A differentiable approach to inductive logic programming

Fan Yang
Carnegie Mellon University
fanyang1@cs.cmu.edu

Zhilin Yang
Carnegie Mellon University
zhiliny@cs.cmu.edu

William W. Cohen
Carnegie Mellon University
wcohen@cs.cmu.edu

Abstract

Recent work in neural abstract machines has proposed many useful techniques to learn sequences of applications of discrete but differentiable operators. These techniques allow us to model traditionally procedural problems using neural networks. In this work, we are interested in using neural networks to learn to perform logic reasoning. We propose a model that has access to differentiable operators which can be composed to perform reasoning. These differentiable reasoning operators were first introduced in TensorLog, a recently proposed probabilistic deductive database. Equipped with a model than can perform logic reasoning, we further investigate the task of inductive logic programming.

1 Introduction

Inductive logic programming (ILP) [1] refers to a broad class of problems that aim to find logic rules that model the observed data. The observed data usually contains background knowledge and examples, typically in the form of database relations or knowledge graphs. Inductive logic programming is often combined with use of probabilistic logics, and is a useful technique for knowledge base completion and other relational learning tasks [2]. However, past inductive logic programming approaches have involved discrete search through the space of possible structures [3]. This search is expensive, and difficult to integrate with neural networks.

TensorLog [4] is a recently proposed probabilistic deductive database. The major contribution of TensorLog is that it provides a principled way to define *differentiable* reasoning processes. TensorLog reduces a broad class of logic programs to inferences made on factor graphs, with logical variables encoded as multinomials over the domain of database constants and logical relations encoded as factors. By "unrolling" the factor graph inference into a sequence of message passing steps, one can obtain a differentiable function that answers a particular class of local queries against a database.

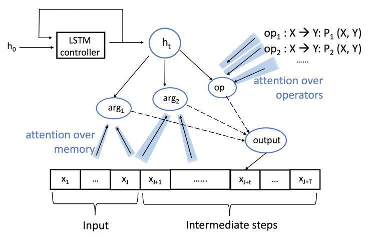
The key idea in this paper is to learn a neural network controller that composes TensorLog's message passing operators sequentially. Since the operations are differentiable, and can support reasoning, the resulting "neural ILP" system can learn logic programs. These logic programs can be interpreted as induced logic rules.

2 A neural network model for inductive logic programming

Even though the reasoning process can be made differentiable, it is still not an easy task to design a neural network model for inductive logic programming. Many challenges remain, such as the interface between the neural network controller and the reasoning operators, the representation of logic rules, and the interface to dynamic memory.

To address these problems, we adapt techniques developed in neural abstract machines literature, including Neural Programmer [5], Memory Networks [6], Differentiable Neural Computer [7], and

1st Workshop on Neural Abstract Machines & Program Induction (NAMPI), @NIPS 2016, Barcelona, Spain.



h_0
LSTM controller
h_t
$op_1 : X \rightarrow Y: P_1 (X, Y)$
$op_2 : X \rightarrow Y: P_2 (X, Y)$
......
attention over operators
op
arg_1
arg_2
attention over memory
output
x_1 ... x_J x_{J+1} ...... x_{J+t} ... x_{J+T}
Input
Intermediate steps

Figure 1: An overview of the model

Queries	Induced logic rules
father(z, x)	husband(z, y) ∧ mother(y, x)
sister(z, x)	daughter(z, y) ∧ mother(y, x)
husband(z, x)	father(z, y) ∧ son(y, x)
uncle(z, x)	brother(z, y) ∧ (father(y, x) ∨ mother(y, x))
nephew(z, x)	son(z, y) ∧ (brother(y, x) ∨ sister(y, x))
niece(z, x)	sister(z, y) ∧ nephew(y, x)

Table 1: Experiment results

attention mechanism [8], as well as recent work on learning procedural tasks, such as Neural Theorem Prover [9].

The main part of the model is a recurrent neural network acting as a controller, and the controller has access to differentiable operators and memory. Figure 1 provides an overview of the model. The controller takes in the previous state and produces a new state to select three things via attention mechanisms: the next operator, the first input to that operator, and the second input (if there is one). After selection, the operator is applied to the arguments and the output is stored in the next available memory slot.

Intuitively, the operators correspond to the mathematical operations used in message passings. The operators map distributions over database constants to a new distribution: for example, one operator $\text{op}_{\texttt{relation}}$ might map a distribution over set $\mathcal{X}$ to a distribution over $\{y : \texttt{relation}(x, y), x \in \mathcal{X}\}$, the set of all database constants that satisfy this particular relation with $\mathcal{X}$. There are a constant number of operators for each database predicate (i.e. relation), as well as some additional operations for set union and intersections.

The memory has two parts. The first part contains embeddings of inputs to query. In the case of argument retrieval query [1], the inputs to the query are the database constants and predicates. The embeddings can be learned jointly during training [10]. The second part of the memory is used to store intermediate operator outputs, which can be thought of as the messages being passed around in graphical models setting.

Once trained, the model can be used to induce logic rules that connect relations in the database. For example, if we want to know how other relations imply the relation `uncle`, we can ask the trained model a query that involves `uncle`. The trained model will compose operators to perform reasoning about the query. We can then read off the operators that have the most attention at each time step. If these operators correspond to `brother` and `father`, we know that the model has correctly learned to use these two relations to reason about `uncle`.

3 Experiments

We experiment with an European royal family dataset. This dataset contains 3007 entities, 28373 facts, and 12 relations. The relations are father, mother, husband, wife, son, daughter, brother, sister, uncle, aunt, nephew, niece. We split the entities into train and test sets.

To learn to induce logic rules about a specific relation `R`, we let the database consists of facts about the other relations for both train and test sets. During training, we ask the model to answer queries about the relation `R` using facts in the database. The loss is the mean squared error between the model's answer and the true answer. The model is trained with weak supervision. Only the query input and answer are used, and intermediate steps do not have supervision. Table 1 lists induced logic rules for some relations. We found that the model can learn not only rules with multiple predicates, but also rules that involve disjunctions. In the future, we plan to apply the model on more complex and challenging datasets, including web-scale knowledge bases.

[1] An argument retrieval query is where a constant c and a predicate P are given, and we want to infer the distribution over $\{x \mid P(c, x) \text{ is True }\}$

References

[1] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[2] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Structure learning via parameter learning. In *CIKM 2014*, 2014.

[3] Stanley Kok and Pedro Domingos. Learning the structure of markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448. ACM, 2005.

[4] William W Cohen. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*, 2016.

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

[6] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[7] 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*, 2016.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Tim Rocktäschel and Sebastian Riedel. Learning knowledge base inference with neural theorem provers. In *NAACL Workshop on Automated Knowledge Base Construction (AKBC)*, 2016.

[10] William Yang Wang and William W Cohen. Learning first-order logic embeddings via matrix factorization. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2015), New York, NY*, 2016.