Deep Deductive Reasoning is a Hard Deep Learning Problem

Pascal Hitzler *, Rushrukh Rayan, Joseph Zalewski, Sanaz Saki Norouzi, Aaron Eberhart and Eugene Y. Vasserman

Kansas State University, USA
E-mails: {hitzler, rushrukh, jzalewski, sanazsn, aaroneberhart, eyv}@ksu.edu

Abstract. Deep Deductive Reasoning refers to the training and then executing of deep learning systems to perform deductive reasoning in the sense of formal, mathematical logic. We discuss why this is an interesting and relevant problem to study, and explore how hard it is as a deep learning problem. In particular, we present some of the progress made on this topic in recent years, understand some of the theoretical limitations that can be assessed from existing literature, and discuss negative results we have obtained regarding improving on the state of the art.

Keywords: Deductive Reasoning, Deep Learning, Theoretical Limitations of Deep Learning, Memory Networks, Transformers

1. Introduction

Deductive reasoning over formal logics is arguably *the* central computational task in symbolic Artificial Intelligence (AI) [1]. A given formal logic usually carries a formal semantics that is often model-theoretic and as such defines in mathematically precise terms the notion of logical consequence. In symbolic form, given a logical theory K (in symbolic AI – more precisely in the subfield of *Knowledge Representation* – often called a *knowledge base*, i.e., a set of logical formulas over the formal logic), we write $K \models F$ if F is a logical formula over the same language and F is a logical consequence of K, meaning that F can be derived from K by *deductive reasoning*.

An important aspect of Knowledge Representation (KR) research is to find correct and scalable algorithms for deductive reasoning over KR-relevant logics. However, the mathematical definitions often do not easily translate into algorithms. Indeed, algorithms for more expressive logics tend to be so involved that they often need correctness proofs. Furthermore, they tend to be of high computational complexity, if the problem of computing logical consequences is at all decidable. Indeed, some prominent logics, such as first-order predicate logic, are not decidable but only semi-decidable, and some higher-order logics, including some non-monotonic logics of importance to KR, are not even semi-decidable [2]. Among the decidable logics are many from the Description Logics family [3] that are relevant for KR-based data management, and some of which have become W3C standards for the Semantic Web research field [4]. Their complexities can be very high, e.g., the OWL 2 DL standard is based on the Description Logic $\mathcal{SROIQ}$ and has $N2\mathsf{EXPTIME}$[1] worst-case complexity [5]. In this paper we will mostly focus on expressive logics of relevance to KR, i.e., logics expressed over languages with variables such as first-order predicate logic.

*Corresponding author. E-mail hitzler@ksu.edu.

[1] which is nondeterministic double exponential, i.e., nondeterministic $2^{2^{f(n)}}$ where f is polynomial in n

Propositional logic, which is computationally much more friendly,[2] is of course also important for many computer science applications, but less relevant for our interests.

Deep Deductive Reasoning (DDR) [8] refers to the training and then executing of deep learning (DL) systems to perform deductive reasoning. There are several reasons why the study of DDR is a worthwhile undertaking. Fundamentally, DDR presents us with a set of related problems of differing difficulty, at scales that can be chosen arbitrarily, and with mostly easy-to-obtain training data, and thus makes it possible to explore the capabilities and limitations of DL in what is similar to a controlled laboratory setting. A different perspective on the problem is motivated from a cognitive science viewpoint, in that KR and DL can be understood as simple formal abstractions of cognition, respectively, the human brain: And while we humans are capable of performing deductive reasoning, we would like to understand if and how this can be done with DL systems. A third perspective is more applied: DDR, if it could be made to work with high accuracy and at scale, may lead to improvements over conventional deductive reasoning algorithms through better tolerance of errors and noise in the data (which conventional algorithms do not handle well) and to improved reasoning speed (in the light of high computational complexities). At the same time, capturing deductive reasoning in DL systems opens avenues for neurosymbolic architectures that truly combine learning and reasoning, without the need to recourse to hybrid architectures.

The rest of the paper is organized as follows. In Section 2, we will give a brief overview of existing DDR research. In Section 3 we will discuss theoretical limitations on DDR that can be derived from the literature. In Section 4 we report on some of our failed attempts to improve DDR, which we include to emphasize how difficult the problem is, and in order to aid other researchers by reporting on paths that appear to be dead ends. In Section 5 we conclude.

2. Brief Literature Review

There are a number of particularly important features when assessing DDR approaches and systems:

- The specific logic that is being reasoned over, and in some cases also the reasoning task, as different reasoning tasks have differing complexities for some logics. For example, in logic programming [9], often only the derivation of logical consequences that are *ground facts* (and not complex formulas) is important. Computational complexities of the reasoning tasks provide a first indicator how difficult reasoning over the logic is. A less formal notion than complexity is *expressivity* of a logic: A more expressive logic makes it easier for a human to encode complex problems in them. For example, RDFS and $\mathcal{EL}^+$ are both of polynomial complexity (the latter for so-called completion reasoning) [5], but it would be reasonable to argue that $\mathcal{EL}^+$ is more expressive than RDFS, despite the lack of a formal notion for this. It is then reasonable to expect that logics that are of higher computational complexity, or are more expressive, pose higher difficulties for DDR.
- In order to do DDR, the deductive reasoning problem needs to be transformed into a problem suitable for DL. There appear to be two different principled ways to do this. The first is to phrase DDR as a classification problem: Given a knowledge base K and a logical formula F, determine whether or not $K \models F$. The second is to create a *generative* DDR system that is capable of producing *all* logical consequences of a given knowledge base. Of course, for logics where the set of all logical consequences is infinite, some reasonable finiteness restrictions have to be imposed, but this can usually be done in some meaningful and straightforward way, e.g., by limiting the syntactic length of formulas considered as logical consequences. It is reasonable to assume that generative DDR is more difficult than DDR via classification: although in principle they can be reduced to each other, the reduction of classification to the generative approach is trivial, while the converse requires running the classification for each possible formula, an approach that will not scale. For this scalability reason, generative DDR promises more avenues for applications.
- In deductive reasoning, the names of identifiers (constants, predicates, etc.) are insubstantial in that a consistent renaming within a knowledge base does not change the deductive reasoning results or procedures. Symbolic algorithms are likewise independent of identifier names. Ideally, DDR systems would have the same capability, i.e., to transfer to inputs over new and previously unseen vocabularies.

[2] Propositional reasoning can in fact be represented *exactly* using artificial neural networks, see, e.g., [6, 7].

Table 1
Comparison of DDR approaches

Method	Logic	Generative	Transferable	Scalability	Training time	Testing time	Accuracy
[10]	RDF	No	No	Moderate	$\approx$ 60 min	< 1ms	Accuracy of 87-99%
[11]	RDF	**Yes**	No	Moderate	N/A	N/A	F1-score of 0.03-1
[12]	RDFS	**Yes**	No	Low	$\approx$ 12 min	N/A	Accuracy of $\approx$ 100%
[13]	RDFS	**Yes**	No	Low	N/A	N/A	Accuracy of 95%
[14]	RDFS	No	**Yes**	Moderate	N/A	N/A	Accuracy of 52-96%
[15]	ASP	**Yes**	No	Very low	N/A	N/A	N/A
[16]	$\mathcal{EL}^+$	**Yes**	No	Moderate	N/A	N/A	Somewhat better than guessing
[17]	$\mathcal{EL}^{++}$	No	No	High	N/A	N/A	Hits at rank 1 $\approx$ 0.06
[18]	OWL 2 RL	No	No	Low	N/A	N/A	Accuracy of 99%
[19]	$\mathcal{ALC}$	No	No	Moderate	< 27 min	N/A	Accuracy of $\approx$ 97%
[20]	OWL DL	**Yes**	No	High	N/A	< 355 s	F1-score of $\approx$ 0.95
[21]	FOL	**Yes**	No	Very low	$\approx$ 20 sec	N/A	Precision of 0.7

– Scalability is another important aspect and has at least three distinct variants:
  * Scalability in terms of input size. Application oriented knowledge bases sometimes have hundreds, and sometimes millions or even billions of logical statements. For DDR systems, it is important to note what input sizes can be ingested so that accuracy is still high.
  * Scalability in terms of training time needed, which of course is related to the input sizes considered.
  * Scalability in terms of the runtime of the trained DDR system.
– Finally, accuracy of a DDR system is of importance, and it is probably best assessed in terms of precision, recall, and f-measure values by taking a symbolic reasoner as basis for ground truth.

Considering these dimensions, we provide a comparison of existing DDR approaches of which we are aware, summarized in Table 1.[3] In [10], their goal was to use LSTMs and CNNs to approximate stream reasoning with C-SPARQL (Continuous SPARQL). C-SPARQL has a limit of processing triples per second, and as queries become more complicated, the time to provide answers will increase, and in cases where the amount of triples per second exceeds the limit, C-SPARQL provides incorrect results. Thus, the reasoning time will increase when we have an enormous amount of data, so an approximation method should be applied. The results show that for highly complicated queries, the LSTM provides better results, while the CNN performed a little bit better for other queries and the time is less than for the LSTM. A Recursive Reasoning Network (RRN) is proposed as the deep learning model for ontology reasoning in [18]. It is mentioned in their paper that knowledge bases with the same ontology but different facts are used for training, so regarding the transferability, they noted that the trained model can be used on different databases with similar vocabularies (unseen KB but for the same ontology), so we considered it as a non-transferable reasoning model in our sense. In [21], a model based on Logic Tensor Networks (LTNs) is applied for reasoning over first-order predicate logic (FOL). They claim that this model is a good model for simple reasoning, and also, after training, they can reason over combinations of axioms they were not trained on. A Bidirectional GRU-based model is proposed in [12] for noise-tolerant symbolic reasoning. Their results show that in those classes with thousands of instances, the accuracy is high, on the other hand, this model for the small-instanced classes does not perform well as is expected in deep learning. In [13], they proposed a model that also provides explanations for how the inferences are generated besides the inferences; their model consists of a parallel path using bidirectional GRU. A non-generative but transferable model based on memory networks is developed in [14]. For this purpose, RDF triples are stored and the model is trained to embed the knowledge graph to be able to predict the query result. Note that this is the only approach to date that has been shown to be highly transferable, with high accuracy and moderate scalability. In [16], an LSTM-based noise-resistant model is presented for learning $\mathcal{EL}^+$ reasoning, however with only low accuracy; more details can be found in Section 4. Also, [15] presents an end-to-end method for answer set programming (ASP), considering loop formula constraints in Lin-Zhao's theorem to minimize the cost function, and their neural network computation only consists of a RELU network and a linear output layer. Their

[3] Most, but not all, of DDR work for KR-relevant logics has been done on logics relevant to Semantic Web [5].

algorithm was evaluated on very small problems and the average of its output compared to actual solutions, as well as the algorithm's running time, was reported. In [11] they approached reasoning as a problem of transforming one graph into another (inference graph) and the model is based on the Bidirectional-GRU. An approximate reasoning method for ABox is proposed in [20], it is a CNN-based model which aims to learn to do logical deduction on their generated synthetic data, and also, it is mentioned that the test data does not have any interaction with train data, but since they used a shared TBox, it cannot be considered fully transferable. It is worth mentioning here that the reasoning time of Pellet and Hermit, which are state-of-the-art deductive reasoning systems over description logics, is much higher than the proposed deep learning method. In [17, 22], the authors proposed a model to consider many-to-many relations for embedding ontologies, and then they provide a subsumption classification evaluation. In the table, the average of their evaluation of many-to-many relations is shown. We can see that accuracy is rather low. [19] presents a feed-forward neural network in order to provide an approximate classification of subsumptions for $\mathcal{ALC}$ ontologies, and also, it should be noted that as the embedding layers for a new knowledge base must be trained using the reasoner (the reasoner head is frozen for this purpose) this model is partially transferable.

We note, in particular, that none of the current systems is both generative and able to transfer. Furthermore, scalability is generally below many requirements in practice. The exception may be [20] – however this system makes use of only small TBoxes (which are the actual logical axioms), i.e., only the ABox (essentially, the non-axiom facts) is large.

While it is not our focus herein, we would like to mention that some studies have focused on investigating reasoning within propositional logic. For instance, in [23] efforts were made to fine-tune GPT-2 and GPT-3 to emulate resolution rules applicable to propositional logic programs, and in [24] the issue of entailment within propositional logic is addressed.

Also related is research in automated theorem proving involving DL, which can be categorized into two main groups. Firstly, there are hybrid approaches that combine neural network encodings with conventional deduction methods [25–27]. Secondly, there exist fully deep learning-based models such as the one proposed by [28], which presents a DNN architecture customized for automated proof synthesis, estimating the probability of applying an inference rule at each step of the proof. Additionally, in [29, 30], a graph neural network-based model is proposed for the classification and resolution of SAT problems.

OVerall it seems fair to say that, at present, research results do not yet meet realistic application requirements, in particular for KR applications.

3. Theoretical Limitations

Deductive reasoning problems have a long history as complete problems for complexity classes, and those in knowledge representation tend to possess high complexity. Therefore it is natural to think that DDR is subject to severe theoretical limitations, *but is this true*?

Deep learning can be studied through several abstract models. The most direct and literal is the analytic model (see e.g., [31, 32]), in which neural networks are a class of functions of real variables, studied in terms of approximation, convergence, etc., with computational considerations kept to a minimum. An advantage of this point of view is its ability to see specifics of different architectures that would be washed out in other models, letting us more often compare predictions against empirical results. However, it is not well suited to study hardness in absolute terms: typically one can prove facts such as rapid convergence-in-probability of stochastic gradient descent, or lack thereof, but this only concerns the performance of one algorithm (e.g., stochastic gradient descent) at a time.

The other obvious model is boolean circuit computation [33]. We represent neural networks as discrete functions acting on boolean vectors, and so all considerations are quite different. Typically some proxy such as "majority threshold circuits of constant depth" (TC_0 circuits) is used for neural networks [34], which are, after all, structured like big circuits, with inputs feeding forward to outputs, and threshold gates are indeed similar in spirit to neurons. However, a "reduction" is implicit between the neural architecture and the abstraction, which makes differences

between architectures invisible.[4] The facts that this model represents are different at a more basic level too: phenomena of floating-point representation and rounding error, for example, are implicitly taken into account, which are ignored in the analytic model. Could these differences be purely a priori, and in fact the practical limitations implied by the two systems are similar? The state of the art in both is too undeveloped to know, but we can mention one "coincidence": in both models, a small fixed number of layers seems to provide a lot of expressiveness. MLPs with one hidden layer can uniformly approximate continuous functions on a compact domain[5] [31]. And no one has yet proven that TC_0 circuits of three layers do not solve all problems in EXPTIME [34].

The circuit model is more germane to the study of DDR, because deductive reasoning problems are discrete, but there is another issue that bears mentioning. Whereas a Turing machine has a finite number of states, a circuit family, with enough circuits to process inputs of any length, is an infinite one [34] because a different circuit is required for each input length. It is thus possible to decide some nonrecursive languages with circuits. This feature, called "nonuniformity", is a plague on lower-bound proofs; proving problems too hard for circuits is generally harder than proving them too hard for Turing machines with similar resource bounds. We do not understand whether this reflects a significant phenomenon or just a gap in our knowledge.

When a decision (classification) problem is not in a nonuniform complexity class, such as TC_0, this means that *no family of small circuits of the appropriate type can perform the classification correctly*. Assuming the often-assumed analogy between circuits and neural networks [34], it would say the problem is hard in a way that has nothing to do with *training* – no matter how well we can train our networks, there is no correct network to be found by training. Therefore we might get better bounds if we modeled neural networks instead by *uniform* families of threshold circuits, which have an efficient procedure that generates the circuits, a process that is potentially analogous to training. One well-studied notion of uniformity is efficient decidability of the "direct connection language" which fully describes a circuit family,[6] but this represents the extent to which a network's structure has a simple mathematically-regular "plan"; intuitively this has nothing in common with the kind of constructiveness in networks trained by backpropagation.[7] For the special case of DDR, we may fall back on a blunter notion: we may model a pair of training algorithm and neural architecture as a circuit family generated by a Turing machine, the action of this Turing machine encompassing both generation of training data *and* the learning procedure. This model is more appropriate for DDR than for most deep learning problems, since we know how to write a program to generate ground truth in the form of sets of valid axioms. Neural networks conceived this way have serious limitations. For instance: suppose we want to use a polynomial-time[8] procedure to train a reasoning network for an EXPTIME-hard description logic problem, such as "$\mathcal{ALC}$ subsumption with respect to a general TBox" [39]. Suppose we generate training data deterministically using a reasoner program, and we demand perfect accuracy from the trained network. If we can generate the data in less than exponential time (strictly, time $O(2^{n^{o(1)}})$), then by pipelining the two processes we will end up with a deterministic algorithm that does $\mathcal{ALC}$ subsumption testing in less than exponential time, which is impossible. The takeaway is that for hard logic tasks we cannot both learn efficiently and generate quality training data efficiently – at least one of these steps must be a bottleneck.

We have neglected the issue that deep learning is inherently random and fuzzy. Since most known facts about hardness of logic concern exact decision problems, with no provision for error, the easiest type of limitations to prove are those of exactly-correct networks. These results mostly just confirm the intuition that deep learning *has* to be random or fuzzy. Sometimes, though, results do generalize to the fuzzy case. Suppose we want to train, in polynomial time and from a polynomial amount of data, a network that does $\mathcal{SROIQ}$ reasoning with merely approximate

[4] There are exceptions. Some neural architectures, e.g., recurrent, can feed variable-length inputs to a single network [35]; we can easily represent its different "length modes" as different circuits, but this makes "constant depth" implausible and introduces uniformity (see below). Constant-depth may also be implausible for convolutional networks. Recognizing objects in images generally requires more down-sampling layers for higher resolution, i.e., at least logarithmically more layers for larger inputs. The class of languages accepted by logarithmically-deep threshold circuits is called TC_1.

[5] a fact used to link deductive and reasoning with MLPs in the pre-DL era [7, 36]

[6] variously defined; see e.g., [33, 37]

[7] Though it may have something unintuitively in common; sometimes trained networks "organize" in a planned-looking way [38].

[8] in the network input width and in the amount of training data

correctness.[9] Then no polynomial-space procedure can generate our training samples. If it could, then there would be a PSPACE algorithm for $\mathcal{SROIQ}$,[10] which is impossible because $\mathcal{SROIQ}$ satisfiability is N2EXPTIME-hard.

For deep learning in general, one might argue that we do not have a computational model for the knowledge in a dataset of arbitrary real-world examples because thinking about the example generator as a Turing machine is misleading – we do not know whether the universe can be modeled as a Turing machine.[11] Even for DDR, perhaps the bottleneck really is just the uninsightful ways we can automatically generate ground truth, and if we trained on example deductions made by human experts we could do better.[12] If this is so, then perhaps we ought to model neural networks as if training data is arbitrary. This approximation made, the only circuit-based limits to their power are non-uniform ones, but there is now no possibility of stochastic models performing better than exact ones, since randomness and error tolerance give no advantage to nonuniform computers.[13] It is almost trivial to prove that 2EXPTIME-hard problems do not have polynomial-size circuits. So $\mathcal{SROIQ}$ reasoning is out of reach even with the most ideal training set and unlimited time to train stochastically.

Weaker logics provide more interesting problems. It is widely conjectured, though hard to prove (it implies P $\neq$ NP), that NP-hard problems do not have polynomial-size circuits. If this is the case, then regardless of our assumptions about training, no neural architecture whose operation, once trained, involves a polynomial number of calculations can solve the SAT problem for Boolean formulas, which are vastly less expressive than e.g., $\mathcal{SROIQ}$.

One pertinent compromise model between uniform and nonuniform is *exact learning* [42, 43], in which circuits are constructed by a Turing machine with query access to the target function, and an oracle that can provide counterexamples for incorrect circuits. From an AI perspective, this corresponds to *active learning* with ideal data available but limited computational resources. Klivans, Kothari, and Oliveira show [44, Theorem 8] implies[14] that if there exists an algorithm to exact-learn *all* functions learnable by a particular class of circuits (i.e., an "architecture") in polynomially many queries, then there are functions computable in almost polynomial time[15] that cannot be represented by this architecture. The takeaway from this is that, insofar as exact learning is a good abstraction, there is a tradeoff between the generalization capacity of an architecture and guarantees of efficient learning, and many deductive reasoning problems are hard enough to bring this tradeoff into play.

Overall, the verdict of theory (as of the time of writing) is that the limitations of deep learning for logic problems are not well understood. Since these problems occur at many levels of the complexity hierarchy, they provide an interesting bellwether for the hardness of deep learning in general. Indeed, all the results we have been able to cite above have been excessively general; if there is something particularly challenging for practical neural systems about reasoning, as compared to other computationally hard tasks, this remains to be explained by future research.

4. Practical Hardness

In principle, DDR should represent a deep learning problem that is readily investigated. Using symbolic reasoners it is straightforward to produce vast amounts of training data algorithmically. Given the promises of deep learning as a cure-all for all manner of complex problems, it should be expected that, at least for simple logics, solutions for DDR should be findable without too much hassle. It turns out that this is not the case at all, even when staying within

[9] We mean that, for some ϵ, for every input x, if we train a fresh network, it will accept x with probability at least $0.5 + \epsilon$ if x is a positive instance and no more than $0.5 - \epsilon$ otherwise.

[10] This is because we can use random bits to train a network, then simulate it on the input, and the result is a polynomial-time stochastic algorithm that is correct if given the right training data, but whenever such an algorithm exists there is also a deterministic polynomial-space algorithm [40], which we can pipeline with the data-generator.

[11] e.g., because the Turing machine in question would be so intricate that its size is not negligible compared to our inputs, which calls into question the whole framework of "asymptotic resource use in terms of input size"

[12] This is an idealization, because human experts only work up to a certain input size, but they can solve many reasoning problems not captured by any known decidable logics.

[13] BPP $\subseteq$ P/poly: See [41, Section 3.2] for Turing machines with polynomial randomness. We can also prove that if any problem is solvable with bounded 2-sided error by choosing poly-size circuits at random from an arbitrary distribution, that problem can be solved deterministically by poly-size circuits.

[14] see the abstract for this claim

[15] $\leqslant n^{f(n)}$ where f is an arbitrary $\omega(1)$ function

the bounds of the theoretical limitations discussed above. In this section, we report on two of our investigations, and their (relative) failures. Given the frequent emphasis in published research on positive results, we take this opportunity to discuss negative results, i.e., failure to solve DDR in some settings.

4.1. LSTMs for $\mathcal{EL}^+$ Reasoning

We briefly discuss the investigation reported in detail in [16] regarding the use of LSTMs for $\mathcal{EL}^+$ DDR. We have looked at a number of different approaches using LSTMs that naturally lead to generative but non-transferable systems. To already state the result: as reported in [16], we obtained DDR that was just a little bit better than random guessing, but for larger knowledge bases. We now recap in more detail what we investigated.

It was our intention to avoid, as much as possible, embeddings or any distance adjustments that might help the system learn answers based on embedding similarity without first learning the reasoning structure.[16] In fact, a primary feature of our network is its lack of complexity. We take a very simple logic, reason over knowledge bases in that logic, and then extract so-called *supports* from the reasoning steps, mapping the reasoner supports back to sets of the original knowledge base axioms. This allows us to encode the input data in terms of only knowledge base statements. Essentially, such a support consists, for each reasoning step in a symbolic reasoner, of the set of axioms (in the original knowledge base) that allow to draw the inferences for this reasoning step.[17] Support also provides an intermediate answer that might improve results when provided to the system. We experimented with three different LSTM architectures with identical input and output dimensionalities. One architecture, which we call "Deep", does not train with support data but has a hidden layer the same size as the supports we have defined. Another architecture, called "Piecewise", trains two separate half-systems, one with knowledge bases and one with supports from the reasoner.[18] The last system, called "Flat", simply learns to map inputs directly to outputs for each reasoning step. We will discuss in detail each part of the system in the following sections, then provide some results.

$\mathcal{EL}^+$ is a lightweight and highly tractable description logic. A typical reasoning task in $\mathcal{EL}^+$ is a sequential process with a fixed endpoint, making it a perfect candidate for sequence learning. Unlike RDF(S), which reasons over instance data in triple format (*ABox* in description logic terminology [5]), $\mathcal{EL}^+$ reasoning occurs on the predicate level (*TBox* in description logic terminology [5]). Thus we will have to train the system to actually learn the reasoning patterns and logical structure of $\mathcal{EL}^+$ directly from encoded knowledge bases. More information about $\mathcal{EL}^+$ can be found in [45, 46].

LSTMs are a type of recurrent neural network that work well with data that has long-term dependencies [47]. We choose to use an LSTM to learn the reasoning patterns of $\mathcal{EL}^+$ because LSTMs are designed to learn sequence patterns. The network is kept as simple as possible to achieve the outputs we require and maximize transparency. For the piecewise system, depicted in Figure 1 (top), we use two separate flat single LSTMs that have the number of neurons matching first the support shape, and later the output shape. The deep system shown in Figure 1 (middle) is constructed to match this shape so that the two will be comparable, it simply stacks the LSTMs together so that it can train without supports. We also train a flat system, shown in Figure 1 (bottom), that has the same input and output shape but lacks the internal supports. We have done this three ways because we can compare the behavior of the systems with support against the flat system as a baseline, and then see whether the piecewise training is helping. All designs treat the reasoning steps as the sequence to learn, so the number of LSTM cells is defined by the maximum reasoning sequence length. We train our LSTMs using regression on mean squared error to minimize the distances between predicted answers and correct answers.

We would like to emphasize the distinction between our method, which we prefer to call an encoding, and a traditional embedding. Our encoding adds no additional information to the names beyond the transformation from integer to scaled floating point number. (It can be directly reversed with a slight loss of precision due to integer conversion.) The semantics of each encoded predicate name with regards to its respective knowledge base is structural

[16] We just don't see how pre-compiled embeddings would be helpful for deep deductive reasoning if reasoning should transfer in the sense discussed in Section 2: To have embeddings reflect similarity relevant for transferable deductive reasoning would have to capture similarity in some sense conveyed by the input theory, which is not known before runtime, i.e., which is in particular not available for training embeddings.

[17] As such, support is dependent on the symbolic reasoning algorithms chosen, and we used the most basic one [5].

[18] I.e., the system learns to produce support, and then learns to create output from both support and input knowledge base

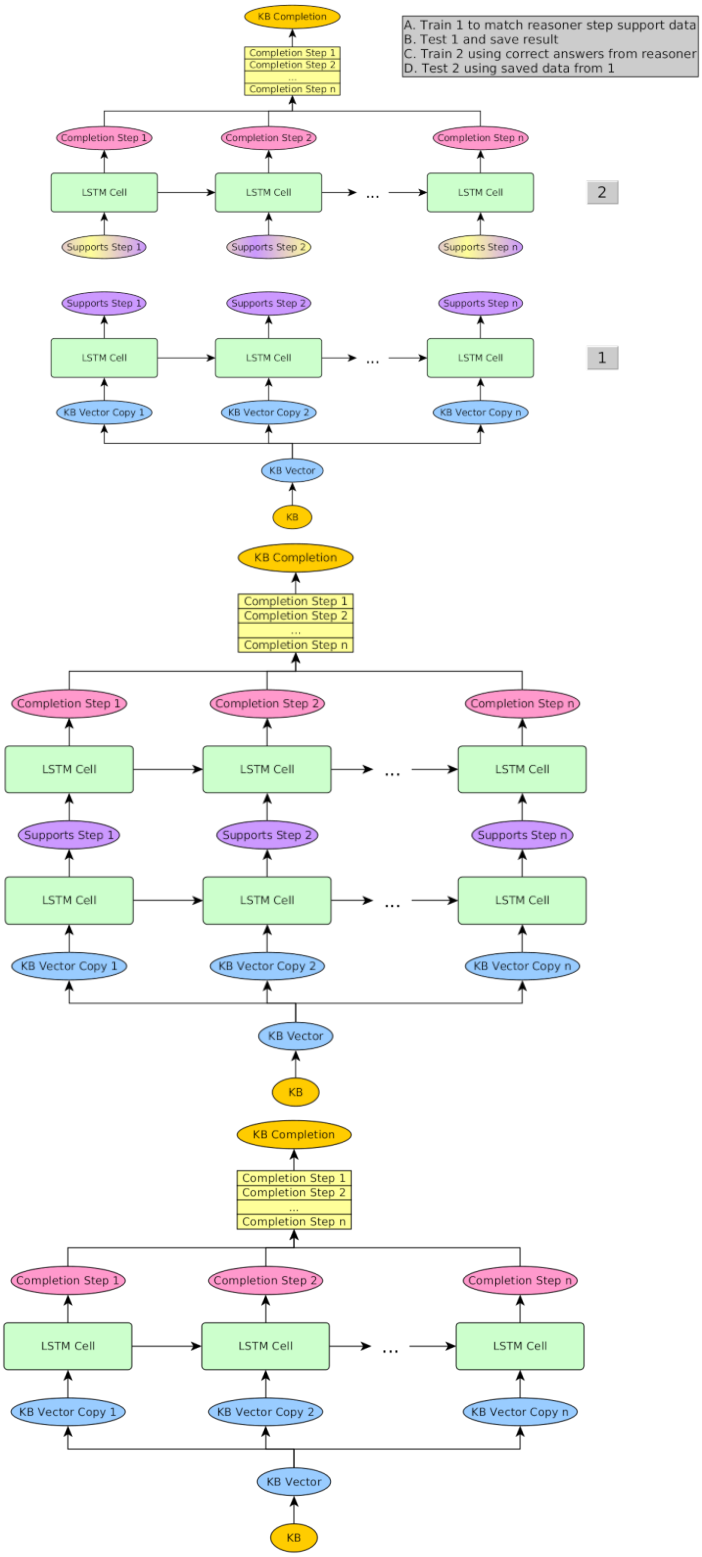

Fig. 1. Piecewise Architecture (top); Deep Architecture (middle); Flat Architecture (bottom)

in relation to its position in the 4-tuple (just like it would be in an $\mathcal{EL}^+$ axiom) and its semantics is not related to the other non-equal predicate names within the same 4-tuple.

Every data set we use has a fixed number of names that it can contain for both roles and concepts, and every concept or role has an arbitrary number of name identifiers. Identifier labels are stripped and the concept and role numbers are shuffled around and substituted with random integers to remove any possibility of the names injecting a learning bias. These labels are remembered for later retrieval for output but the LSTMs do not see them. Knowledge bases have a maximum number of role and concept names that are used to scale all of the integer values for names into a range of $[-1, 1]$. To enforce disjointness of concepts and roles, we map all concepts to $(0, 1]$ and all roles to $[-1, 0)$. Each of the six possible normalized axiom forms is encoded into a 4-tuple based on the logical encodings defined in Table 2.

To input knowledge bases into the LSTM, we first apply the encoding defined in the previous section to each of its statements then concatenate each of these encodings end-to-end to obtain a much longer vector. For instance, C1 $\sqsubseteq \exists$R1.C2, R1 $\sqsubseteq$ R2 might translate to $[0.0, 0.5, -0.5, 1.0, 0.0, -0.5, -1.0, 0.0]$. This knowledge base vector is then copied the same number of times as there are reasoning steps and stuck together once again along a new dimension. Experiments are repeated hundreds or thousands of times, filling up the tensor that is given to the LSTM to learn. Because some of the randomness can cause ragged data, any tensor dimensions that are not the same length as the maximum are padded with zeros.

Table 2: Translation rules:
c = number of possible concept names
r = number of possible role names

KB statement		Vectorization
CX $\sqsubseteq$ CY	$\rightarrow$	$[\ 0.0,\ \frac{X}{c}, \frac{Y}{c},\ 0.0\ ]$
CX $\sqcap$ CY $\sqsubseteq$ CZ	$\rightarrow$	$[\ \frac{X}{c},\ \frac{Y}{c},\ \frac{Z}{c},\ 0.0\ ]$
CX $\sqsubseteq \exists$RY.CZ	$\rightarrow$	$[0.0,\ \frac{X}{c},\ \frac{-Y}{r},\ \frac{Z}{c}]$
$\exists$RX.CY $\sqsubseteq$ CZ	$\rightarrow$	$[\frac{-X}{r},\ \frac{Y}{c},\ \frac{Z}{c},\ 0.0]$
RX $\sqsubseteq$ RY	$\rightarrow$	$[0.0, \frac{-X}{r}, \frac{-Y}{r}, 0.0]$
RX $\circ$ RY $\sqsubseteq$ RZ	$\rightarrow$	$[\frac{-X}{r}, \frac{-Y}{r}, \frac{-Z}{r}, 0.0]$

As already mentioned (and we will not replicate the results tables from [16]), our attempts failed in so far as we only managed to do a little bit better than a random guesser. Of course this does not mean that the problem is not solvable, but it appears that a-priori reasonable set-ups do not readily solve it.

4.2. Pointer Networks for Generative RDFS Reasoning

Artificial Neural Networks (ANNs) have been in use to learn sequence-to-sequence problems for a long time. Particularly, Recurrent Neural Networks (RNNs) have achieved state-of-the-art performance in mapping such problems. One key point in such applications is that the dimensions of the output dictionary must be known beforehand. In combinatorial problems such as the Travelling Salesman Problem (or for generative DDR), each input sequence may have any number of entities. In addition, the size of the output dictionary depends on the input sequence (i.e., not only on its length). This creates a distinction with the general settings of sequence-to-sequence problems.

To understand the task of (generative) DDR, we shall look at it through the lens of RDF, or more precisely RDFS [5]. It is a logic where each statement consists of a *triple* (s, p, o) interpreted as subject, predicate, and object, some of which can be taken from a pre-defined (RDFS) vocabulary which provides a formal (model-theoretic) semantics. (Symbolic) deductive reasoning over a set of RDFS triples (called an RDF graph or knowledge graph [4]) can be done by means of just 15 so-called *entailment rules*. By iteratively (and exhaustively) applying these rules, it is possible to generate all the logical consequences of the input graph, which is another larger graph, called the *completion* of the input graph. Any two input graphs may have different sizes, and the size of each output depends heavily on size and content of the input.

Pointer Networks, introduced by Vinyals, Fortunato, and Jaitly [48] learn the conditional probability of an output sequence. Each token in the output sequence points to an item in the input sequence. It uses attention to point to a token of the input sequence as part of the probable output sequence. In combinatorial optimization problems such as TSP, Convex-hull, and Delaunay Triangulations, the authors show that the Pointer Network achieves state-of-the-art performance, and therefore seems to be a natural choice to explore generative DDR in this case. Since in RDFS reasoning, the input triples define the different node-edge-node sections of a graph, and the output of the reasoning entails establish previously undefined relationships among nodes using information already provided in the input – Pointer Networks becomes a candidate architecture because its general working principle involves pointing to input

tokens as output tokens. This has prompted the study reported in [49] with positive initial results. However, as we are reporting in more detail below, these results do not appear to stand up to additional scrutiny.[19]

A Pointer Network has an encoder-decoder based architecture with attention mechanism that uses position of tokens in the input sequence to represent the output tokens. In a sample input sequence, positions of different tokens are encoded, while during decoding – the output sequence obtains a part of the input tokens expressed by their respective positions. For more implementation details we refer the reader to [48]. The completion of a graph T is denoted $c(T)$, and the pointer network is trained with input-output pairs $(T, c'(T))$ where $c'(T)$ refers to the indices of tokens from the inputs (i.e., they are *pointed to*). The goal is to compute the conditional probability for each token in the output sequence. Having trained the model, the test graphs are fed to the pointer network and generated sequences are compared against the true inferences to assess the quality of the prediction. The generated sequences represent indices in the corresponding input graph.

Using Apache Jena API and an RDF dataset containing approximately 16K graphs to generate the corresponding inferences for each graph to supervise the model. The dataset is split into 60% training, 20% validation, and 20% test. The dataset is first encoded on the URI level, where a dictionary is created with all the unique URIs. Thereafter, the encoded graphs and inferences are fed to train the model. We have tested with different learning rates between 0.001 and 0.1, maximum length of a graph between 200 and 700. (We have also experimented with encoding the dataset on the token level where each token in the URIs is encoded, but then the the produced output sequences contained a large number of invalid URIs where tokens from different URIs are combined.)

To evaluate the approach, a metric needs to be calculated on a triple level instead of the URI level because a correct prediction needs to have all URIs to be valid to compose a correct triple. In other words, a partial true prediction for a triple does not count towards a right prediction. In our analysis we have found that Pointer Network's capability of producing correct triples is very limited. It resulted in less than 10% accuracy in terms of predicting triples. Unsurprisingly, we have found Precision and Recall to be insignificant as well. Given these negative findings, we also returned to our initial study reported in [49] which showed better results, but these were not on the triple level, but on the token level, and it appears that token padding (to arrive at same-length inputs) played a major role and the results simply did not hold on the triple level. However, any fit-for-purpose ANN-based model must learn the underlying relationships between subjects and objects of a triple when trying to perform deep deductive reasoning.

For instance, in the case of transitive property, where entities A and B have a relation, then B and C have the same relation, the model should be able to understand that A and C have the same relation. However, it should learn this property in a symbol-invariant fashion. Concretely, if the same property is fed into the model with a different set of symbols, ideally the model should be able to identify it too. Our speculation for the Pointer Network failing to perform well is that the model is trying to learn the relationship between symbols, which is rather a difficult job. Consequently, it fails to remember what it has seen before. We believe the Pointer Network approach works on the Travelling Salesman Problem because there is only one relationship to consider amongst all nodes (cities) and the number of considered cities is small. However, in case of DDR, the number of nodes (subjects and objects) is naturally rather large. To make things even more difficult, in DDR there are as many relations to consider as the number of RDFS rules in play, and it appears that the Pointer Network fails to grasp the relationships due to their number being large compared to the overall input size. Finally, due to the fact that it is trying to learn the relationship between "symbols", it fails to remember what it has seen before. This is important because, in RDFS graphs, the order of triples do not matter. In the transitive property example, after the model sees the first part of a transitive property, if the second part comes after a few triples, the model fails to connect the two parts.

5. Conclusions

We have presented the state of the art in DDR, discussed theoretical limitations, and argued that (even within the theoretical limits) it is a hard deep learning problem in practice based on our own prior experiences.

For the theoretical limitations, we have shown that very expressive logics are out of bounds (with 100% accuracy), and even NP-complete ones may be out of bounds (depending on how the P vs. NP problem will be addressable

[19] On a related note, this serves to show that preprints that have not been peer reviewed always need to be taken with caution.

eventually). Of course, this does not mean that investigating DDR for NP-complete or harder logics should not be done, but it does mean that any DDR system for such logics will always be approximate; however it can of course still be useful [50, 51].

In terms of practical hardness, we note that there is currently no well-performing system even for the comparatively simple polynomial logic $\mathcal{EL}^+$. While there is a high accuracy system for OWL 2 RL (which is polynomial in terms of reasoning towards facts, or ABox reasoning), it has low scalability and no transfer or generative ability [18]. For the even less expressive logic RDF(S), we have good systems that are either generative and non-transferable at low scalability [12, 13], or are transferable but non-generative at medium scalability [14]. High scalability systems are either of low accuracy [17] or limited logical expressivity ([11] uses large ABoxes but small TBoxes). We have further reported, in Section 4, on some of our failed attempts to improve the state of the art.

As for practical usefulness, it should be mentioned that RDFS is a prominent logic in the context of Knowledge Graphs, Ontologies, and the Semantic Web [4], and that a generative system capable of transfer and scalability up to non-trivial input sizes with high accuracy (say, even inputs of about 10^6 triples would already find applications – the non-generative transferable system in [14] allows input sizes of 10^3 triples) could have very significant impact and practical applications in knowledge graph-based data management. In particular, such a system – based on deep learning – would be much more easily combinable with other deep learning systems that perform other tasks relevant to the practice of knowledge graph based data management [52].

Deep Deductive Reasoning at scale, and with logics more expressive than RDFS, would of course open up many more opportunities by providing a path to a deep integration of symbolic formal logical reasoning *within* deep learning systems [53] – in contrast to approaches that use deep learning and separate symbolic reasoners in more of a hybrid systems fashion requiring clearly defined interactions between the separate components (e.g., [54]), and corresponding mapping between vector and symbolic components.

As such, DDR remains a highly interesting area of research, where practical applications seem within grasp. However we have also seen that the right approaches have not been found yet. To advance, it requires new ideas, and investigations on a broad front, with many different deep learning architectures and approaches.

Acknowledgement The authors acknowledge partial funding by the National Science Foundation under grants 2033521 *KnowWhereGraph*, 2119753 *BioWRAP* and 2333532 *EduGate*.

References

[1] R.J. Brachman and H.J. Levesque, *Knowledge Representation and Reasoning*, Elsevier, 2004. ISBN 978-1-55860-932-7.

[2] E. Dantsin, T. Eiter, G. Gottlob and A. Voronkov, Complexity and expressive power of logic programming, *ACM Comput. Surv.* **33**(3) (2001), 374–425. doi:10.1145/502807.502810.

[3] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi and P.F. Patel-Schneider (eds), *The Description Logic Handbook: Theory, Implementation, and Applications*, 2nd edn, Cambridge University Press, 2007.

[4] P. Hitzler, A review of the semantic web field, *Commun. ACM* **64**(2) (2021), 76–83. doi:10.1145/3397512.

[5] P. Hitzler, M. Krötzsch and S. Rudolph, *Foundations of Semantic Web Technologies*, Chapman and Hall/CRC Press, 2010.

[6] W.S. McCulloch and W.H. Pitts, A Logical Calculus of the Ideas Immanent in Nervous Activity, in: *The Philosophy of Artificial Intelligence*, M.A. Boden, ed., Oxford Readings in Philosophy, Oxford University Press, 1990, pp. 22–39.

[7] P. Hitzler, S. Hölldobler and A.K. Seda, Logic programs and connectionist networks, *Journal of Applied Logic* **2**(3) (2004), 245–272.

[8] M. Ebrahimi, A. Eberhart, F. Bianchi and P. Hitzler, Towards bridging the neuro-symbolic gap: deep deductive reasoners, *Appl. Intell.* **51**(9) (2021), 6326–6348. doi:10.1007/s10489-020-02165-6.

[9] P. Hitzler and A.K. Seda, *Mathematical Aspects of Logic Programming Semantics*, Studies in Informatics, Chapman and Hall/CRC Press, 2011.

[10] R. Ferreira, C. Lopes, R. Gonçalves, M. Knorr, L. Krippahl and J. Leite, Deep neural networks for approximating stream reasoning with C-SPARQL, in: *Progress in Artificial Intelligence: 20th EPIA Conference on Artificial Intelligence, EPIA 2021, Virtual Event, September 7–9, 2021, Proceedings 20*, Springer, 2021, pp. 338–350.

[11] X. Zhu, B. Liu, Z. Ding, C. Zhu and L. Yao, Approximate Ontology Reasoning for Domain-Specific Knowledge Graph based on Deep Learning, in: *2021 7th International Conference on Big Data and Information Analytics (BigDIA)*, 2021, pp. 172–179. doi:10.1109/BigDIA53151.2021.9619694.

[12] B. Makni and J. Hendler, Deep learning for noise-tolerant RDFS reasoning, *Semantic Web* **10**(5) (2019), 823–862.

[13] B. Makni, I. Abdelaziz and J. Hendler, Explainable deep RDFS reasoner, *arXiv preprint* **abs/2002.03514** (2020).

[14] M. Ebrahimi, M.K. Sarker, F. Bianchi, N. Xie, A. Eberhart, D. Doran, H. Kim and P. Hitzler, Neuro-Symbolic Deductive Reasoning for Cross-Knowledge Graph Entailment, in: *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering*, 2021.

[15] T. Sato, A. Takemura and T. Inoue, Towards end-to-end ASP computation, *arXiv preprint* **abs/2306.06821** (2023).

[16] A. Eberhart, M. Ebrahimi, L. Zhou, C. Shimizu and P. Hitzler, Completion Reasoning Emulation for the Description Logic EL+, in: *AAAI Spring Symposium on Combining Machine Learning and Knowledge Engineering in Practice (AAAI-MAKE) Volume I*, A. Martin, K. Hinkelmann, H. Fill, A. Gerber, D. Lenat, R. Stolle and F. van Harmelen, eds, CEUR Workshop Proceedings, Vol. 2600, CEUR-WS.org, 2020.

[17] B. Mohapatra, A. Bhattacharya, S. Bhatia, R. Mutharaju and G. Srinivasaraghavan, EmEL-V: EL Ontology Embeddings for Many-to-Many Relationships, Preprint, available from https://www.semantic-web-journal.net/content/emel-v-el-ontology-embeddings-many-many-relationships.

[18] P. Hohenecker and T. Lukasiewicz, Ontology reasoning with deep neural networks, *Journal of Artificial Intelligence Research* **68** (2020), 503–540.

[19] D.M. Adamski and J. Potoniec, Reason-able embeddings: Learning concept embeddings with a transferable neural reasoner, *Semantic Web* (2023). doi:10.3233/SW-233355.

[20] X. Zhu, B. Liu, C. Zhu, Z. Ding and L. Yao, Approximate Reasoning for Large-Scale ABox in OWL DL Based on Neural-Symbolic Learning, *Mathematics* **11**(3) (2023), 495.

[21] F. Bianchi and P. Hitzler, On the Capabilities of Logic Tensor Networks for Deductive Reasoning, in: *Proceedings of the AAAI Spring Symposium on Combining Machine Learning with Knowledge Engineering (AAAI-MAKE)*, A. Martin, K. Hinkelmann, A. Gerber, D. Lenat, F. van Harmelen and P. Clark, eds, CEUR Workshop Proceedings, Vol. 2350, CEUR-WS.org, 2019.

[22] B. Mohapatra, S. Bhatia, R. Mutharaju and G. Srinivasaraghavan, EmELvar: A NeuroSymbolic Reasoner for the EL++ Description Logic, in: *SemREC 2021, Semantic Reasoning Evaluation Challenge 2021, Proceedings of the Semantic Reasoning Evaluation Challenge (SemREC 2021) co-located with the 20th International Semantic Web Conference (ISWC 2021). Virtual Event, October 27th, 2021*, Vol. 3123, G. Singh, R. Mutharaju and P. Kapanipathi, eds, CEUR Workshop Proceedings, 2021, pp. 44–51.

[23] A. Tomasic, O.J. Romero, J. Zimmerman and A. Steinfeld, Propositional Reasoning via Neural Transformer Language Models (2021).

[24] R. Evans, D. Saxton, D. Amos, P. Kohli and E. Grefenstette, Can neural networks understand logical entailment?, *arXiv preprint arXiv:1802.08535* (2018).

[25] G. Irving, C. Szegedy, A.A. Alemi, N. Eén, F. Chollet and J. Urban, Deepmath-deep sequence models for premise selection, *Advances in neural information processing systems* **29** (2016).

[26] C. Kaliszyk, F. Chollet and C. Szegedy, Holstep: A machine learning dataset for higher-order logic theorem proving, *arXiv preprint arXiv:1703.00426* (2017).

[27] S. Loos, G. Irving, C. Szegedy and C. Kaliszyk, Deep network guided proof search, *arXiv preprint arXiv:1701.06972* (2017).

[28] T. Sekiyama and K. Suenaga, Automated proof synthesis for propositional logic with deep neural networks, *arXiv preprint arXiv:1805.11799* (2018).

[29] B. Bünz and M. Lamm, Graph neural networks and boolean satisfiability, *arXiv preprint arXiv:1702.03592* (2017).

[30] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura and D.L. Dill, Learning a SAT solver from single-bit supervision, *arXiv preprint arXiv:1802.03685* (2018).

[31] J. Berner, P. Grohs, G. Kutyniok and P. Petersen, *The Modern Mathematics of Deep Learning*, in: *Mathematical Aspects of Deep Learning*, P. Grohs and G. Kutyniok, eds, Cambridge University Press, 2022, pp. 1–111–.

[32] G. Yang, Tensor Programs I: Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes, *CoRR* **abs/1910.12478** (2019).

[33] H. Vollmer, Introduction to Circuit Complexity, in: *Texts in Theoretical Computer Science. An EATCS Series*, 1999.

[34] R. Williams, Algorithms for Circuits and Circuits for Algorithms, 2014, pp. 248–261. ISBN 978-1-4799-3626-7. doi:10.1109/CCC.2014.33.

[35] K.P. Murphy, *Probabilistic Machine Learning: An introduction*, MIT Press, 2022.

[36] S. Bader, P. Hitzler, S. Hölldobler and A. Witzel, A Fully Connectionist Model Generator for Covered First-Order Logic Programs, in: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, M.M. Veloso, ed., 2007, pp. 666–671. http://ijcai.org/Proceedings/07/Papers/106.pdf.

[37] R. Chen, V. Kabanets and J. Kinne, Lower Bounds Against Weakly-Uniform Threshold Circuits, *Algorithmica* **70**(1) (2014), 47–75. doi:10.1007/s00453-013-9823-y.

[38] D. Erhan, Y. Bengio, A. Courville and P. Vincent, Visualizing Higher-Layer Features of a Deep Network, *Technical Report, Université de Montréal* (2009).

[39] A. Meier, Generalized Complexity of ALC Subsumption, *CoRR* **abs/1205.0722** (2012).

[40] C. Lautemann, BPP and the Polynomial Hierarchy, *Inf. Process. Lett.* **17** (1983), 215–217.

[41] S.P. Vadhan, Pseudorandomness, *Foundations and Trends in Theoretical Computer Science* **7**(1–3) (2012), 1–336. doi:10.1561/0400000010.

[42] D. Angluin, Queries and concept learning, *Machine Learning* **2** (1988), 319–342.

[43] A. Kawachi, Circuit Lower Bounds from Learning-theoretic Approaches, *Theoretical Computer Science* **733** (2018). doi:10.1016/j.tcs.2018.04.038.

[44] A.R. Klivans, P. Kothari and I.C. Oliveira, Constructing Hard Functions from Learning Algorithms, *Electron. Colloquium Comput. Complex.* **TR13** (2013).

[45] T.R. Besold, A.S. d'Avila Garcez, S. Bader, H. Bowman, P.M. Domingos, P. Hitzler, K. Kühnberger, L.C. Lamb, P.M.V. Lima, L. de Penning, G. Pinkas, H. Poon and G. Zaverucha, Neural-Symbolic Learning and Reasoning: A Survey and Interpretation, in: *Neuro-Symbolic Artificial Intelligence: The State of the Art*, P. Hitzler and M.K. Sarker, eds, Frontiers in Artificial Intelligence and Applications, Vol. 342, IOS Press, 2021, pp. 1–51. doi:10.3233/FAIA210348.

[46] Y. Kazakov, M. Krötzsch and F. Simančík, ELK: a reasoner for OWL EL ontologies, *System Description* (2012).

[47] S. Hochreiter and J. Schmidhuber, LSTM can solve hard long time lag problems, in: *Advances in neural information processing systems*, 1997, pp. 473–479.

[48] O. Vinyals, M. Fortunato and N. Jaitly, Pointer Networks **28** (2015).

[49] M. Ebrahimi, A. Eberhart and P. Hitzler, On the Capabilities of Pointer Networks for Deep Deductive Reasoning, *CoRR* **abs/2106.09225** (2021).

[50] P. Hitzler and F. van Harmelen, A reasonable Semantic Web, *Semantic Web* **1**(1–2) (2010), 39–44. doi:10.3233/SW-2010-0010.

[51] S. Rudolph, T. Tserendorj and P. Hitzler, What Is Approximate Reasoning?, in: *Web Reasoning and Rule Systems, Second International Conference, RR*, D. Calvanese and G. Lausen, eds, Lecture Notes in Computer Science, Vol. 5341, Springer, 2008, pp. 150–164. doi:10.1007/978-3-540-88737-9_12.

[52] P. Hitzler, F. Bianchi, M. Ebrahimi and M.K. Sarker, Neural-symbolic integration and the Semantic Web, *Semantic Web (Preprint)* (2019), 1–9.

[53] S. Artemov, H. Barringer, A. d'Avila Garcez, L.C. Lamb and J. Woods, *We Will Show Them in Honour of Dov Gabbay* **2** (2005), 167–194.

[54] R. Manhaeve, G. Marra, T. Demeester, S. Dumancic, A. Kimmig and L.D. Raedt, Neuro-Symbolic AI = Neural + Logical + Probabilistic AI, in: *Neuro-Symbolic Artificial Intelligence: The State of the Art*, P. Hitzler and M.K. Sarker, eds, Frontiers in Artificial Intelligence and Applications, Vol. 342, IOS Press, 2021, pp. 173–191. doi:10.3233/FAIA210354.