Neuro-Symbolic Learning

Patrik Begáň[*]
Masaryk University, FI
540478@mail.muni.cz

Lucile Charpentier[†]
Masaryk University, CST FI
543072@mail.muni.cz

Adam Bodnár[‡]
Masaryk University, FI
485613@mail.muni.cz

Toon Lemmens[§]
Masaryk University, CST PriF
532525@mail.muni.cz

Michal Barnišin[¶]
Masaryk University, FI
485135@mail.muni.cz

Marek Toma[‖]
Masaryk University, FI
485275@mail.muni.cz

Abstract

Neuro-symbolic learning combines the benefits of neural networks with the strengths and verifiable safety of symbolic engines. In this article, we present an overview of neuro-symbolic systems, including their taxonomy and applications. We focus in detail on the three selected neuro-symbolic approaches: *DeepProbLog*, *REVEL* and *Learning on knowledge graphs*.

1 Introduction

Deep learning has revolutionized the field of machine learning and has proven to be effective in many applications, such as game playing or natural language processing. However, it has been criticized for its susceptibility to opposing attacks, lack of explainability, and the need for large amounts of data. According to award-winner and machine learning theory pioneer Leslie Valiant, one of the key challenges for computer science is combining reasoning and learning. Neuro-symbolic AI tries to tackle this issue by connecting neural models with symbolic reasoning [9, 11].

Artificial neural networks (ANNs) have been around since the 1940s; the models were developed to simulate the behaviour of neurons in the brain. Over the years, ANNs have become increasingly sophisticated and are now widely used in, for example, image recognition, natural language processing or autonomous vehicles. Neuro-symbolic learning, on the other hand, is a more recent development that combines the strengths of ANNs and symbolic reasoning. It seeks to create a hybrid system that can reason about the world symbolically while also being able to learn from data using ANNs [13].

The idea of combining symbolic reasoning and machine learning has been around since the 1980s, and it has only recently become feasible due to advances in computing power and machine learning techniques. Neuro-symbolic systems are being developed to tackle complex tasks such as robotic manipulation, natural language understanding and scientific discovery. While ANNs have been the dominant approach in machine learning for many years, neuro-symbolic learning represents a promising new direction that combines the best of both worlds [13].

[*] Chapter *Why combining neural networks with symbolic reasoning*, co-authored chapter *Representation Learning on Knowledge Graphs*(section 4.3.2 and 4.3.3).

[†] Co-authored chapter *DeepProgLog*.

[‡] Chapter *Challenges & Current Research*, co-authored chapter *Neuro-symbolic Reinforcement Learning with Formally Verified Exploration*.

[§] Chapter *Introduction*, co-authored chapter *DeepProgLog*.

[¶] Chapter *Forms of Integration*, co-authored chapter *Neuro-symbolic Reinforcement Learning with Formally Verified Exploration*.

[‖] Chapter *Applications*, introduction of *Representation Learning on Knowledge Graphs*, and section *Ontologies*.

Neuro-symbolic artificial intelligence is a combinatorial approach that integrates low-level perception with high-level reasoning. Neural networks and deep learning entail the low-level perception part as they successfully learn and optimize. However, neural networks tend to have difficulties with extrapolation, explainability, and goal-directed reasoning, which are provided by the high-level reasoning part, namely probabilistic-logical approaches. They introduce knowledge as constraints for learning, enabling a more directed process. In addition, probabilistic-logical modelling and reasoning are complemented by neural networks as, in contrast, they are less flexible and sensitive to noisy data. Hence, neural-symbolic integration allows the designing of neural architectures representing differential counterparts of symbolic operations in classic reasoning tools. Neuro-symbolic artificial intelligence tackles multiple issues, such as language translation, computer games, and protein folding [9, 11].

Studying neuro-symbolic AI is motivated by two key reasons. First, it is applied as a tool to advance the understanding of the human brain in cognitive science. Artificial neural networks can be considered an abstraction of the physical workings of the brain. At the same time, formal logic is an abstraction of what is perceived through self-reflections and cognitive reasoning. Both aspects are an inherent part of the human brain; therefore, relating or unifying them can help discover their functioning. The second reason was already quoted before. Both approaches complement each other. Deep learning can deal with raw data and are robust against outliers and data errors, while symbolic systems are more sensitive. However, symbolic systems can use expert knowledge and are self-explanatory, as humans can understand their algorithms in detail. Combining both approaches enables them to overcome the weaknesses they face individually [9, 11].

2 Why combining neural networks with symbolic reasoning

Facing the current developments in deep learning, one might pose a question: *Why is its combination with symbolic reasoning beneficial?*

Even modern thinkers outside of the field of AI acknowledge the potential of combining neural networks with symbolic reasoning. During the AAAI-2020 debate, psychologist and Nobel prize winner Daniel Kahneman used abstract notions of human decision-making System 1 and System 2 from his book *Thinking Fast and Slow*[7] as an analogy to neural-symbolic learning where these systems should resemble neural and symbolic parts consequently, stating that System 1 knows language and System 2 involves manipulation with symbols [9].

Deep learning employs neural networks and stochastic gradient descent to explore a continuous space for solving classification or regression tasks while generating vector-based, distributed representations instead of logical or symbolic ones. On the other hand, symbolic machine learning involves the manipulation of symbols as part of a discrete search process to identify the optimal representation for solving a given classification or regression task. Decision trees are a typical example of symbolic machine learning, although more complex forms of representation exist, such as relational representations using first-order or higher-order logic. Although current neural networks can represent propositional logic, nonmonotonic logic programming, propositional modal logic and fragments of first-order logic, they fail with the full first-order or higher-order logic. It is also difficult for neural networks alone to achieve true relational learning, e.g. learning relation *ancestor* from a few examples of *mother*, *father*, and *grandparent*. In this case, rather than extracting a whole knowledge base to learn a given relation, one can be more efficient with learning relations by jumping to conclusions from a relatively small number of examples and further using similarity measures to infer new relations. At the same time, symbolic descriptions can be derived and used for reasoning beyond the distribution of the data [9].

[7] In this book, Kahneman introduces a concept of 2 systems taking part in human reasoning. The dominant System 1 makes fast decisions based on heuristics and previous experiences as opposed to System 2, which allocates much more cognitive resources to reason about specific decisions (a reader can imagine confronting with two mathematical formulas: $2 + 2$ and $35 * 47$. Whereas the answer to the 1st formula comes to mind intuitively without the need to do any calculations, the second formula requires more time and effort to calculate and thus engaging System 2).

Many modern AI applications have identified the need for a well-founded knowledge representation and reasoning to be integrated with deep learning and the need for sound explainability. Unlike the blackbox behaviour of neural networks, the explainability of symbolic approaches is prominent. That is because of a clear hierarchy of semantics and language expressiveness. Knowledge-base is also more explainable than a neural network since it provides a trace (a proof history) that elucidates why an outcome was obtained, in contrast to the showing of how the propagation of activation through the network resulted in that outcome [9].

Another good reason for combining neural networks with symbolic reasoning is that neural networks alone offer only approximate solutions in a continuous space. In contrast, reasoning can be both approximate and precise in a discrete space. Once a complex neural network has learned the function $f(x) = x$, it is desirable for the following calculation to be precise and to extrapolate well to any value of x. A similar case is also in learning a general set of rules, constraints and exceptions to the rules. Learning systems may have difficulty adopting universal quantification over variables such as $\forall x P(x)$, in which case the learning system would theoretically have to be exposed to all instances of x to learn such a statement. To efficiently learn formulas such as $\forall x P(x)$, the learning system must be able to extrapolate it given a sufficient amount of evidence (number of instances of x in case of $\forall x P(x)$), as well as revise its conclusions over time in the presence of new evidence. Problems solved by AI applications often require modelling cause and effect using implicit information. This requires learning general rules and exceptions to the rules evolving over time. In such cases, deep learning is inadequate in handling cases where examples beyond the training data distribution are presented. Symbolism can provide additional knowledge in the form of constraints for learning, which also diminishes neural networks' notorious problems with forgetting or difficulty with extrapolation in unbounded domains or outlier data. [9]

To sum up this section, we highlight an idea of an ideal neuro-symbolic application proposed by [9]. Such an application should be able to derive information at different levels (apart from what can be perceived from the data), such as complex concept learning, where more straightforward concepts are systematically organised to create a definition for the higher concept. Conceptual structures yet to be discovered using data require knowledge governed by general rules and exceptions to the rules. That allows good generalisation as well as handling specific cases. According to the Turing award winner Leslie Valiant, the critical challenge for Computer Science nowadays is the scrupulous combination of reasoning and learning, building significant semantics and powerful representation language for intelligent cognitive behaviour.

3 Forms of Integration

Among current neuro-symbolic systems, a variety of forms of cooperation between neural (statistical) and symbolic (rule-based) subsystems can be found. In this chapter, we present a taxonomy of the neuro-symbolic systems, adapted from [9] and [14][8], based on how the neural and symbolic parts communicate, and how they can be used to achieve the desired goal.[9]

- TYPE I (or **Symbolic Neuro Symbolic**) can be generally characterized as a weak-cooperation between the neural and the symbolic engine. TYPE I systems use a symbolic representation of inputs (i.e. tokens), which are then converted into vectors which are then fed into the neural network. Finally, the outputs of the neural part are interpreted as a probability distribution on the output tokens.

[8] Although both papers contain the same taxonomy by *H. A. Kautz*, we will be using the one presented in [9], as it seems clearer. However, for completeness, we will also mention the alternative naming.

[9] Images were taken from [18].

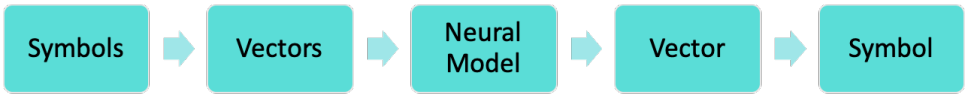


Figure 1: The higher-level architecture of TYPE I neuro-symbolic system. The input consists of (encoded) symbols, e.g. words, which are converted into vectors. The neural network performs computation on the vectors and the result is translated back to symbolic representation.

This is currently used mostly in language models (for example in question answering or text translation), where the input consists of tokenized sentences, which are then embedded into a high dimensional vector space and the outputs of the model are used to guide a search algorithm (e.g. *beam search*) to construct the system's response.

- TYPE II (or **Symbolic[Neuro]**) architecture uses neural network subroutine with a symbolic problem solver.

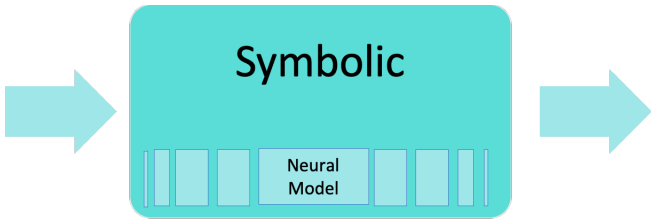


Figure 2: TYPE II systems contain a symbolic solver with a neural model, which is used as either the input processor or as a substitution for demanding full state search. In this case, the neural network serves as a supporting subsystem.

One of the best-known examples of this type of neuro-symbolic system is *Alpha Zero*, developed in [29]. It uses a general reinforcement algorithm employing *Monte-Carlo Tree Search* guided by the neural network. It benefits both from fast, trained positional evaluation and a tree search to focus on exploring the most promising moves [29]. This type can be also found in most robots or autonomous vehicles [14].

- TYPE III (or **Neuro | Symbolic**) is a system where neural part runs in parallel with the symbolic engine, each focusing on a particular subtask, while communicating and reusing outputs of one another.

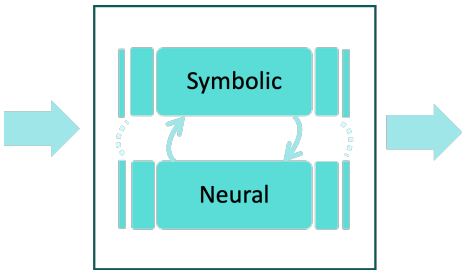


Figure 3: In TYPE III systems, the symbolic module and the neural module work concurrently. The neural module converts non-symbolic inputs into a symbolic data structures, and the symbolic module performs the computation on it [14]. These parts are equally important, as opposed to TYPE II systems, where the neural network works as a subroutine.

Examples of these systems include *neuro-symbolic concept learner* [21], which uses a neural network to convert images into inputs for the symbolic reasoning engine; or *DeepProbLog* [20] (see also chapter 4.1).

- TYPE IV (or **Neuro: Symbolic $\rightarrow$ Neuro**) are systems using symbolic knowledge to construct or enrich the training set, or to initialize and design the architecture of the neural module.

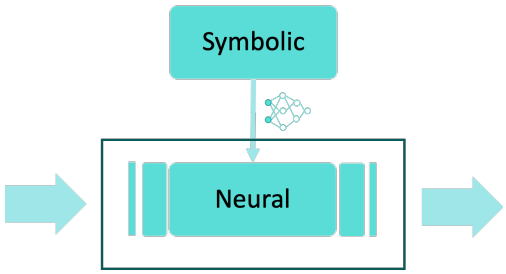


Figure 4: In TYPE IV, the symbolic module is used to create new training data, modify the training routine or to help design the network's architecture.

This approach was used e.g. in [4] to convert mathematical formula trees into recursive tree-LSTMs.

- TYPE V (or **Neuro_{Symbolic}**) consists of systems that map symbolic rules (such as first order logic (FOL), including objects, relations and predicates [18]) into tensors via embeddings.

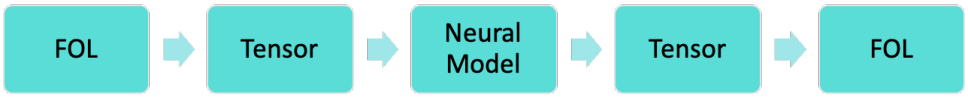


Figure 5: In TYPE V systems, the symbolic representation of first-order logic is mapped to tensors which become the input to the neural module. The outputs are mapped back to FOL. The difference with TYPE I is that the whole logic is being encoded as opposed to mere symbols. [18]

Examples of these systems include *Logic Tensor Networks* (see [6]).

- TYPE VI (or **Neuro[Symbolic]**) is a fully-integrated neuro-symbolic system, where the neural part is capable of using the symbolic engine autonomously.

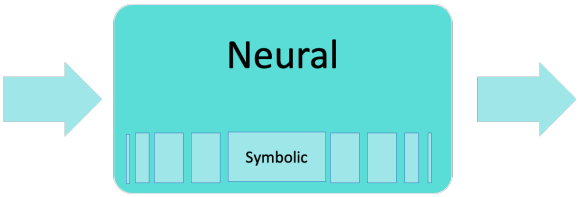


Figure 6: TYPE VI systems is based on the *System 1* and *System 2* from [9]. The neural module represents the fast, decision-making system, which can either give fast but not very precise answers, or it can invoke the symbolic subsystem, which is slower but offers higher precision. It is therefore the responsibility of the network to decide when to use the symbolic engine and to prepare and process its in(out)puts.

Systems like *HuggingGPT* [28][10] or *ChatGPT* with *Wolfram Alpha* plugin [32] are essentially large language models, which are capable of querying other systems and reusing their outputs to complete their task.

4 Selected Approaches

In this chapter, we present 3 selected state-of-the-art papers on combining symbolic computation with neural networks. Our choice of papers from different areas of machine learning highlights that many areas can benefit from incorporating symbolic subsystems.

4.1 DeepProgLog

DeepProbLog can be linked to the field of neuro-symbolic learning, which aims to integrate neural networks with symbolic reasoning to enable more powerful and flexible AI systems. The incorporation

[10] *HuggingGPT* based on the *ChatGPT* [24], at the time of the writing, can access other neural models via structured queries, which can be viewed as using symbolic black-box APIs.

of neural predicates in DeepProbLog allows for the integration of deep learning and probabilistic-logical reasoning, which are two essential components of neuro-symbolic learning. The ability of DeepProbLog to support both symbolic and sub-symbolic representations and inference, as well as program induction, makes it a promising tool for exploring and advancing the field of neuro-symbolic learning.

DeepProbLog or Neural Probabilistic Logic Programming starts with an existing probabilistic programming language, here ProbLog, and connects it to a neural network capable of processing logic functions or statements. According to Henry Kautz's taxonomy for neuro-symbolic AI [14], it is classified as a Type 3 system. Probabilistic logic expressions of the form $q(t_1, \ldots, t_n)$ where q is a predicate and t_i, its terms, have a probability p. That being, the output components of a neural network can be employed to construct "neural" predicates provided that the neural network's output on the expression can be interpreted as a probability. For example, the predicate `addition(X,Y,Z)` where `X` and `Y` are images of digits and `Z` a natural number representing the sum of `X` and `Y`. Although this predicate can be processed by a standard neural classifier where it not only learns how to interpret the images but also how to add them, it will have a hard time taking into account logic background knowledge and the definition of addition. In contrast, this logic knowledge consisting of logic rules is easily incorporated by DeepProbLog. Now, the neural network is only responsible for interpreting the digits, introducing new capabilities and significantly improving performance. In addition, the classifier can be extended to multi-digit numbers without additional training.

4.1.1 Logic Programming Concepts

To understand the DeepProbLog programming and, by extension, the DeepProbLog model, basic knowledge concerning ProbLog is required. First, a ProbLog program consists of a set of ground probabilistic facts F of the form $p :: f$ where p is the probability of an expression $q(t_1, \ldots, t_n)$ and a set of rules R. Each probabilistic fact represents an independent Boolean random variable with a True probability of p and a False probability of $1 - p$. Secondly, an annotated disjunction (AD) is a probabilistic fact of the form $p_1 :: h_1; \ldots; p_n :: h_n$:- $b_1, \ldots, b_m$. where p_i are probabilities that sum up to 1, h_i and b_i are expressions. When b_i is valid, h_i will be true with a probability of p_i. In DeepProbLog programming, these AD are extended with a set of neural ADs (nAD) that look like

$$nn(m_q, \vec{t}, \vec{u}) :: q(\vec{t}, u_1); \ldots; q(\vec{t}, u_n) \text{ :- } b_1, \ldots, b_m$$

where the b_i are atoms, $\vec{t} = t1, \ldots, t_k k$ is a vector term representing the inputs of the neural network for predicate q, u_1 to u_n are the possible output values of the neural network. N_n indicates using a neural network and m_q the neural network model specifying a probability distribution over its output values $\vec{u}$, given input $\vec{t}$.

4.1.2 DeepProbLog Model

The DeepProbLog learning pipeline will be explained with an example and is visualised in figure 7 . A program written in the DeepProbLog language is the basis of the machine learning experiment. This program is the training example and can consist of neural or non-neural annotated disjunctions, probabilistic facts and rules explaining the query or goal.

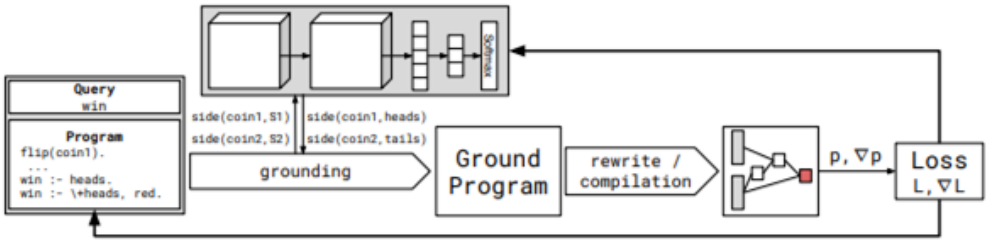


Figure 7: The example of the DeepProbLog program. Probabilistic facts are in red, the nAD and AD in green and the rules in blue.

4.1.3 The Learning Pipeline

The aim of the example is to calculate the probability of winning the query. A win is accomplished if either the ball is red or at least one coin comes up heads. The program consists of 6 parameters: the first four originate from the neural expression, the heads and tails of the first and second coin. The other two are logic parameters representing the chance to pick a red or blue ball. So the machine learning model will use these six parameters to learn the query. The neural network will learn the interpretation of coin1 and coin2, while the logic part takes care of the red-blue ball distribution and the final win probability.

```
flip(coin1). flip(coin2).
nn(m_side,C,[heads,tails])::side(C,heads);side(C,tails).
t(0.5)::red;t(0.5)::blue.
heads :- flip(X), side(X,heads).
win :- heads.
win :- \+heads, red.
query(win).
```

4.1.4 The DeepProbLog Program

The DeepProbLog Inference is divided into four steps that are conducted for each training example. First, the logic program is grounded, meaning all instances of clauses the query depends on are grounded. In case a neural predicate is encountered, the required inputs, here pictures of the coins, are sent to the neural network. The resulting scores are generated by the SoftMax output layer and used as the probabilities of the ground AD. Secondly, the ground logic program is rewritten into a formula that defines the true value of the query in terms of the truth values of the probabilistic facts (can we make that formula ourselves?). Subsequently, the formula is compiled into a Sentential Decision Diagram (SDD), which allows efficient query evaluation. The SDD of the example is shown in figure 8, where the red rectangle portrays the query, the grey rectangles represent the variables of the model that correspond to the probabilistic facts and the white rectangles are the applied logic operators. During the fourth step, the SDD is evaluated bottom-up, and the success probability of the query p and its gradient or derivative ∇p is calculated. These two parameters will be used to compute the loss function.

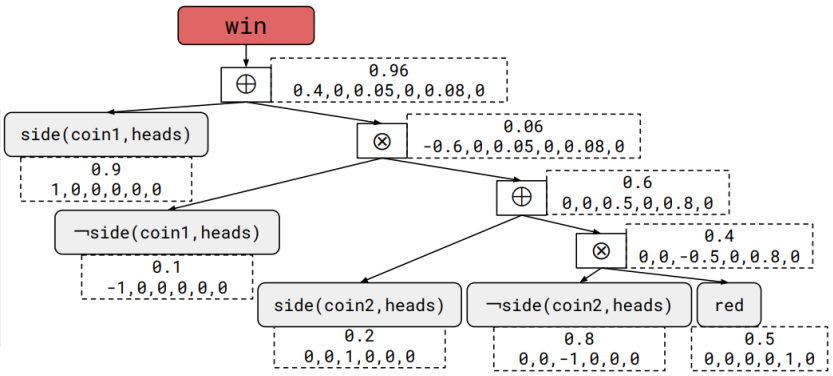


Figure 8: The SDD for the example program. The results are shown in the white boxes near the red, grey and white rectangles.

4.1.5 SDD for Query Win

The calculations of p and ∇p for the higher rectangles of the SSD depend on which logic operator combines the lower rectangles. Two operations are distinguished, addition $\oplus$ and multiplication $\otimes$. To calculate a higher p value, the lower values are simply added or multiplied according to the operator. In contrast, ∇p calculation requires more complex calculation as ∇p represents the partial derivative $\frac{\partial p}{\partial x}$ of the probability p with respect to parameter x, that is, the probability p_i of the probabilistic fact with learnable probability, written as $t(p_i) :: f_i$. This can be extended to a vector of parameters

$\vec{x} = [x_1, \ldots, x_N]^T$, representing all N parameters in the ground program ($N = 6$, in the example). The addition and multiplication of these vectors happen as follows:

$$(a_1, \vec{a_2}) \oplus (b_1, \vec{b_2}) = (a_1 + b_1, \vec{a_2} + \vec{b_2})$$
$$(a_1, \vec{a_2}) \otimes (b_1, \vec{b_2}) = (a_1 b_1, b_1 \vec{a_2} + a_1 \vec{b_2})$$

The parameters of the probabilistic facts and neural networks $\vec{x}$ are trained together by learning from entailment. For a DeepProbLog program with parameters X, a set Q of pairs (q, p) with q a query, p the desired success probability of the query and a loss function L, the learning from entailment formula is as follows:

$$\underset{\vec{x}}{argmin} \frac{1}{|Q|} \sum_{(q,p) \in Q} L(P_{\chi=\vec{x}}(q), p)$$

The parameter learning employs gradient descent, which allows the integration of logic parameter learning with neural network training. The parameters of probabilistic facts and ADs are simply updated based on the updated gradients. In contrast, the neural predicates form the act as an interface between the logic and neural side., with both sides treating each other as a black box. The logic side can calculate the gradient of the loss with respect to the neural network's output but is unaware of its internal parameters. Though, the gradient with respect to the output is sufficient to start backpropagation, which calculates the gradient for the internal parameters of the neural network. In the next step, gradient-based optimisers are applied to update the parameters of the network.

4.1.6 Experimental Evaluation

The experimental evaluation shows that DeepProbLog is capable of program induction, as shown in the second set of problems. The approach is similar to differentiable Forth, where neural networks fill the gaps in provided programs. Three tasks are considered: addition, sorting, and word algebra problems. And DeepProbLog supports logical reasoning and deep learning by expanding the MNIST dataset task to two more intricate reasoning problems.

1. *Logical reasoning and deep learning*
   Extending the classical learning task on the MNIST dataset to two more complex problems requiring T1 and T2 reasoning. To show that DeeProbLog supports logical reasoning and deep learning.

   The results plotted in figure 9, comparing DeepProbLog and a convolutional neural network (CNN), show that DeepProbLog, which combines symbolic and subsymbolic reasoning, achieves a higher F1 score than the CNN after a few thousand iterations, although the CNN converges more slowly. An alternative for the CNN baseline was also tested, but it remains slower and less accurate than DeepProbLog. On T2, DeepProbLog achieves slightly lower accuracy than the single-digit problem due to the error accumulation effect. Still, unlike CNN, which does not generalise to this variable length problem, the model generalises well.

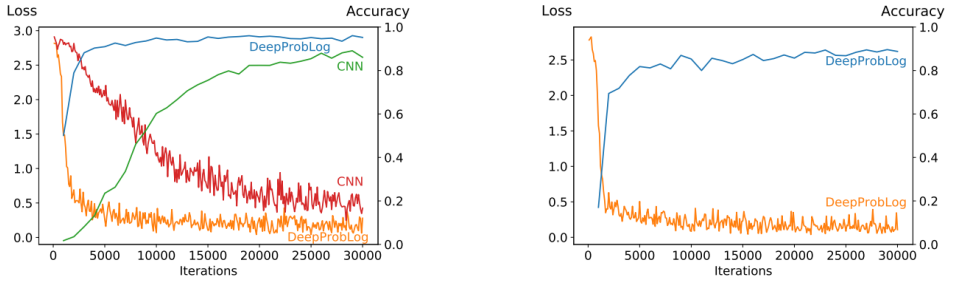


Figure 9: Comparison between DeepProbLog and a Convolutional neural network, on the addition of numbers with single digits (left) and two digits (right).

2. *Program induction*
   This section shows that DeepProbLog can perform program induction by following the differentiable Forth program sketch, where holes in the given programs must be filled by neural networks trained on input-output examples for the entire program. We consider the three tasks: addition T3, sorting T4, and word algebra problems (WAP) T5.

For Forth language addition and sorting problems, DeepProbLog achieves 100% accuracy, while the differentiable Forth language has computational complexity problems with longer learning lengths. On WAP problems, DeepProbLog achieves similar accuracy to Bosnjak et al. [2017] (96% to 97%). For the Forth language sorting problem, DeepProbLog generalises better to longer training lengths than the differentiable Forth language, which shows poor results starting at a training length of 4. Furthermore, DeepProbLog runs faster and scales better with increasing training length, as shown in Table 1.

Training length	2	3	4	5	6
$\partial 4$ on GPU	42s	160s	-	-	-
$\partial 4$ on CPU	61s	390s	-	-	-
DeepProbLog on CPU	11s	14s	32s	114s	245s

Table 1: Time until 100% accurate on test length 8 for the sorting (T4) problem

3. *Probabilistic programming and deep learning*
   Here belongs the coin problem, which is a standard example in the probabilistic programming community. The game involves tossing a potentially biased coin and taking a ball from two urns containing red, blue and green balls. The goal is to determine the bias of the coin and the ratio of coloured balls in each urn. Task T6 involves learning the game/4 predicate using a combination of symbolic, subsymbolic and probabilistic reasoning. The data is presented as RGB triples and trained simultaneously with two neural networks. The problem uses a simple neural network component, and the training converges quickly with 100% accuracy on the test set.
4. *Implementation details*
   Technical details of experiments conducted to solve various problems using neural networks and probabilistic programming. The experiments include MNIST image and colour classification, solving the coin and Forth language problem, and the Coin-Urn experiment. The networks used have specific architectures, such as convolutional and fully connected layers, with specific hyperparameters, such as learning rate and Gaussian noise. Cross-entropy loss is used to optimise the predicted and desired query probabilities. Tools used include PyTorch and ProbLog2.

Neural networks and reasoning are often combined in the neuro-symbolic reasoning literature, but most approaches are limited in their ability to support probabilistic reasoning and cognition. DeepProbLog takes a different approach, integrating neural networks into a probabilistic logic framework. Related work has focused on developing differentiable frameworks for logical reasoning, including theorem proving and compiling logic programs into differentiable functions. Another work area is incorporating background knowledge into training as a regularisation term. It also combines perception with reasoning, but DeepProbLog handles uncertainty through probabilistic reasoning. Finally, the work develops probabilistic logic formulations of fundamental deep learning primitives.

4.1.7 Conclusion

DeepProbLog integrates neural networks and probabilistic logic programming to their strengths and is always trainable by example. It extends ProbLog by adding neural predicates and uses ProbLog for training to compute loss gradients. Using standard gradient descent methods, this optimises parameters for probabilistic logic programs and neural networks. The experiments show that DeepProbLog excels in combinatorial symbolic and subsymbolic reasoning, program induction, and probabilistic logic programming.

4.2 Neuro-symbolic Reinforcement Learning with Formally Verified Exploration

Reinforcement learning (RL) is a machine-learning task where an intelligent agent(s) learn to take optimal action[11] to navigate through an environment to maximise accumulated reward. Ensuring that, during the exploration of the environment, the agent's actions can be considered safe is a fundamental

[11]The mapping from the states to the selected action is also called *a policy*. It is the policy that the agents are trying to optimize, as it is used to decide the next action of the agent in the environment.

problem. Most approaches define safety constraints and a high threshold required to satisfy these constraints. However, in some real-world applications, such as autonomous robotics, any unsafe action poses a financial risk and a hazard to humans. [3]

Previous methods offer safety for worst-case inputs by constructing a space of provably safe policies. These have to be defined before the learning process. During the learning, a monitoring system evaluates all actions and detects violations of safety. If such action is detected, the system replaces it with one of the safety policies. One of the drawbacks is that the safe space has to be constructed beforehand, as the monitoring system does not have information about the internal state of the agent during learning. This can lead to policies being too strict and, consequently, deteriorate the agent's performance by forbidding (potentially) rewarding safe actions. Furthermore, these methods have only been used on finite state-action spaces [3]. Complex settings, such as continuous action spaces propose a much harder challenge.

This paper[12] improves on the previous algorithms in all mentioned deficiencies. The REVEL[13] framework supports learning over the continuous state and action spaces, as well as updating the monitoring system during the learning process. Built on the system PROPEL[14], REVEL in addition to the neural network, maintains symbolic forms to represent safe policies. Instead of verifying the neural network for safety – which can be computationally expensive – the framework verifies the less demanding symbolic representation of the policies. The process of creating efficient, safe and not overly strict policies is done via repeated application of applying operations LIFT, UPDATE and PROJECT.

4.2.1 Learning Algorithm

The proposed learning method is based on a functional mirror descent in policy space, based on approximate gradients. The algorithm uses two classes of policies, symbolic easily verifiable for safe policies and a neuro-symbolic. These are then combined using an oracle for proving the safety of the safe policy into the resulting policy:

$$NextAction(state) := \text{if } IsSafe(NN, state) \text{ then } NN(state) \text{ else } SafePolicy(state)$$

that is if the selected action by the neural network is safe (this decision is made by the oracle) then it is applied, otherwise the safe policy is used to determine next action.

For the computation to be feasible, the class of symbolic safe policies consist of deterministic piece-wise linear policies.

Algorithm 1 Reinforcement Learning with Formally Verified Exploration

Input: Initial symbolic safe policy g
for $t = 1, \cdots, T$ **do**
  $f \leftarrow \text{LIFT}(g)$
  $f \leftarrow \text{UPDATE}(g, f)$
  $g \leftarrow \text{PROJECT}(f)$
end for
return Policy g

The algorithm repeats the following steps:

- **LIFT** Using imitation learning, a neural network f is trained to imitate the safe policy g (lifted into the symbolic space).
- **UPDATE** This consists of several gradient updates to both policies, the safe policy g and the imitating neural network policy f, using approximate gradients of f.

[12] Referring to [3].
[13] REinforcement learning with VErified expLoration – REVEL.
[14] See [31].

- **PROJECT** Compute a symbolic policy that is the closest[15] to the policy obtained from the combination of the oracle, g and f, e.g. using cutting planes sampling and adding components to the linear policy g.

4.2.2 Experiments

The empirical evaluation focuses on two aspects; how much safer the REVEL policies are, compared to the state-of-the-art RL approaches and the performance decrease connected to the achieved safety—whether REVEL does have a performance advantage over approaches using static policies.

To evaluate the performance and safety improvements, the authors of the paper chose to compare the proposed algorithm REVEL to modified[16] existing approaches for safe reinforcement learning, namely Deep Deterministic Policy Gradients (DDPG) [19] , Constrained Policy Optimisation (CPO) [1] and a (static) variant of REVEL without policy updates. The experiments used 10 classic RL benchmarks[17] including control problems and robotic applications.

In terms of performance, the comparison between the cost of policies learned using REVEL against other approaches shows that [3]:

- The performance of REVEL is competitive with (or even better than) DDPG for 7 out of the 10 benchmarks. REVEL achieves significantly better reward than DDPG in the *car-racing* benchmark, and the reward is only slightly worse for 2 benchmarks.
- REVEL has better performance than CPO on 4 out of the 10 benchmarks and only performs slightly worse on 2. Furthermore, the cost incurred by CPO is significantly worse on 2 benchmarks (*noisy-road* and *car-racing*).
- REVEL outperforms the static approach on 4 out of 10 benchmarks. Furthermore, the difference is very substantial on two of these benchmarks (*noisy-road* and *mountain-car*).

Following table (2) shows the average number of safety violations per run for DDPG and CPO. Both approaches perform safety violations in 8 out of 10 benchmarks.

Benchmarks	DDPG	CPO
mountain-car	0	3.6
road	0	0
road-2d	113.4	70.8
noisy-road	1130.4	8526.4
noisy-road-2d	107.4	0
obstacle	12.4	1
obstacle2	96	118.6
pendulum	92.4	9906
acc	4	673
car-racing	4956.2	22.4

Table 2: Safety violations of DDPG and CPO [3].

For a qualitative assessment of the learned policies, we present the learned trajectories for the game *acc*[18] from the original paper, see figure 10. While both, DDPG and CPO agents failed and caused the crash, REVEL agent successfully learned a safe policy.

[15]Here by closest, we mean that it minimizes the Bergman divergence of both policies.

[16]Modified by inducing negative rewards for safety violations.

[17]*mountain-car, road, road-2d, noisy-road, noisy-road-2d, obstacle, obstacle2, pendulum, acc, car-racing*

[18]The game *acc* (Adaptive cruise control) is a simple game of one agent that is responsible for accelerating the car or using the brakes to decelerate. The goal of the agent is to follow another car as closely as possible without causing a crash. The leading car can change its speed at any time.

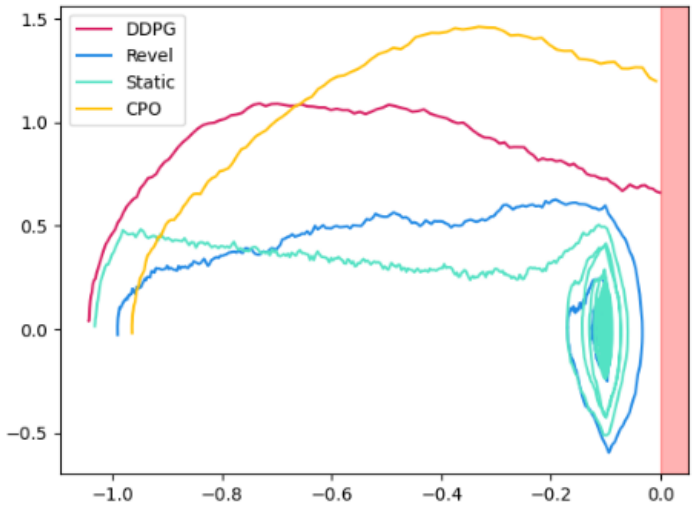


Figure 10: The trajectories of the learned agents and the leading car for the game *acc*. The x-axis shows the relative position, and the y-axis is the relative speed. Note that the red region represents the situations, in which the follower overtook the leader, thus causing a crash and failing the task. (*Static* represents a REVEL-type policy without the updates of the monitoring system.) [3]

4.2.3 Conclusion

The REVEL framework solves the safety in reinforcement learning over continuous state-action spaces, by using a verification-friendly symbolic representation of the policy. It was shown, that despite being more computationally demanding, the REVEL algorithm significantly reduces the number of safety violations during the training process.

One limitation of this work is the assumption of a fixed worst-case model for the environment and the necessity to handcraft a strong enough initial safe policy to improve training performance.

4.3 Representation Learning on Knowledge Graphs

A knowledge graph (KG) is a knowledge base with a graph structure, that consists of entities or concepts linked together with relations. The goal of representation learning on KGs is to learn vector embeddings of nodes or relations, usually for a downstream task of link prediction, node classification, or entity resolution. However, knowledge graphs are oftentimes incomplete, especially if they are created by combining multiple smaller knowledge graphs. One possible solution to this problem is the use of prior knowledge to construct logical rules that can be used to infer relations and associations. This might include relationships between classes and, subsequently, their instances. To this end, we review a neuro-symbolic approach that combines neural embedding learning and logical inference by automated reasoning over ontologies.

4.3.1 Ontologies

An ontology specifies a conceptualization [10]. Meaning it may specify standard identifiers, definitions, and representations of classes (sometimes called categories), properties, annotations of classes, individual entities, and relationships between them. Importantly it also provides axioms and formal definitions (usually based on subsets of predicate logic, e.g. description logic), which allow automated and computational access to the meaning of classes and relationships and deductive inference to derive new associations or relationships. [12]

Ontologies represent quite a sophisticated knowledge organization system. However, they are expensive to create and maintain, limiting their development and subsequent use width. Despite that,

there are several larger ontologies, namely in the biomedical domain, e.g. Gene Ontology [5], Human Phenotype Ontology [17], and Disease Ontology [16].

Automated Reasoning

Automated reasoning deals with the creation of algorithms that are able to reason and infer new knowledge with respect to some formal languages. In the context of ontologies, an example of such language might be Web Ontology Language, which offers a variety of reasoning algorithms over it, which are able to compute taxonomies, check the consistency of given ontology, and check the satisfiability of concepts.

In their paper, Alshahrani et al. [2] used OWL 2 EL profile (sub-language) to represent the knowledge graph they created. OWL 2 EL supports several axioms and class restrictions, some of which are:

- class restrictions - existential quantification ($\exists m.A$), intersection of classes ($A \cap B$), etc.
- axioms - class inclusion ($A \subseteq B$), class equivalence ($A \equiv B$), property equivalence ($m \equiv n$), transitive and reflexive object properties

This graph representation was then used to deductively close the knowledge graph with respect to the OWL 2 EL profile with polynomial time ELK reasoner [15]; KG is deductively closed if $\forall \phi(KG \models \phi \Rightarrow \phi \in KG)$. They then added the obtained inference as edges to the original knowledge graph. Where such an inferred graph can be further used as input to an embedding learning task.

4.3.2 DeepWalk

DeepWalk [25] is a novel unsupervised feature learning algorithm that generalises neural language models to process a particular language composed of a set of randomly-generated walks on a graph. The resulting latent representations of the graph nodes can be used for various downstream tasks, including link prediction and community detection. One key innovation of DeepWalk is its use of random walks as a language-like structure for learning representations of the nodes in a graph. In particular, each random walk is analogous to a short sentence. Given all the previous vertices visited so far, the likelihood of observing a specific vertex is estimated using a similar approach to that used in traditional language models:

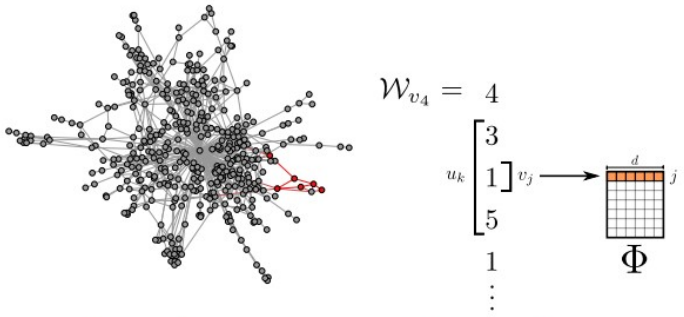
$$Pr\Big(v_i|\big(\Phi(v_1), \Phi(v_2), \ldots, \Phi(v_{i-1})\big)\Big)$$

However, this approach is time and resource-consuming as the walk length grows. Recent relaxation in language models can be utilised in the context of random walks to generate more informative and robust representations. Instead of using context to predict a missing word, it uses one word to predict context. The context is also derived using words to the left side, the same as to the right side of the particular word. And finally, instead of ordering constraint, the model is required to maximise the probability of any word appearing in the context. Incorporating these relaxations in the vertex optimisation modelling, the optimisation problem now looks like this:

$$\underset{\Phi}{minimize} - logPr\big(\{v_{i-w}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+w}\}|\Phi(v_i)\big)$$

DeepWalk takes a graph as input and produces a latent representation as output. The likelihood of observing each vertex in the graph is transformed into a latent representation suitable for statistical modelling using a mapping function Φ. The process of generating a random walk and transforming vertices with a mapping function is illustrated in figure 11. The resulting representations are learned through a process of deep learning. The algorithm consists of two main components: a random walk generator and an update procedure. The random walk generator generates a set of random walks on the graph, which are then used to train the update procedure to produce valuable representations of the graph's nodes.

One of the strengths of DeepWalk is its ability to be adjusted to suit specific graph structures. The algorithm is flexible enough to be adapted to different types of graphs, such as directed, weighted, or bipartite graphs, by incorporating appropriate modifications to the random walk generator or the update procedure. This adaptability makes DeepWalk a versatile tool that can be customised for various graph-based applications. Moreover, experiments have shown that the performance of



(a) Random walk generation. (b) Representation mapping.

Figure 11: DeepWalk illustration. A window of size $2w + 1$ is moved across the random walk W_{v4} that maps the central vertex v_1 to its representation $\Phi(v_1)$. The representation $\Phi(v_1)$ is then adjusted to optimize the likelihood of v_1 appearing together with its context vertices (v_3, v_5). Source: [25]

DeepWalk can be significantly improved by considering the graph's specific structure, demonstrating its ability to generate tailored representations optimised for specific tasks.

A modified version of the DeepWalk algorithm has been used to generate corpus C, consisting of a set of edge-labelled random walks, in the neural-symbolic approach for structured biological data [2]. Since the generated knowledge graphs have different edge types, edge-labelled random walks are adopted. The extended algorithm outputs a set of such random walks from the input knowledge graph $G = (V, E)$, where the parameters of the algorithm, such as the *length of the walks* and the *number of walks per node*, can be specified to optimise the performance of the model.

Learning Embedings and Prediction

Corpus C, an output of DeepWalk (consisting of edge-labelled random walks), is used as an input for generating edge embeddings. Embedding is a vector representation of the word s from corpus C. In [2], the skip-gram model [22] was used to learn and generate the embeddings. Such embeddings can be further used for machine learning tasks such as edge prediction. The objective is to calculate the likelihood of the existence of an edge with a specific label l between two vertices v_1 and v_2, based on their vector representation. They used the logistic regression classifier implemented in the `sklearn` library for the edge prediction task. Separate binary prediction models are created for each object property in the knowledge graph. For model building and testing, 5-fold cross-validation was employed. Cross-validation folds are created by removing 20% of edges with label l for each object property representing this label. Following is applying deductive inference, corpus generation through edge-labelled random walks, learning of node vector representations, and building a binary logistic regression model.

4.3.3 Summary

Authors of [2] first build a knowledge graph using Semantic Web technologies, which is then deductively closed by the Elk reasoner. This knowledge graph is further used as an input to the algorithm that can learn representations of nodes through repeatedly performing random walks. Final random walks represent sentences in a corpus, which is an input to the Word2Vec skip-gram model that can learn embeddings for each node. Those steps are automated in an algorithm illustrated in figure 12. Further machine learning tasks can be applied to the resulting embeddings.

This neuro-symbolic learning approach did not demonstrate a significant performance increase over the state-of-the-art in predicting certain biological relations. However, the embeddings produced by this algorithm can be incorporated as additional features into pre-existing machine-learning techniques without requiring substantial manual effort for feature extraction and representation. Moreover, this approach can take advantage of structured data and ontologies to perform automated reasoning to encode associations between biological entities, including their ontology-based classifications. It can do so even when these associations are not explicitly stated but inferred.

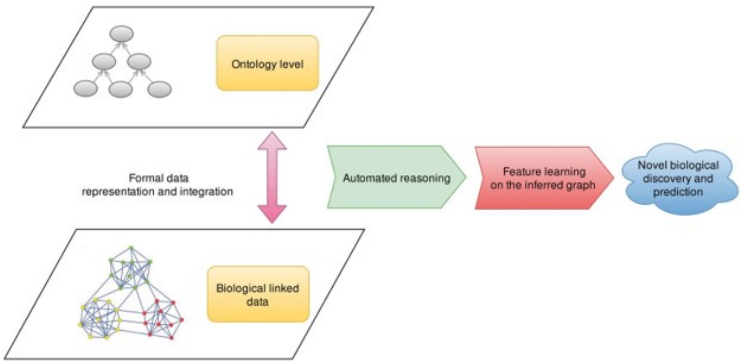


Figure 12: Overview of the workflow of the algorithm used to learn node embeddings from knowledge graphs. Source: [2]

5 Applications

This chapter contains an overview of other successful applications of neuro-symbolic learning. Instead of exhaustive enumeration of different algorithms, we concentrate here more on deeper explanation of selected, recent or better-known, approaches.

5.1 Games

The creation of programs to play board games or computer games has been one of the focuses of the field of artificial intelligence since its beginning, apart from entertainment these programs oftentimes serve as benchmarks of how close we have come in simulating "intelligence". However, purely symbolic approaches tend to fail in solving these problems due to the sheer size of state space and the number of possible actions. Furthermore, in games that require communication with other players, purely neural models may be prone to exploitation and manipulation.

5.1.1 AlphaZero

In their paper, Silver et al. [30], proposed a method to learn to play the game Go that combines symbolic search, specifically Monte-Carlo tree search, with a deep neural network that serves as a position evaluation heuristic. Later they generalized this algorithm into a generic algorithm called AlphaZero [29], which was able to achieve superhuman performance in various games like chess, shogi as well as Go.

The neural part of the AlphaZero [29] algorithm is a deep neural network, which serves as a heuristic for position evaluation and move choice; formally the neural network takes position s as an input and outputs estimation $v \approx \mathbb{E}[z|s]$ of expected outcome z for position s, and vector $\mathbf{p}$ of probabilities $p_a = P(a|s)$ for each action a, that represents the confidence of the network that a is a "good" move.

AlphaZero [29] uses the Monte-Carlo tree search to simulate self-play games, by means of traversing the tree from root to leaf node. At each state in the simulation, the next action with a low visit count, high move probability, and high value is selected (here move probabilities and values given by the neural network are used).

The network in AlphaZero [29] was trained by reinforcement learning, where it tries to minimize the difference between predicted outcomes (values) and game outcomes as well as minimize the difference between the predicted policies (action probabilities) and the probabilities π gathered from the Monte-Carlo tree search. Formally loss is calculated as:

$$loss = (z - v)^2 - \pi^T \log \mathbf{p} + c||\theta||^2.$$

Where the last term in the summation is L_2 regularisation.

5.1.2 Cicero

Bakhtin et al.[7] presented an AI agent Cicero, designed to play the game Diplomacy. Diplomacy is a game that requires both strategic planning and communication with other players, meaning the agent

needs to understand what other players are saying, plan his actions, and negotiate with others in a written dialogue.

The first part of Cicero is a neural language model used to understand and generate messages, that was pre-trained on text data from the web and fine-tuned on messages from the game with intent labels. The second part of the agent is a strategic reasoning module that is based on a planning algorithm that tries to choose an optimal action based on the game state and past dialogue with other players. Where the value and policy functions were trained with self-play reinforcement learning. The third Cicero component is a message filtering module based on classifiers that try to filter out messages that would be nonsensical or inconsistent with planned intent. [7]

5.2 Mathematics

5.2.1 Neural Theorem Provers

Neural theorem provers are an extension of automated theorem provers (computer programs that prove mathematical theorems) that substitute or expand some of their modules with neural networks.

In their paper, Polu et al. [26] proposed a neural theorem prover based on a language model, which is an encoder-only transformer, pre-trained on general and math-specific text data. That was then trained for theorem proving in Lean [23]. Lean is a functional programming language, that can be used as an interactive theorem prover. A part of it is the tactics framework, where tactics are user-defined or built-in commands or instructions that describe how to build terms (terms in this context might be a representation of construction or mathematical proof, e.g. rewrite tactic). The model is then trained on two objectives. First is the proof-step objective, in which the model tries to generate a proof-step (a Lean tactic) given some goal (some unification constraints). The second objective is the proof-size objective, which tries to guide the model to finish the proof and to converge to shorter proofs.

Knowladge Base Inference

Another possible application for theorem provers is a knowledge base inference. Symbolic theorem provers offer effective and precise reasoning or inference over knowledge expressed with logic however due to working with discrete symbols they fail to make use of similarities between predicates, constants, or symbols. To solve this problem, Rocktäschel and Riedel [27] propose combining the generalization capabilities of deep neural networks with theorem provers based on backward chaining.

The goal of backward chaining is finding the substitution of free variables with constants or known facts. It achieves this by iterating through OR and AND functions and unifying free variables using rules from the knowledge base. More specifically, at every step, a sub-goal's (sub-goals are in form of implication) left side of the implication is inputted into OR function that attempts to unify it with every rule's right side of the implication. Rules with successful unifications are then passed into the AND function that attempts to prove every atom on the right side of the rule's implication. This recursive process is repeated until we successfully unify all atoms with known facts or we fail (no unification is possible or maximum depth is reached). [27]

In the proposed neural theorem prover [27], the neural network is being recursively constructed. First by replacing goals, and substitution with initial vector representations of involved predicates and constants, where the representations are then optimized during training. Second, the unification operators are replaced with scoring functions. The scores are based on the similarity of the vector representation of predicates, constants, and the previous unification success of the network in subsequent steps. Lastly, the recursive steps of calling OR and AND functions with the knowledge base's rules define the connection (or structure) of the neural network. The training is done by iterating over the known facts as positive examples and unobserved facts as negative examples and optimizing the proof success (1 for positive and 0 for negative examples) with cross-entropy loss.

5.2.2 AlphaTensor

AlphaTensor presented by Fawzi et al. [8] is an agent designed for algorithm discovery, specifically the agent's goal is to find an efficient algorithm for the multiplication of arbitrary matrices. AlphaTensor is based on a similar approach to AlphaZero [29], it uses a transformer-based neural network as a value/policy function to guide the Monte-Carlo tree search. The model is trained with reinforcement

learning in a single-player game of tensor decomposition into rank-one terms, defined as:

$$T_{n,n,n} = \sum_{r=1}^{R} \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}.$$

Where $\otimes$ is the outer vector product, and $\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}$ are vectors. At each step in the game, where the current state is denoted as S_t (where the initial state is $S_0 = T_{n,n,n}$), the model chooses a triple $(\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)})$, and generates new current state as $S_t = S_{t-1} - \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$. The goal of the game is to reach $S_t = \mathbf{0}$ in as few steps as possible. The agent managed to find an algorithm that improved Strassen's two-level algorithm for matrix multiplication. [8]

5.3 Computer Vision

One of the important problems addressed by computer vision is visual concept learning, where the goal is to learn to recognize some specific objects and their attributes in an image. In this domain, neuro-symbolic learning might offer higher reliability and explainability, which is especially important for high-risk applications like robotics or autonomous driving.

In their paper Mao et al. [21] presented a concept learner for visual question answering, that is able to identify objects in a scene and answer questions about them or their relationship. The model comprises three main modules. First is a visual perception module based on Mask R-CNN (used for instance segmentation of an image), and ResNet-34 for object feature extraction, where each object's features are also extended with scene features to add contextual information. The second module is a semantic parser that translates a question into an executable program (represented in a formal language designed for visual question-answering), that tries to capture the semantic meaning of the given question. The final module is a program executor that instantiated the program with classified filtered object embeddings and executes it to obtain an answer; the filters are attribute-specific mapping from object embedding to attribute embedding spaces.

The training is done by jointly training the image feature extractor and the questions' semantic parser by maximizing the likelihood that the program gives the correct answer. The training was done in four stages, where the motivation stems from human concept learning (this means starting with simple localized concepts and expanding it into more complex relations in the scene): first was learning object-level visual concepts (e.g. What color is the object?), second learning relational questions (e.g. how many squares are right of the sphere?), third learning more complex questions, fourth joint fine-tuning of all modules. The model achieved state-of-the-art performance on various benchmarks and is able to generalize well to new visual compositions and concepts. However for the model to be applicable in, for example, robotics, it would need to be extended to work with actions, not just static visual concepts. [21]

6 Challenges & Current Research

Returning to the System 1 and System 2 discussion that started during the AAAI 2020 conference, multiple highly respected researchers such as Daniel Kahneman, Yoshua Bengio and Yann LeCun proposed different ways of bridging between System 1 and System 2. The debate also revolved around what would be the ingredients of an intelligent system. It is possible that two different research directions emerge, creating robust AI and achieving brain-like systems. Authors of the paper [9] define five ingredients for neuro-symbolic AI:

- Gradient-based optimization - used by current deep learning systems on large amounts of data.
- Modularity - allows one to refer to large parts of the network by the composition of symbols and relations among them.
- Symbolic language - describes knowledge encoded inside the network with the argument to use first-order logic as the canonical form of representation.
- Reasoning - once a complex network can be described symbolically, ideally in a compact abstract form any deductive reasoning is possible.
- Constraint satisfaction - symbols, learned, derived, or even invented, can act as constraints and help learning performance.

With these ingredients, there are many challenges to find the best way of achieving the combination of symbolic language and neural structure. The authors also set out three immediate challenges for neuro-symbolic AI:

- First-order logic and higher-order knowledge extraction from very large networks that are provably sound and yet efficient, explain the entire model and local network interactions and account for different levels of abstraction.
- Goal-directed commonsense and efficient combinatorial reasoning about what has been learned by a complex deep network trained on large amounts of multi-modal data.
- Human-network communication as part of a multi-agent system that promotes communication and argumentation protocols between the user and an agent that can ask questions and check her understanding.

When the field of AI started to rapidly develop, benchmarks and standardized tests for a variety of applications were created. Neuro-symbolic AI currently lacks any standard benchmarks and associated comprehensibility tests. These could offer a fair comparison with other approaches. With a focus on learning from fewer data, reasoning about extrapolation, computational complexity, and energy consumption [9].

7 Conclusion

Neuro-symbolic learning is a rather novel approach that connects artificial neural networks with symbolic reasoning. The methods are complementary to each other, thereby mitigating their weaknesses and accumulating their strengths. Expert knowledge and robust learning are integrated, resulting in highly-efficient learning algorithms.

In our overview, we explored the motivations behind neuro-symbolic learning and then the multitude of ways in which symbolic reasoning and neural networks can be combined. We selected three specific approaches and explained them in detail. First is the DeepProbLog approach combining neural networks with the probabilistic reasoning of the ProbLog programming language. The second is reinforcement learning settings to guarantee safe agent actions via efficiently verifiable symbolic representation of the agent's policy, and the third approach combines automated reasoning over ontologies with neural networks for knowledge graph embedding.

We further highlighted several applications of neuro-symbolic learning in games, mathematics, and computer vision. Lastly, we presented several challenges and possible research directions for neuro-symbolic learning.

References

[1] Joshua Achiam et al. *Constrained Policy Optimization*. 2017. arXiv: 1705.10528 [cs.LG].

[2] Mona Alshahrani et al. "Neuro-symbolic representation learning on biological knowledge graphs". In: *Bioinformatics* 33.17 (Apr. 2017), pp. 2723–2730. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx275. eprint: https://academic.oup.com/bioinformatics/article-pdf/33/17/2723/49040863/bioinformatics_33_17_2723_s1.pdf. URL: https://doi.org/10.1093/bioinformatics/btx275.

[3] Greg Anderson et al. "Neurosymbolic Reinforcement Learning with Formally Verified Exploration". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.

[4] Forough Arabshahi et al. *Compositional Generalization with Tree Stack Memory Units*. 2020. arXiv: 1911.01545 [cs.LG].

[5] M.M. Ashburner et al. "Gene Ontology: tool for the unification of biology. The Gene Ontology Consortium". In: *Nat Genet* 25 (May 2000), pp. 25–29.

[6] Samy Badreddine et al. "Logic Tensor Networks". In: *Artificial Intelligence* 303 (2022), p. 103649. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2021.103649. URL: https://www.sciencedirect.com/science/article/pii/S0004370221002009.

[7] Anton Bakhtin et al. "Human-level play in the game of Diplomacy by combining language models with strategic reasoning". In: *Science* 378.6624 (2022), pp. 1067–1074. DOI: 10.1126/science.ade9097. eprint: https://www.science.org/doi/pdf/10.1126/science.ade9097. URL: https://www.science.org/doi/abs/10.1126/science.ade9097.

[8] Alhussein Fawzi et al. "Discovering faster matrix multiplication algorithms with reinforcement learning". In: *Nature* 610.7930 (Oct. 2022), pp. 47–53. ISSN: 1476-4687. DOI: 10.1038/s41586-022-05172-4. URL: https://doi.org/10.1038/s41586-022-05172-4.

[9] Artur d'Avila Garcez and Luis C. Lamb. *Neurosymbolic AI: The 3rd Wave*. 2020. arXiv: 2012.05876 [cs.AI].

[10] Thomas R. Gruber and Gregory R. Olsen. "An Ontology for Engineering Mathematics". In: *Principles of Knowledge Representation and Reasoning*. Ed. by Jon Doyle, Erik Sandewall, and Pietro Torasso. The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, 1994, pp. 258–269. DOI: https://doi.org/10.1016/B978-1-4832-1452-8.50120-2. URL: https://www.sciencedirect.com/science/article/pii/B9781483214528501202.

[11] Pascal Hitzler et al. "Neuro-symbolic approaches in artificial intelligence". In: *National Science Review* 9.6 (2022), nwac035.

[12] Robert Hoehndorf, Paul Schofield, and Georgios Gkoutos. "The role of ontologies in biological and biomedical research: A functional perspective". In: *Briefings in bioinformatics* 16 (Apr. 2015). DOI: 10.1093/bib/bbv011.

[13] Henry Kautz. "The third AI summer: AAAI Robert S. Engelmore memorial lecture". In: *AI Magazine* 43.1 (2022), pp. 105–125.

[14] Henry A. Kautz. "The third AI summer: AAAI Robert S. Engelmore Memorial Lecture". In: *AI Magazine* 43.1 (2022), pp. 105–125. DOI: https://doi.org/10.1002/aaai.12036. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aaai.12036. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aaai.12036.

[15] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. "The Incredible ELK". In: *Journal of Automated Reasoning* 53.1 (June 2014), pp. 1–61. ISSN: 1573-0670. DOI: 10.1007/s10817-013-9296-3. URL: https://doi.org/10.1007/s10817-013-9296-3.

[16] Warren A. Kibbe et al. "Disease Ontology 2015 update: an expanded and updated database of human diseases for linking biomedical knowledge through disease data". In: *Nucleic Acids Research* 43.D1 (Oct. 2014), pp. D1071–D1078. ISSN: 0305-1048. DOI: 10.1093/nar/gku1011. eprint: https://academic.oup.com/nar/article-pdf/43/D1/D1071/17435884/gku1011.pdf. URL: https://doi.org/10.1093/nar/gku1011.

[17] Sebastian Köhler et al. “The Human Phenotype Ontology project: linking molecular biology and disease through phenotype data”. In: *Nucleic Acids Research* 42.D1 (Nov. 2013), pp. D966–D974. ISSN: 0305-1048. DOI: 10.1093/nar/gkt1026. eprint: https://academic.oup.com/nar/article-pdf/42/D1/D966/3529478/gkt1026.pdf. URL: https://doi.org/10.1093/nar/gkt1026.

[18] Harsha Kokel. *The 6 types of neuro-symbolic systems*. 2020. URL: https://harshakokel.com/posts/neurosymbolic-systems/.

[19] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[20] Robin Manhaeve et al. “DeepProbLog: Neural Probabilistic Logic Programming”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf.

[21] Jiayuan Mao et al. *The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision*. 2019. arXiv: 1904.12584 [cs.CV].

[22] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. arXiv: 1310.4546 [cs.CL].

[23] Leonardo de Moura et al. “The Lean Theorem Prover (System Description)”. In: *Automated Deduction - CADE-25*. Ed. by Amy P. Felty and Aart Middeldorp. Cham: Springer International Publishing, 2015, pp. 378–388. ISBN: 978-3-319-21401-6.

[24] OpenAI. 2022. URL: https://openai.com/blog/chatgpt.

[25] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “DeepWalk”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, Aug. 2014. DOI: 10.1145/2623330.2623732. URL: https://doi.org/10.1145%5C%2F2623330.2623732.

[26] Stanislas Polu et al. *Formal Mathematics Statement Curriculum Learning*. 2022. arXiv: 2202.01344 [cs.LG].

[27] Tim Rocktäschel and Sebastian Riedel. “Learning Knowledge Base Inference with Neural Theorem Provers”. In: *Proceedings of the 5th Workshop on Automated Knowledge Base Construction*. San Diego, CA: Association for Computational Linguistics, June 2016, pp. 45–50. DOI: 10.18653/v1/W16-1309. URL: https://aclanthology.org/W16-1309.

[28] Yongliang Shen et al. *HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in HuggingFace*. 2023. arXiv: 2303.17580 [cs.CL].

[29] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI].

[30] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (Oct. 2017), pp. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270.

[31] Abhinav Verma et al. “Imitation-Projected Programmatic Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/5a44a53b7d26bb1e54c05222f186dcfb-Paper.pdf.

[32] Stephen Wolfram. *Chatgpt gets its "Wolfram superpowers"!* Mar. 2023. URL: https://writings.stephenwolfram.com/2023/03/chatgpt-gets-its-wolfram-superpowers/.