Learning Explanatory Rules from Noisy Data (Extended Abstract)*

Richard Evans[1], **Edward Grefenstette**[1]
[1] DeepMind
richardevans@google.com, etg@google.com

Abstract

Artificial Neural Networks are powerful function approximators capable of modelling solutions to a wide variety of problems, both supervised and unsupervised. As their size and expressivity increases, so too does the variance of the model, yielding a nearly ubiquitous overfitting problem. Although mitigated by a variety of model regularisation methods, the common cure is to seek large amounts of training data—which is not necessarily easily obtained—that sufficiently approximates the data distribution of the domain we wish to test on. In contrast, logic programming methods such as Inductive Logic Programming offer an extremely data-efficient process by which models can be trained to reason on symbolic domains. However, these methods are unable to deal with the variety of domains neural networks can be applied to: they are not robust to noise in or mislabelling of inputs, and perhaps more importantly, cannot be applied to non-symbolic domains where the data is ambiguous, such as operating on raw pixels. In this paper, we propose a Differentiable Inductive Logic framework (∂ILP), which can not only solve tasks which traditional ILP systems are suited for, but shows a robustness to noise and error in the training data which ILP cannot cope with. Furthermore, as it is trained by backpropagation against a likelihood objective, it can be hybridised by connecting it with neural networks over ambiguous data in order to be applied to domains which ILP cannot address, while providing data efficiency and generalisation beyond what neural networks on their own can achieve.

1 Introduction

Inductive Logic Programming (ILP) is a collection of techniques for constructing logic programs from examples. Given a set of positive examples, and a set of negative examples, an ILP system constructs a logic program that entails all the positive examples but does not entail any of the negative examples. From a machine learning perspective, an ILP system can be interpreted as implementing a rule-based binary classifier over examples, mapping each example to an evaluation of its truth or falsehood according to the axioms provided to the system, alongside new rules inferred by the system during training.

ILP has a number of appealing features. First, the learned program is an explicit symbolic structure that can be inspected, understood, and verified. Second, ILP systems tend to be impressively data-efficient, able to generalise well from a small handful of examples. The reason for this data-efficiency is that ILP imposes a strong language bias on the sorts of programs that can be learned: a short general program will be preferred to a program consisting of a large number of special-case ad-hoc rules that happen to cover the training data. Third, ILP systems support continual and transfer learning. The program learned in one training session, being declarative and free of side-effects, can be copied and pasted into the knowledge base before the next training session, providing an economical way of storing learned knowledge.

The main disadvantage of traditional ILP systems is their inability to handle noisy, erroneous, or ambiguous data. If the positive or negative examples contain any mislabelled data, these systems will not be able to learn the intended rule. [De Raedt and Kersting, 2008] discuss this issue in depth, stressing the importance of building systems capable of applying relational learning to uncertain data.

A key strength of neural networks is that they are robust to noise and ambiguity. One way to overcome the brittleness of traditional ILP systems is to reimplement them in a robust connectionist framework.

Recently, a different approach to program induction has emerged from the deep learning community [Graves *et al.*, 2014; Reed and de Freitas, 2015; Neelakantan *et al.*, 2015; Kaiser, 2015; Andrychowicz and Kurach, 2016; Graves *et al.*, 2016]. These neural network-based systems do not construct an explicit symbolic representation of a program. Instead, they learn an implicit procedure (distributed in the weights of the net) that produces the intended results. These approaches take a relatively low-level model of computation—a model that is much "closer to the metal" than the Horn clauses used in ILP—and produce a differentiable implementation of that low-level model. The implicit procedure that is learned is a

*This paper is an extended abstract of an article in the Journal of Artificial Intelligence Research [Evans and Grefenstette, 2018].

way of operating within that low-level model of computation (by moving the tape head, reading and writing in the case of differentiable Turing machines; by pushing and popping in the case of differentiable pushdown automata).

There are two appealing features of this differentiable approach to program induction. First, these systems are robust to noise. Unlike ILP, a neural system will tolerate some bad (mis-labeled) data. Second, a neural program induction system can be provided with fuzzy or ambiguous data (from a camera, for example). Unlike traditional ILP systems (which have to be fed crisp, symbolic input), a differentiable induction system can start with raw, un-preprocessed pixel input.

However, the neural approaches to program induction have two disadvantages when compared to ILP. First, the implicit procedure learned by a neural network is not inspectable or human-readable. It is notoriously hard to understand what it has learned, or to what extent it has generalised beyond the training data. Second, the performance of these systems tails off sharply when the test data are significantly larger than the training data: if we train the neural system to add numbers of length 10, they may also be successful when tested on numbers of length 20. But if we test them on numbers of length 100, the performance deteriorates [Kaiser, 2015; Reed and de Freitas, 2015]. General-purpose neural architectures, being universal function approximators, produce solutions with high variance. There is an ever-present danger of over-fitting.

This paper proposes a system that addresses the limits of connectionist systems and ILP systems, and attempts to combine the strengths of both. **Differentiable Inductive Logic Programming** (∂ILP) is a reimplementation of ILP in an an end-to-end differentiable architecture. It attempts to combine the advantages of ILP with the advantages of the neural network-based systems: a data-efficient induction system that can learn explicit human-readable symbolic rules, that is robust to noisy and ambiguous data, and that does not deteriorate when applied to unseen test data. The central component of this system is a differentiable implementation of deduction through forward chaining on definite clauses. We reinterpret the ILP task as a binary classification problem, and we minimise cross-entropy loss with regard to ground-truth boolean labels during training.

2 ∂ILP

An **ILP problem** is a tuple $(\mathcal{B}, \mathcal{P}, \mathcal{N})$ of ground atoms, where $\mathcal{B}$ is a set of background assumptions, $\mathcal{P}$ is a set of positive instances, and $\mathcal{N}$ is a set of negative instances. Given an ILP problem $(\mathcal{B}, \mathcal{P}, \mathcal{N})$, a **solution** is a set R of definite clauses such that:

- $\mathcal{B}, R \models \gamma$ for all $\gamma \in \mathcal{P}$
- $\mathcal{B}, R \not\models \gamma$ for all $\gamma \in \mathcal{N}$

Induction is finding a set of rules R such that, when they are applied deductively to the background assumptions $\mathcal{B}$, they produce the desired conclusions.

One approach to ILP is to transform the induction problem into a satisfiability problem. Given a program template (a high-level specification of a grammar of possible programs), generate a set C of clauses satisfying that template. Define a set of flags Φ, where each flag indicates whether a particular clause in C is to be used in the program. Now a SAT solver can be used to find a truth-assignment to the propositions in Φ, and we can extract the induced rules from the subset of propositions in Φ that are set to True.

Our approach is an extension to this established approach to ILP: our contribution is to provide, via a continuous relaxation of satisfiability, a *differentiable implementation* of this architecture. This allows us to apply gradient descent to learn which clauses to turn on and off, even in the presence of noisy or ambiguous data.

Instead of the discrete semantics in which ground atoms are mapped to $\{False, True\}$, we now use a continuous semantics which maps atoms to the real unit interval $[0, 1]$. Instead of using Boolean flags to choose a discrete subset of clauses, we now use continuous weights to determine a probability distribution over clauses.

Given a set G of n ground atoms, a **valuation** is a vector $[0, 1]^n$ mapping each ground atom $\gamma_i \in G$ to the real unit interval.

Given the sets $\mathcal{P}$ and $\mathcal{N}$ of positive and negative examples, we form a set Λ of atom-label pairs:

$$\Lambda = \{(\gamma, 1) \mid \gamma \in \mathcal{P}\} \cup \{(\gamma, 0) \mid \gamma \in \mathcal{N}\}$$

Each pair (γ, λ) indicates whether atom γ is in $\mathcal{P}$ (when $\lambda = 1$) or $\mathcal{N}$ (when $\lambda = 0$). This can be thought of as a dataset, used to learn a binary classifier that maps atoms γ to their truth or falsehood.

Now given an ILP problem $(\mathcal{B}, \mathcal{P}, \mathcal{N})$, a language $\mathcal{L}$, a program template Π and a set of clause-weights W, we construct a differentiable model that implements the conditional probability of λ for a ground atom γ:

$$p(\lambda \mid \gamma, W, \Pi, \mathcal{L}, \mathcal{B})$$

Here, W represents our degree of belief in each clause (so W is a continuous representation of our flags Φ above). We want our predicted label $p(\lambda \mid \gamma, W, \Pi, \mathcal{L}, \mathcal{B})$ to match the actual label λ in the pair (γ, λ) we sample from Λ. We wish, in other words, to minimise the expected negative log likelihood when sampling uniformly (γ, λ) pairs from Λ. To calculate the probability of the label λ given the atom γ, we infer the consequences of applying the rules to the background facts (using T steps of forward chaining). In Figure 1 below, these consequences are called the "Conclusion Valuation". Then, we extract λ as the probability of γ in this valuation.

To calculate $p(\lambda \mid \gamma, W, \Pi, \mathcal{L}, \mathcal{B})$, we perform the following steps (see Figure 1):

- transform the background $\mathcal{B}$ into an *initial valuation* representing our premises
- *infer* the consequences of applying our clauses (according to our clause weights W) using T steps of forward-chaining inference
- look at the final results, the *conclusion valuation*, after T steps of inference
- *extract* the truth-value assigned to γ in the conclusion valuation; call the truth-value $\hat{\lambda}$

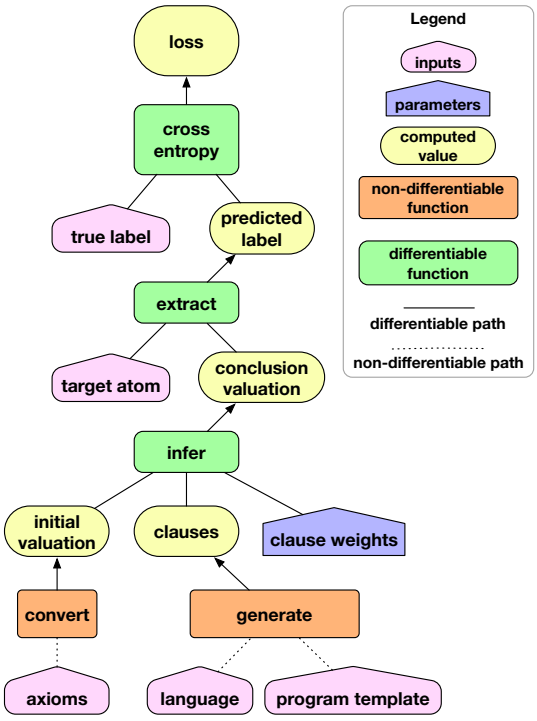


Figure 1: The ∂ILP Architecture.

G	$\mathbf{a}_0$	$\mathcal{F}_c(\mathbf{a_0})$	$\mathbf{a}_1$	$\mathcal{F}_c(\mathbf{a}_1)$
$p(a)$	0.0	0.1	0.2	0.7
$p(b)$	0.0	0.3	0.9	0.4
$q(a)$	0.1	0.0	0.7	0.0
$q(b)$	0.3	0.0	0.4	0.0
$\bot$	0.0	0.0	0.0	0.0

Table 1: Applying $c = p(X) \leftarrow q(X)$, treated as a function $\mathcal{F}_c$, to valuations $\mathbf{a}_0$ and $\mathbf{a}_1$

- compare the predicted value $\hat{\lambda}$ with the ground truth label λ, and compute the binary cross-entropy *loss*
- propagate the error back through the network, updating the *clause weights*

This approach relies on our differentiable implementation of forward chaining inference. The $f_{infer} : [0,1]^n \times C \times W \times \mathbb{N} \rightarrow [0,1]^n$ function is where all the heavy-lifting takes place. It performs T steps of forward-chaining inference using the generated clauses C, amalgamating the various conclusions together using the clause weights W. It is described in detail in the paper.

The central idea behind our differentiable implementation of inference is that each clause c induces a differentiable function $\mathcal{F}_c : [0,1]^n \rightarrow [0,1]^n$ on valuations. Consider, for example, the clause c:

$$p(X) \leftarrow q(X)$$

Table 1 shows the results of applying the corresponding function $\mathcal{F}_c$ to two valuations on the set $G = \{p(a), p(b), q(a), q(b), \bot\}$ of ground atoms. We can automatically generate, from each clause c, a differentiable function $\mathcal{F}_c$ on valuations that implements a single step of forward chaining inference using c.

We combine the valuations from the different clauses together, using the softmax of the weights W, to produce a single valuation for each time-step. In our differentiable implementation of forward chaining, we implement conjunction using the product t-norm, and disjunction using the probabilistic sum.

3 Experimental Results

We implemented our model in TensorFlow [Abadi *et al.*, 2016] and tested it with three types of experiment. First, we used standard symbolic ILP tasks, where ∂ILP is given discrete error-free input. Second, we modified the standard symbolic ILP tasks so that a certain proportion of the positive and negative examples are wilfully mis-labelled. Third, we tested it with fuzzy, ambiguous data, connecting ∂ILP to the output of a pretrained convolution neural network that classifies MNIST digits.

3.1 ILP Benchmark Tasks

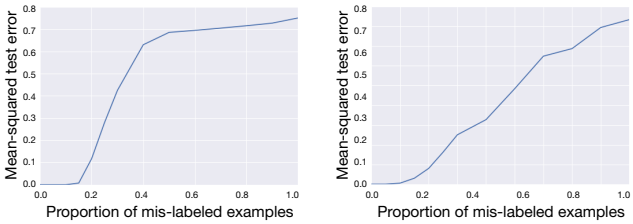
We tested ∂ILP on 20 ILP tasks, taken from four domains: arithmetic, lists, group-theory, and family tree relations. Some of the arithmetic examples appeared in [Cropper and Muggleton, 2016]. The list examples are used in [Feser *et al.*, 2015]. The problems are mostly non-trivial in that they require recursion and predicate invention (the construction of auxiliary helper functions).

3.2 Dealing with Mislabelled Data

Standard ILP methods try to find a program that satisfies all the positive examples and fails to satisfy any of the negative examples. These requirements are strict and there is no room for error. If one of the positive examples $\mathcal{P}$ or negative examples $\mathcal{N}$ is mislabelled, then standard ILP methods cannot find the intended program.

∂ILP, by contrast, is minimising a loss rather than trying to satisfy a strict requirement, so is capable of handling mislabelled data. We tested this by adding a global variable ρ, the proportion of mislabelled data, and varying it between 0 and 1 in small increments. At the beginning of each experiment, a proportion ρ of the atoms from $\mathcal{P}$ and $\mathcal{N}$ are sampled, without replacement, and transferred to the other group. Note that the algorithm sees the same consistently mislabelled examples over all training steps, rather than letting the mis-labelling vary between training steps.

The results are shown in Figure 2. Each cell shows the mean squared test error for a particular task with a particular proportion of mislabelled data. The results show that ∂ILP is robust to mislabelled data. The mean-squared error degrades gracefully as the proportion of mislabelled data increases. This is in stark contrast to a traditional ILP system, where test error increases sharply as soon as there is a single piece of mislabelled training data. At 10% mislabelled data, ∂ILP still finds a perfect answer in 5 out of 6 of the tasks. In some of the tasks, ∂ILP still finds good answers with 20% or 30% mislabelled data.



(a) Predecessor (b) Connectedness

Figure 2: Mean squared test error as a function of the proportion of mislabelled data

3.3 Dealing with Ambiguous Data

Standard ILP assumes that atoms are either true or false. There is no room for vagueness or indecision. ∂ILP, by contrast, uses a continuous semantics which maps atoms to the real unit interval $[0, 1]$. It was designed to reason with cases where we are unsure if a proposition is true or false.

We tested ∂ILP's ability to handle ambiguous input by connecting it to the results of a pre-trained convolution neural networks (a "convnet"). In one experiment, we trained it to detect less-than on 28×28 MNIST images. In this task, the system is given a *pair* of images (left and right) each training step. The training label is a 1 if the number represented by the left image is less than the number represented by the right image.

Each image is fed into the pre-trained convnet. The logits from the convnet are transformed, via soft-max, into a probability distribution over a set of atoms. We use $image1(X)$ to represent that the left image has numeric value X, and $image2(X)$ to represent that the right image has numeric value X. For example, if the left image represents a "2", the valuation produced by the convnet might be:

$$image1(0) \mapsto 0.0 \qquad image1(3) \mapsto 0.0$$
$$image1(1) \mapsto 0.0 \qquad image1(4) \mapsto 0.1$$
$$image1(2) \mapsto 0.8 \qquad image1(5) \mapsto 0.1$$

We only consider integers 0-5 to keep the example simple. See Figure 3.

The $target$ predicate in this case is a nullary predicate: $target$ is true if the left image represents a number that is less than the number represented by the right image.

This task requires synthesising two auxiliary predicates, one of which is recursive. One of the solutions that ∂ILP found is:

$$\begin{aligned} target() &\leftarrow image2(X), pred1(X) \\ pred1(X) &\leftarrow image1(Y), pred2(Y, X) \\ pred2(X, Y) &\leftarrow succ(X, Y) \\ pred2(X, Y) &\leftarrow pred2(Z, Y), pred2(X, Z) \end{aligned}$$

Here, $pred2$ is the synthesised recursive less-than relation.

4 Conclusions

Our main contribution is a differentiable implementation of Inductive Logic Programming. This model is able to learn moderately complex programs: programs requiring multiple invented predicates and recursive clauses. Unlike traditional symbolic ILP systems, our model is robust to mislabelled training data. Furthermore, unlike traditional symbolic ILP systems, our model can handle ambiguous data in domains where artificial neural networks are traditionally the tool of choice. Unlike neural networks on their own, it offers the same data efficiency and generalisation guarantees as traditional ILP systems, extended to new domains. In doing so, this work opens the door to an end-to-end differentiable system that is capable of learning perceptual and inferential rules *simultaneously*.

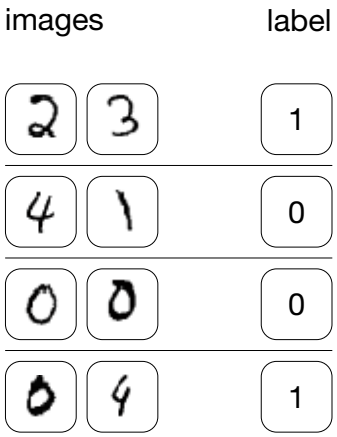


Figure 3: Learning Less-Than from Raw Pixel Images

We evaluated this model on 20 symbolic ILP tasks, and showed that it can consistently solve problems traditional ILP systems excel on. We tested its tolerance to mislabelled data, and demonstrated it was able to learn good models even with 20% mislabelled training examples. We also tested it with ambiguous data, connecting the output of a pretrained convnet to the inputs of the induction system, and it was able to learn effectively and consistently generalise.

There are two main limitations of our framework that we wish to address in future work. First, we currently use program templates to constrain the set of programs that are searched through. Second, the experiments on ambiguous data relied on pre-trained neural network classifiers. We "baked in" the knowledge that the MNIST images were clustered into ten classes corresponding to the ten digits. Preliminary experiments showed that pre-training only the convolutional layers of a vision network, and jointly learning the projection into symbols and reasoning over such symbols, yielded satisfactory and interpretable results for our MNIST-based tasks, but full end-to-end joint training of vision modules and ∂ILP requires further research. Future work will focus on sensible pretraining strategies for these neural networks, which do not obviously bias them towards the specific problems into which they will be integrated.

References

[Abadi *et al.*, 2016] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[Andrychowicz and Kurach, 2016] Marcin Andrychowicz and Karol Kurach. Learning efficient algorithms with hierarchical attentive memory. *arXiv preprint arXiv:1602.03218*, 2016.

[Bader *et al.*, 2008] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler. Connectionist model generation: A first-order approach. *Neurocomputing*, 71(13):2420–2432, 2008.

[Corapi *et al.*, 2010] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming as abductive search. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 7. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[Corapi *et al.*, 2011] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In *International Conference on Inductive Logic Programming*, pages 91–97. Springer, 2011.

[Cropper and Muggleton, 2016] Andrew Cropper and Stephen H Muggleton. Learning higher-order logic programs through abstraction and invention. In *Proceedings of the 25th International Joint Conference Artificial Intelligence (IJCAI 2016). IJCAI*, 2016.

[Cropper *et al.*, 2015] Andrew Cropper, Alireza Tamaddoni-Nezhad, and Stephen H Muggleton. Meta-interpretive learning of data transformation programs. In *International Conference on Inductive Logic Programming*, pages 46–59. Springer, 2015.

[De Raedt and Kersting, 2008] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming*, pages 1–27. Springer, 2008.

[De Raedt *et al.*, 2016] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(2):1–189, 2016.

[Evans and Grefenstette, 2018] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[Feser *et al.*, 2015] John K Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. In *ACM SIGPLAN Notices*, volume 50, pages 229–239. ACM, 2015.

[Garcez *et al.*, 2012] Artur S d'Avila Garcez, Krysia Broda, and Dov M Gabbay. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media, 2012.

[Garcez *et al.*, 2014] Artur S d'Avila Garcez, Dov M Gabbay, and Luis C Lamb. A neural cognitive model of argumentation with application to legal inference and decision making. *Journal of Applied Logic*, 12(2):109–127, 2014.

[Garcez *et al.*, 2015] Garcez, Besold, de Raedt, Foldiak, Hitzler, Icard, Kuhnberger, Lamb, Miikkulainen, and Silver. Neural-symbolic learning and reasoning: contributions and challenges. *Proceedings of the AAAI Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches, Stanford*, 2015.

[Graves *et al.*, 2014] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[Graves *et al.*, 2016] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[Hölldobler *et al.*, 1999] Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11(1):45–58, 1999.

[Kaiser, 2015] Lukasz Kaiser. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

[Law *et al.*, 2014] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In *European Workshop on Logics in Artificial Intelligence*, pages 311–325. Springer, 2014.

[Muggleton *et al.*, 2014] Stephen H Muggleton, Dianhuan Lin, Niels Pahlavi, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94(1):25–49, 2014.

[Muggleton *et al.*, 2015] Stephen H Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.

[Neelakantan *et al.*, 2015] Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

[Reed and de Freitas, 2015] Scott Reed and Nando de Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

[Rocktäschel and Riedel, 2017] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, pages 3791–3803, 2017.

[Serafini and Garcez, 2016] Luciano Serafini and Artur Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*, 2016.