve R, et al. From statistical relational to neuro-symbolic artificial intelligence. In: IJCAI; 2020.

[6] De Raedt L, Kersting K, Natarajan S, et al. Statistical relational artificial intelligence: Logic, probability, and computation. Synthesis Lectures on Artificial Intelligence and Machine Learning. 2016;10(2):1–189.

[7] Getoor L, Taskar B. Introduction to statistical relational learning. MIT press; 2007.

[8] De Raedt L, Kimmig A. Probabilistic (logic) programming concepts. Machine Learning. 2015;100(1):5–47.

[9] Flach P. Simply logical: intelligent reasoning by example. John Wiley & Sons, Inc.; 1994.

[10] Pearl J. Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc.; 1988.

[11] Van Gelder A, Ross KA, Schlipf JS. The well-founded semantics for general logic programs. Journal of the ACM. 1991;38(3):620–650.

[12] Koller D, Friedman N. Probabilistic graphical models: principles and techniques. MIT press; 2009.

[13] Robert CP, Casella G. Monte carlo statistical methods. Springer; 2004. Springer Texts in Statistics.

[14] Wainwright MJ, Jordan MI. Graphical models, exponential families, and variational inference. Now Publishers Inc; 2008.

[15] Darwiche A. Modeling and reasoning with bayesian networks. Cambridge university press; 2009.

[16] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press; 2016. `http://www.deeplearningbook.org`.

[17] Kingma DP, Ba J. Adam: A method for stochastic optimization. In: International Conference on Learning Representations (ICLR); 2015. p. 1–13.

[18] Marcus G. Deep learning: A critical appraisal. arXiv preprint arXiv:180100631. 2018;.

[19] Santoro A, Raposo D, Barrett DG, et al. A simple neural network module for relational reasoning. In: NIPS; 2017.

[20] Makni B, Hendler JA. Deep learning for noise-tolerant RDFS reasoning. Semantic Web. 2019; 10(5):823–862.

[21] Ebrahimi M, Sarker MK, Bianchi F, et al. Neuro-symbolic deductive reasoning for cross-knowledge graph entailment. In: Martin A, Hinkelmann K, Fill H, et al., editors. Proceedings of the AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021), Stanford University, Palo Alto, California, USA, March 22-24, 2021; (CEUR Workshop Proceedings; Vol. 2846). CEUR-WS.org; 2021.

[22] Ebrahimi M, Eberhart A, Hitzler P. On the capabilities of pointer networks for deep deductive reasoning. CoRR. 2021;abs/2106.09225. Available from: `https://arxiv.org/abs/2106.09225`.

[23] Sinha K, Sodhani S, Dong J, et al. Clutrr: A diagnostic benchmark for inductive reasoning from text. Empirical Methods of Natural Language Processing (EMNLP). 2019;.

[24] De Raedt L, Kimmig A, Toivonen H. ProbLog: A probabilistic Prolog and its application in link discovery. In: IJCAI; 2007. p. 2462–2467.

[25] Poole D. Probabilistic horn abduction and bayesian networks. Artif Intell. 1993;64(1):81–129. Available from: `https://doi.org/10.1016/0004-3702(93)90061-F`.

[26] Sato T. A statistical learning method for logic programs with distribution semantics. In: Sterling L, editor. Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995. MIT Press; 1995. p. 715–729.

[27] Diligenti M, Gori M, Sacca C. Semantic-based regularization for learning and inference. Artificial Intelligence. 2017;244:143–165.

[28] Xu J, Zhang Z, Friedman T, et al. A semantic loss function for deep learning with symbolic knowledge. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018; 2018. p. 5498–5507.

[29] Andreas J, Rohrbach M, Darrell T, et al. Neural module networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2016. p. 39–48.

[30] Sourek G, Aschenbrenner V, Zelezny F, et al. Lifted relational neural networks: Efficient learning of latent relational structures. Journal of Artificial Intelligence Research. 2018;62:69–100.

[31] Rocktäschel T, Riedel S. End-to-end differentiable proving. In: Advances in Neural Information Processing Systems; Vol. 30; 2017. p. 3788–3800.

[32] Manhaeve R, Dumancic S, Kimmig A, et al. Deepproblog: Neural probabilistic logic programming. In: NeurIPS; 2018.

[33] Yang Z, Ishay A, Lee J. Neurasp: Embracing neural networks into answer set programming. In: Bessiere C, editor. Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020. ijcai.org; 2020. p. 1755–1762. Available from: https://doi.org/10.24963/ijcai.2020/243.

[34] Dai WZ, Xu Q, Yu Y, et al. Bridging machine learning and logical reasoning by abductive learning. In: Wallach H, Larochelle H, Beygelzimer A, et al., editors. Advances in Neural Information Processing Systems; Vol. 32. Curran Associates, Inc.; 2019. Available from: https://proceedings.neurips.cc/paper/2019/file/9c19a2aa1d84e04b0bd4bc888792bd1e-Paper.pdf.

[35] Li Q, Huang S, Hong Y, et al. Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event; (Proceedings of Machine Learning Research; Vol. 119). PMLR; 2020. p. 5884–5894. Available from: http://proceedings.mlr.press/v119/li20f.html.

[36] Si X, Raghothaman M, Heo K, et al. Synthesizing datalog programs using numerical relaxation. In: IJCAI; 2019. Available from: https://doi.org/10.24963/ijcai.2019/847.

[37] De Raedt L. Logical and relational learning. Springer; 2008.

[38] Muggleton S, De Raedt L, Poole D, et al. Ilp turns 20. Machine learning. 2012;86(1):3–23.

[39] Manhaeve R, Dumančić S, Kimmig A, et al. Neural Probabilistic Logic Programming in DeepProbLog. Artificial Intelligence. 2021;298:103504. Available from: https://www.sciencedirect.com/science/article/pii/S0004370221000552.

[40] Fierens D, Van den Broeck G, Renkens J, et al. Inference and learning in probabilistic logic programs using weighted Boolean formulas. Theory and Practice of Logic Programming. 2015;15(3):358–401.

[41] Darwiche A, Marquis P. A knowledge compilation map. Journal of Artificial Intelligence Research. 2002;17:229–264.

[42] Gutmann B, Kimmig A, Kersting K, et al. Parameter learning in probabilistic databases: A least squares approach. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases; Springer; 2008. p. 473–488.

[43] Kimmig A, Van den Broeck G, De Raedt L. An algebraic Prolog for reasoning about possible worlds. In: AAAI; 2011.

[44] Eisner J. Parameter estimation for probabilistic finite-state transducers. In: Proceedings of the 40th annual meeting on Association for Computational Linguistics; Association for Computational Linguistics; 2002. p. 1–8.

[45] Lecun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. In: Proceedings of the IEEE; 1998. p. 2278–2324.

[46] Bordes A, Usunier N, Garcia-Duran A, et al. Translating embeddings for modeling multi-relational data. In: Advances in neural information processing systems; 2013. p. 2787–2795.