Differentiable Learning of Logical Rules for Knowledge Base Completion

Fan Yang [1] **Zhilin Yang** [1] **William W. Cohen** [1]

Abstract

Learned models composed of probabilistic logical rules are useful for many tasks, such as knowledge base completion. Unfortunately this learning problem is difficult, since determining the structure of the theory normally requires solving a discrete optimization problem. In this paper, we propose an alternative approach: a completely differentiable model for learning sets of first-order rules. The approach is inspired by a recently-developed differentiable logic, i.e. a subset of first-order logic for which inference tasks can be compiled into sequences of differentiable operations. Here we describe a neural controller system which learns how to sequentially compose the these primitive differentiable operations to solve reasoning tasks, and in particular, to perform knowledge base completion. The long-term goal of this work is to develop integrated, end-to-end systems that can learn to perform high-level logical reasoning as well as lower-level perceptual tasks.

1. Introduction

A large body of work in AI and machine learning has considered the problem of learning models composed of sets of first-order logical rules, such as the rule: for all a,b,c,

$$\text{AthletePlaysForTeam(a, b)} \wedge \text{TeamPlaysInLeague(b, c)} \implies \text{AthletePlaysInLeague(a,c)}$$

This problem is often called *inductive logic programming* (ILP) (Muggleton et al., 1992) or *statistical relational learning* (SRL) (Getoor, 2007) and typically the underlying logic is a probabilistic logic, such as Markov Logic Networks (Richardson & Domingos, 2006) or ProPPR (Wang et al., 2013). Learned models composed of probabilistic logical rules are useful for a number of tasks, including knowledge base completion (Lao et al., 2011; Wang et al., 2015b). Unfortunately the learning problem is quite difficult, since determining the structure of the theory (i.e., the particular sets of rules included) is a *discrete* optimization problem, and one that involves search over a potentially large space of alternatives. Most past learning systems have thus used local-search based optimization methods that interleave moves in a discrete space with moves in parameter space (Kok & Domingos, 2007; Lao & Cohen, 2010; Wang et al., 2014).

[1]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA. Correspondence to: Fan Yang <fanyang1@cs.cmu.edu>, Zhilin Yang <zhiliny@cs.cmu.edu>, William W. Cohen <wcohen@cs.cmu.edu>.

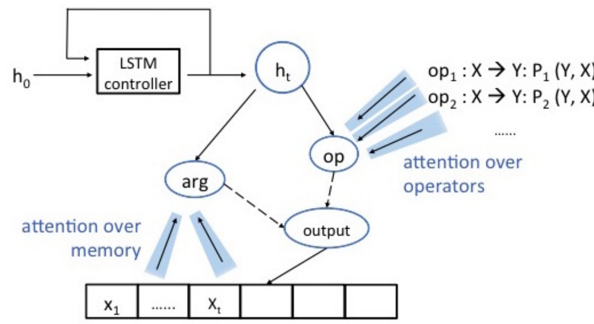


Figure 1. An overview of the model: a memory-augmented recurrent neural network acts as a controller that learns to compose primitive logic operators. At each step, the controller "softly" chooses operators and inputs from memory. The operators correspond to one-hop reasoning in the knowledge base.

In this paper, we propose an alternative approach: a completely differentiable model for learning models defined by sets of first-order rules. The approach is inspired by a differentiable logic called TensorLog (Cohen, 2016), a subset of first-order logic for which inference tasks for can be compiled into sequences of differentiable numerical operations on matrices. TensorLog, however, assumes that a human programmer chooses the set of logical rules, and then uses gradient-based learning methods to optimize parameters (which represent confidences for facts and rules): in other words, it includes no mechanism for structure search.

In this paper, we describe a neural controller system (Neelakantan et al., 2015; Andreas et al., 2016) which *learns* how to sequentially compose the primitive operations used by TensorLog. The space of programs which can be learned by this controller system includes the space of TensorLog programs, but the controller is fully differentiable, and can be learned end-to-end with gradient methods. In outline, our system includes (1) a memory, which holds vectors holding a distribution of potential bindings for logical variables, (2) a neural controller, which is recurrent neural network that takes actions based on the current hidden state, and (3) a set of operations, analogous to the operations used in TensorLog, which perform "primitive logical inference steps". Each operation takes as input one or more previously-bound memory cells, and populates a new memory cell. At each stage of the computation, the controller uses an attention mechanism to pick an operation to perform and inputs to the operation, performs the operation, and finally updates its hidden state.

We conduct experiments on four knowledge base completion tasks of varying scale. Our differentiable rule learning method outperforms a strong iterated structure gradient (Wang et al., 2014) baseline.

The long-term goal of this work is to develop integrated, end-to-end systems that can both perform compositional, logical reasoning, as well as lower-level perception tasks. A differentiable rule learning method is a step toward integration of these two types of learning. While the end result is less interpretable than an explicit set of logical rules, the input and output of the can be interpreted as hidden states or probability distributions for "upstream" or "downstream" modules.

In the remainder of the paper, we introduce the knowledge base completion task in Section 2. In Section 3, we describe in detail the aforementioned TensorLog operators and how to learn rules differentiably. In the Section 4, we compare our method with a strong discrete rule learning baseline on four knowledge bases. Lastly in Section 5, we draw the connections among our methods and other rule learning models.

2. Knowledge Base Completion

2.1. Definitions

Knowledge bases (KB) are sets of facts about real-world entities (such as people, places, and things). The facts in the knowledge base are usually stored in RDF format, i.e. each fact is a triple of the format (head, relation, tail), or equivalently relation(head, tail). For example, a fact Father(Tom, Amy) means that Tom is the father of Amy. Modern knowledge bases such as Freebase (Bollacker et al., 2008) contains millions of entities and billions of facts. Such large amount of structured data has promising applications in many problems, such as information retrieval, natural language understanding, and biological data mining.

Knowledge bases are usually incomplete. There are facts about the entities not included in the database. Hence an important task regarding knowledge base is to automatically discover the missing facts, i.e. knowledge base completion. In some scenarios, the missing facts can be inferred by reasoning from the existing facts. This is based on the assumptions that structure in knowledge bases provides sufficient information. We adopt this assumption and proposes a rule learning method that leverages the structure among relations.

Representation based knowledge base completion methods rely on learning embeddings for entities (Socher et al., 2013). This entity-dependent memorization approach is not as inductively generalizable as our method. Our rule learning method is capable of entity-independent reasoning, a feature that is absent in representation based methods.

2.2. Logic rules for KB completion

We are interested in learning chain-like logic rules of the following form,

$$R_{A_1}(y_1, x) \wedge \cdots \wedge R_{A_n}(z, y_n) \implies R_C(z, x) \tag{1}$$

where $n \leq N$, and N is any fixed integer. $R_{A_1}, \dots, R_{A_n}$, and R_C are relations in the knowledge base, and $y_1, \dots, y_n, x, z$ are entities. These rules can be used for knowledge base completion because if we want to infer the missing fact $R_C(z, x)$, this can be reduced to checking if there exists $y_1, \dots, y_n$ such that $R_{A_1}(y_1, x), \dots, R_{A_n}(z, y_n)$ are all facts in the knowledge base.

Considering the complexity of modern knowledge bases and the fact that they often contain noisy information, it is unlikely to find rules that are consistent with all the data. Thus, we associate each rule with a real number $\alpha \in (0, 1)$ as its *confidence*. The rules now become

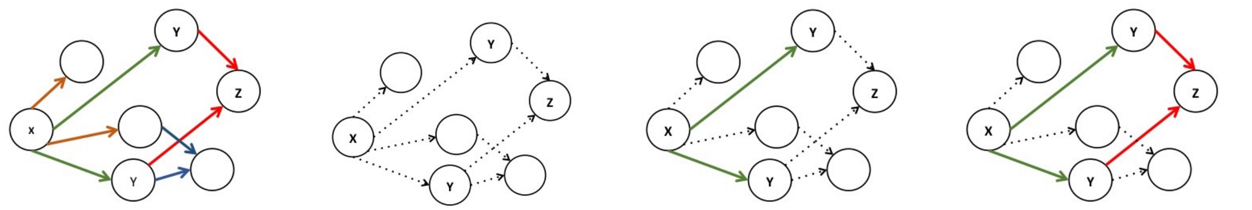
$$R_{A_1}(y_1, x) \wedge \cdots \wedge R_{A_n}(z, y_n) \implies R_C(z, x),\ 0.9 \tag{2}$$

During inference, we will compute the confidence of $R_C(z, x)$ by adding the confidences of all the rules that imply it.

3. Differentiable Rule Learning

3.1. TensorLog representation of KB and logic rule inference

In order to make rule learning differentiable, we first introduce a matrix representation of the knowledge base and logic rule inference, as used in TensorLog (Cohen, 2016). In this representation, entities are numerically encoded. We represent each entity e with a one hot vector $v_e \in \{0, 1\}^N$, where N is the number of entities, and only the e-th component is 1. For each relation C in the KB, we use a matrix



(a) Entities (nodes) and relations (arcs). Different arc colors indicate different relation types.

(b) Before applying any operators. The goal is to find a relation path that connects x and z.

(c) Multiplying with M^A where A is the relation corresponds to the green arcs.

(d) Multiplying with M^B where B is the relation corresponds to the red arcs.

Figure 2. An graph-based view of sequentially applying TensorLog operators. Each TensorLog operator (i.e. matrix multiplication) is equivalent to a one-hop reasoning in the knowledge base.

$M^C \in \{0,1\}^{N\times N}$ to store all the facts about this relation. Then entry $M^C_{i,j}$ is 1 if and only if $R_C(i,j)$ is a fact in the knowledge base.

Using this matrix representation, we can imitate the rule inference $R_A(y,x) \wedge R_B(z,y) \implies R_C(z,x)$ for all y by computing

$$u_{z,B,A,x} \doteq v_z^T M^B M^A v_x. \tag{3}$$

In the context of this paper, the TensorLog *operators* indeed refer to *multiplying with matrix* M^R for some relation R. Figure 2 shows a graph-based equivalent view of applying TensorLog operators. In other words, each TensorLog operators (i.e. multiplications with certain matrices) allow the model to perform one-hop reasoning in the knowledge base.

It is straightforward from the representation definition that

$$|\{y \mid R_A(y,x) \wedge R_B(z,y) \text{ holds }\}| = u_{z,B,A,x} \tag{4}$$

In other words, given z and x, the number of y's that satisfy $R_A(y,x) \wedge R_B(z,y)$ is equal to the matrix product $v_z^T M^B M^A v_x$.

Since $R_A(y,x)$ and $R_B(z,y)$ together implies $R_C(z,x)$, It is natural to assume that the more such y's, the more likely $R_C(z,x)$ holds. Thus, we will use $u \doteq M^B M^A v_x$ to represent the *confidence*, denoted $\mathbb{C}(\cdot)$, over entities $\{z\}$ such that $R_C(z,x)$ holds. Mathematically, this means

$$u_i = |\{y \mid R_A(y,x) \wedge R_B(z_i,y)\}| \propto \mathbb{C}[R_C(z_i,x) \text{ holds}] \tag{5}$$

As we mentioned before, there could be multiple rules that imply R_C, each with different confidences. Let these rules be

$$R_{A_l}(y,x) \wedge R_{B_l}(z,y) \implies R_C(z,x) \tag{6}$$

Let the confidence of these rules be α_l, where $l = 1, \dots, L$. Therefore, the *confidence* of $R_C(z_i,x)$ is

$$\mathbb{C}[R_C(\cdot,x) \text{ holds}] \propto \sum_l \alpha_l M^{B_l} M^{A_l} v_x \tag{7}$$

Though we have described the rules in terms of length two, it is straightforward to generalize to cases where rule lengths varies, resulting the following mathematical formulation,

$$\mathbb{C}[R_C(\cdot,x) \text{ holds}] \propto \sum_l \alpha_l \prod_{j\in\beta_l} M^j v_x \tag{8}$$

where β_l contains the indices of relations in rule l. It is worth pointing out that matrix multiplication is not commutative in this scenario, and we must multiply the relation matrices in the particular order specified by the rules.

3.2. Learning the logic rules

We will now discuss the rule learning process, including learnable parameters and model architecture. As shown in Equation 8, for each query $R_C(\cdot,x)$, we need to learn the set of rules that imply it and the confidences associated with these rules. Since these parameters are dependent on the query relation, we augment the notation with raising indices to make this explicit. The learnable parameters of our model is $\{\alpha_l^C, \beta_l^C\}$, for each relation C.

However, it is difficult to formulate a *differentiable* process to directly learn the parameters $\{\alpha_l^C, \beta_l^C\}$ (Lao & Cohen, 2010). This is because each parameter is associated with a particular rule, and enumerating rules is an inherently discrete task. To overcome this difficulty, we observe that an *almost* equivalent way to write Equation 8 is to interchange the summation and product, resulting the following formula with different parameterization,

$$\prod_t^T \sum_k^K a_t^k M^k v_x \tag{9}$$

where T is the max length of rules and K is the number of relations in the knowledge base. The key parameterization difference between Equation 8 and Equation 9 is that in the latter we associate each relation in the rule with a weight. This combines the rule enumeration and confidence assignment.

However, there is a problem with Equation 9 that it assumes all rules are of the same length. This issue can be solved by using a similar but recurrent formulation

$$u_0 = v_x,\ u_t = \sum_\tau^{t-1} b_\tau^t u_\tau,\ u_{t+1} = \sum_k^K a_t^k M^k u_t \tag{10}$$

Equation 10 can represent rules of different lengths because of the *memory* vectors u_t. The memory vector allows the model to append relations to any partially enumerated rules, hence the various rule lengths.

Given the formulation in Equation 10, the learnable parameters for each relation R_C becomes $\{a_t^k, b_\tau^t\}$ for $t = 1, \ldots, T$ and $k = 1, \ldots K$. We use a memory-augmented Long short-term memory network (Hochreiter & Schmidhuber, 1997; Weston et al., 2014; Graves et al., 2016), denoted *controller*, to model this recurrent process. The number of unroll steps in the *controller* is the maximum length of the logic rules that we want to learn. The *controller* mechanism is the following:

$$h_t, c_t = \text{update}(h_{t-1}, c_{t-1}, \text{input}_t) \tag{11}$$

$$a_t = \text{softmax}(Wh_t + b) \tag{12}$$

$$b^t = \text{softmax}([h_1, \ldots, h_{t-1}]^T h_t) \tag{13}$$

The h_t and c_t are hidden states of the *controller*. The input_t is a continuous representation of the relation R_C. The *controller* predicts relation weights a_t^k based on the relation and previous hidden states. The weights b_τ^t on previous memory are based on comparing current hidden states with all previous hidden states.

The objective function is to minimize the negative cross entropy between normalized u_T, denote $\overline{u_T}$ and v_z for all z that $R_C(z, x)$ holds but are not available in the knowledge base.

$$\text{Loss} = -\left(\overline{u_T} \log v_z + (1 - \overline{u_T} \log(1 - v_z)\right) \tag{14}$$

The loss function in Equation 14 indeed represents our goal of using logic rules for knowledge base completion. It aims to learn logic rules that when executed return the same result as if directly infer the relation R_C.

We can naturally extend the model to include entities information in the knowledge base. The idea is to use the entity to control how much information flow from the hidden state h_t to the relation weights a_t by adding a gate on the hidden states (Dhingra et al., 2016; Yang et al., 2016). Mathematically, this means

$$h_t = h_t \odot \text{sigmoid}(W_1 f(x) + W_2 h_{t-1} + b') \tag{15}$$

where $f(x)$ is a continuous representation of the entity x. This will make the learned logic rules not only relation dependent, but also entity dependent, which can be considered as adding a condition term in the logic rules, such as

$$\text{If entity is } x \text{, then } R_A(y, x) \wedge R_B(z, y) \implies R_C(z, x) \tag{16}$$

4. Experiments

To study the effectiveness of our differentiable rule learning model, we apply it to the knowledge base completion task on four knowledge bases. We compare the experiment results from our model and those from an *iterated structural gradient* (ISG) (Wang et al., 2014) structure learning method based on ProPPR, an efficient and scalable first-order probabilistic logic program (Wang et al., 2013). The ProPPR based ISG method has been compared with other popular ILP methods such as FOIL (Quinlan, 1990), or pseudo-likelihood based structure learning methods for MLNs (Richardson & Domingos, 2006).

4.1. Data sets

We use the following four knowledge bases in our experiment: Unified Medical Language System (UMLS) (Kok & Domingos, 2007), Alyawarra kinship (Kok & Domingos, 2007), Wordnet (Miller, 1995), and a subset of Freebase (Bordes et al., 2013). The statistics of the databases are shown in Table 1.

The UMLS knowledge base contains facts about medical-related objects and symptoms, such as *(bacterium, affects, anatomical abnormality)*. The Alyawarra kinship knowledge base contains facts about demographical information of an aboriginal tribe in central Australia. The Wordnet knowledge base contains lexical information about English words. The Freebase15k knowledge base is a subset of Freebase, it contains general facts about movies, sports, etc.

4.2. Experiment setup

We now describe the experiment setup of rule learning for knowledge base completion. The completion task involves three type of data during training and testing. Firstly, we need a database $\mathcal{DB}$ that store the facts we already know. Secondly, we need training samples and labels of the form

Table 1. Database statistics

Database	# Relation	# Entity	# Fact
UMLS	46	135	6,529
Alyawarra kinship	26	104	10,686
Wordnet	18	40,943	151,442
Freebase15k	1,345	14,951	592,213

(query, answer) for the *completion* task. Each (query, answer) example can be derived from a fact (z, R_C, x). The query is $R_C(?, x)$ and answer is z. The model will use the database $\mathcal{DB}$ to find *answer* for the *query*.

There are multiple ways to divide the knowledge base facts into database, training examples, and test examples. We describe the data splitting and training/testing protocols used in our experiments in the next Section 4.2.1.

4.2.1. Data splitting and training/testing protocols

The first protocol divides the facts in knowledge base into four disjoint sets: train database $\mathcal{DB}_{\text{train}}$, train examples $(q_{\text{train}}, a_{\text{train}})$, test database $\mathcal{DB}_{\text{test}}$, and test examples $(q_{\text{test}}, a_{\text{test}})$. During training (testing), the model will use $\mathcal{DB}_{\text{train}}$ ($\mathcal{DB}_{\text{test}}$) to answer queries q_{train} (q_{test}).

The second protocol first divides the knowledge base into two disjoint sets: $\mathcal{DB}_{\text{all}}$, and test examples $(q_{\text{test}}, a_{\text{test}})$. During each iteration in training, a mini-batch of facts f_{train} is removed from the database $\mathcal{DB}_{\text{all}}$ to derive $(q_{\text{train}}, a_{\text{train}})$. And the database during this training iteration is $\mathcal{DB}_{\text{all}} \setminus f_{\text{train}}$. During testing, the database $\mathcal{DB}_{\text{all}}$ is used by the model to find answers a_{test} for q_{test}. This protocol allows more training examples then the first one.

4.3. Result and analysis

4.3.1. Baseline

We consider a structure learning strong baseline which is an iterated structural gradient structure learning method. Details about this method can be found in Section 5.1. The data splitting and training/testing setup used in this method is the first protocol described in Section 4.2.1. As mentioned in Section 2.2, since our method aims to learn logic rule, we only compare with logic-based inference method.

4.3.2. Implementation

In our differentiable rule learning model, we can use either setup described in Section 4.2.1 and conduct experiments for both protocols. For the optimization part of the model, we use a stochastic gradient descent algorithm ADAM (Kingma & Ba, 2014) with learning rate 0.001. We set the maximum rule length to be two. The input to the neural controller at each step is a continuous representation of the query relation. This is done by looking up the relation embedding matrix. This embedding matrix is randomly initialized.

Table 2. Experiment results on UMLS and Alyawarra kinship

Model	UMLS	Kinship
(Protocol I)		
ISG	0.213	0.075
Diff Rule	**0.229**	**0.156**
(Protocol II)		
Diff Rule	0.472	0.483
Diff Rule Embed	**0.485**	**0.554**

The hidden states size and query continuous representation size are both 128. The mini-batch size is 128 or 64 depending on whether the mini-batch can fit in GPU memory. The model is implemented in TensorFlow (Abadi et al., 2016). We store the matrices M's in Equation 10 as sparse matrices. The model is run on both CPU and GPU, since sparse matrix related operators are only supported on CPU in TensorFlow.

4.3.3. Experiment results

In the context of knowledge base completion, the answer to a query is usually a ranked list of entities. Therefore we use Hits@10 (Bordes et al., 2013) as the evaluation metric. This metric checks if the target entity is ranked in the top 10.

We show the evaluation results on the four data sets in Table 2 and Table 3. The upper half of the table are experiments that use the first data splitting protocol. The lower half of the table are experiments that use the second data splitting protocol. The baseline iterated structure gradient (*ISG*) method is only implemented using the first protocol. *Diff Rule* refers to our differentiable rule learning method. *Diff Rule Embed* refers to where we also use entity embeddings to gate the neural controller hidden states, as described in Equation 15.

Using both data splitting and training/testing protocols, our differentiable rule learning method performs better then iterated structural gradient. This is because our method learns rules for all query relations jointly, so it could leverage the similarity among relations and learn rules more ef-

Table 3. Experiment results on Wordnet and Freebase15k

Model	Wordnet	Freebase15k
(Protocol I)		
ISG	0.349	0.182
Diff Rule	**0.821**	**0.699**
(Protocol II)		
Diff Rule	**0.942**	0.645
Diff Rule Embed	0.928	**0.660**

Table 4. Learned rules and confidences

instance hyponym (z, x) $\Longrightarrow$ similiar to (z, x), 0.5
derivationally related form (z, x) $\Longrightarrow$ similar to (z, x), 0.3
process of (z, y) $\wedge$ affects(y, z) $\Longrightarrow$ affects(z, x), 0.4
result of (z, y) $\wedge$ affects (y, x) $\Longrightarrow$ affects (z, x), 0.3
result of (z, y) $\wedge$ causes (y, x) $\Longrightarrow$ complicates (z, x), 0.45
causes (z, y) $\wedge$ assesses effect of (y, x) $\Longrightarrow$ diagnoses (z, x), 0.43

fectively.

In Table 4, we display some learned logic rules with relative high confidences from the knowledge bases. The logic rules and their confidences are derived based on Equation 8 and Equation 10. Intuitively speaking, we compute the rule confidences α_l by multiplying together the attention a_t^k of each relations that compose the rule.

5. Related Work

5.1. Inductive logic programming

A number of approaches have been explored to perform a data-driven search for for logical theories. Except in trivial cases, this problem is intractable (Cohen & Page, 1995), so heuristics are usually used to search this discrete space. For example, the FOIL system (Quinlan, 1990; Quinlan & Cameron-Jones, 1993) uses decision-tree like information gain measures to guide a greedy search, in which rules are constructed by incrementally adding conditions to an empty rule body, and theories are constructed by incrementally adding rules to an empty theory. Similar heuristics have been applied to search for zeroth-order (propositional rule sets) (e.g., (Cohen, 1995)). Other discrete search methods used to search for logical theories include searches that exploit special operators, such as finding the least general generalization of two candidate rules (Muggleton et al., 1990) or inverse entailment (Muggleton, 1995). While these techniques are often successful they do not exploit the power of modern gradient-based optimization methods, and are not well-suited to use in integrated, end-to-end systems that can combine logical reasoning with neural approaches for lower-level tasks.

In order to model statistically complex data, more recent approaches to learning logical theories adopt probabilistic logics, rather than standard logics. For instance, Kok and Domingos (Kok & Domingos, 2005) describe a system which interleaves parameter optimization of weights for Markov Logic Network (MLN) rules and beam search to find the structure of a MLN (Richardson & Domingos, 2006). These approaches are robust but computationally expensive in practice, especially since for many probabilistic logics, parameter optimization (and often inference) is expensive.

Alternatively, search can be made more efficient by restricting the space of logical rules. For instance, the Path Ranking Algorithm (PRA) (Lao et al., 2011) first enumerates a subset of all possible rules, consisting of chain rules of limited length. Using these rules, the algorithm assigns features to each pair of entities, based on a confidence score derived from a random-walk semantics for the rule. Finally, it trains a classifier using these features that classifies relations between entities, using $L1$-regularization to eliminate unnecessary rules. Our method is broadly similar to PRA: in particular, we also consider chain rules (in these experiments) and associate weights with rules. However, our method jointly enumerates rules and assigns weights in an end-to-end *differentiable* manner.

PRA has since been extended in several ways, for instance, by adopting "softer paths" (Gardner et al., 2014) based on vector space similarity. One extension is the logic ProPPR (Wang et al., 2015a), which generalizes the paths of PRA to a full logic, which extends stochastic logic programs (Muggleton et al., 1996). ProPPR has also been combined with Iterative Structural Gradient (ISG), a variant of a structure learning method based on use of meta-interpreters (Muggleton & Lin, 2013). Briefly, a ProPPR meta-interpreter is constructed which behaves like a theory which contains a large number of rules—in particular, every rule which can be generated by a set of rule templates—such that the parameters of this "template theory" correspond to specific instantiations of a rule. By computing a gradient of the cost function with respect to the parameters, it is possible to identify rule instantiations which would reduce loss. These are added to the meta-interpreter as special cases, and the process is repeated. After several iterations of rule introduction based on "structural gradient", weights are learned for the learned rules using ordinary parameter learning (by gradient descent.) Though ISG is a fairly scalable and general method, this approach still requires discrete moves through structure space, and hence it is not clear how to integrate with other perception tasks in an end-to-end differentiable fashion. ISG is used in the experiments of Section 4 as our strong baseline, and the experiments suggest a substantial improvement in performance on most datasets.

5.2. Neural programmer and Differentiable Neural Computer

Our learning approach is based on past work in learning neural controllers which learn to perform sequences of (usually differentiable) operations on memory. For instance, the Neural Programmer (Neelakantan et al., 2015; 2016) is a similar neural controller model that uses an attention mechanism to "softly" choose which actions to take at each step. Our method differs from Neural Programmer in several details, but notably in the choice of operators. The goal of Neural Programmer is to parse natural language utterances into sequences of actions, and hence each of its action is an independent task, such as table lookup or summation. In contrast, we select operations (such as sparse matrix multiplication) and a database encoding such that operations, properly sequenced, can be used to implement logical reasoning.

The operators we use are based on operations used internally in TensorLog (Cohen, 2016), a recently developed deductive database, for which the reasoning process is differentiable. While we adopt the same data representation and operators of TensorLog, one key difference between our method and TensorLog that TensorLog assumes given the logic programs, it then performs inference using the logic programs. In contrast, in our method, we aim to learn the structure of the logic program.

The Differentiable Neural Computer (DNC) (Graves et al., 2016) is a more general model and is capable of modeling more complex programs than our method. However, during training, DNC requires full supervision, such as step by step program executions, while our method only requires weak supervision: the only "reward signal" in our method is whether the composed logic rules return the desired entity.

5.3. Representation learning

Knowledge base completion, the task considered in the experimental part of this paper, can also be addressed with representation learning methods (Bordes et al., 2013; Socher et al., 2013; Lin et al., 2015; Shen et al., 2016), in which knowledge bases entities and relations are represented as low-dimensional continuous vectors, so that similarity in this space can be used to make certain inferences. A common approach in knowledge-base embedding models is to learn representations of relations (denoted W_R) and entities (denoted v_e) such that for some measurement function f, the value $f(v_{e_1}, W_R, v_{e_2})$ is maximized for all $R(e_1, e_2)$ facts. To infer whether some fact $R'(e_1, e_2)$ holds, we can compute and if necessary threshold the value $f(v_{e_1}, W_{R'}, v_{e_2})$. Unfortunately, embedding methods will always fail to infer $R'(\hat{e}, e_2)$ if $\hat{e}$ is a entity not seen in training, because the representation of the new entity $v_{\hat{e}}$ is unavailable to the model.

Rule learning approaches model the structure of inferences, rather than the structure of entities, so they do not suffer from the same problem. Hence in many ways these approaches are complementary to knowledge-base embedding approaches, and the relative performance of the methods on a task probably depends on the sparsity of relations associated with entities (with more relations per entity favoring embedding models) versus the regularity of the inference process (with logically simpler inferences favoring rule learning approaches). We note that on some standard knowledge-base embedding tasks, previous comparisons have shown that ISG is quantitatively competitive with knowledge-base embedding approaches (Wang & Cohen, 2016).

6. Conclusions and Future Work

We present a *differentiable* rule learning method for the knowledge base completion task. Our method builds upon a recently developed differentiable deductive database TensorLog. The input and output of our rule learning method are real-valued vectors that can be interpreted as hidden states and probability distributions, which are objects that can be conveniently connected with "upstream" and "downstream" learning tasks. We hope this differentiable rule learning method can enable various integrations of reasoning and pattern recognition tasks.

In the future, we plan to work on more problem settings where composition of logic operators is essential and complementary to pattern recognition, such as data integration, reading comprehension, etc.

References

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

Andreas, Jacob, Rohrbach, Marcus, Darrell, Trevor, and Klein, Dan. Learning to compose neural networks for question answering. In *Proceedings of NAACL-HLT*, pp. 1545–1554, 2016.

Bollacker, Kurt, Evans, Colin, Paritosh, Praveen, Sturge, Tim, and Taylor, Jamie. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250. ACM, 2008.

Bordes, Antoine, Usunier, Nicolas, Garcia-Duran, Alberto, Weston, Jason, and Yakhnenko, Oksana. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pp. 2787–2795, 2013.

Cohen, William W. Fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning*, pp. 115–123, 1995.

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

Cohen, William W and Page, C David. Polynomial learnability and inductive logic programming: Methods and results. *New Generation Computing*, 13(3-4):369–409, 1995.

Dhingra, Bhuwan, Liu, Hanxiao, Cohen, William W, and Salakhutdinov, Ruslan. Gated-attention readers for text comprehension. *arXiv preprint arXiv:1606.01549*, 2016.

Gardner, Matt, Talukdar, Partha, Krishnamurthy, Jayant, and Mitchell, Tom. Incorporating vector space similarity in random walk inference over knowledge bases. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar., October 2014. Association for Computational Linguistics.

Getoor, Lise. *Introduction to statistical relational learning*. MIT press, 2007.

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

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Kok, Stanley and Domingos, Pedro. Statistical predicate invention. In *Proceedings of the 24th international conference on Machine learning*, pp. 433–440. ACM, 2007.

Lao, Ni and Cohen, William W. Relational retrieval using a combination of path-constrained random walks. *Machine learning*, 81(1):53–67, 2010.

Lao, Ni, Mitchell, Tom, and Cohen, William W. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 529–539. Association for Computational Linguistics, 2011.

Lin, Yankai, Liu, Zhiyuan, Luan, Huanbo, Sun, Maosong, Rao, Siwei, and Liu, Song. Modeling relation paths for representation learning of knowledge bases. *arXiv preprint arXiv:1506.00379*, 2015.

Miller, George A. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

Muggleton, Stephen. Inverse entailment and progol. *New generation computing*, 13(3-4):245–286, 1995.

Muggleton, Stephen, Feng, Cao, et al. *Efficient induction of logic programs*. Citeseer, 1990.

Muggleton, Stephen, Otero, Ramon, and Tamaddoni-Nezhad, Alireza. *Inductive logic programming*, volume 38. Springer, 1992.

Muggleton, Stephen et al. Stochastic logic programs. *Advances in inductive logic programming*, 32:254–264, 1996.

Muggleton, Stephen H and Lin, Dianhuan. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

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

Neelakantan, Arvind, Le, Quoc V, Abadi, Martin, McCallum, Andrew, and Amodei, Dario. Learning a natural language interface with neural programmer. *arXiv preprint arXiv:1611.08945*, 2016.

Quinlan, J. Ross. Learning logical definitions from relations. *Machine learning*, 5(3):239–266, 1990.

Quinlan, J Ross and Cameron-Jones, R Mike. Foil: A midterm report. In *European conference on machine learning*, pp. 1–20. Springer, 1993.

Richardson, Matthew and Domingos, Pedro. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

Shen, Yelong, Huang, Po-Sen, Chang, Ming-Wei, and Gao, Jianfeng. Implicit reasonet: Modeling large-scale structured relationships with shared memory. *arXiv preprint arXiv:1611.04642*, 2016.

Socher, Richard, Chen, Danqi, Manning, Christopher D, and Ng, Andrew. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pp. 926–934, 2013.

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

Wang, William Yang, Mazaitis, Kathryn, and Cohen, William W. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 2129–2138. ACM, 2013.

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

Wang, William Yang, Mazaitis, Kathryn, Lao, Ni, and Cohen, William W. Efficient inference and learning in a large knowledge base. *Machine Learning*, 100 (1):101–126, 2015a. ISSN 1573-0565. doi: 10.1007/s10994-015-5488-x. URL http://dx.doi.org/10.1007/s10994-015-5488-x.

Wang, William Yang, Mazaitis, Kathryn, Lao, Ni, and Cohen, William W. Efficient inference and learning in a large knowledge base. *Machine Learning*, pp. 1–26, 2015b.

Weston, Jason, Chopra, Sumit, and Bordes, Antoine. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Yang, Zhilin, Dhingra, Bhuwan, Yuan, Ye, Hu, Junjie, Cohen, William W, and Salakhutdinov, Ruslan. Words or characters? fine-grained gating for reading comprehension. *arXiv preprint arXiv:1611.01724*, 2016.