Logical Rule Induction and Theory Learning Using Neural Theorem Proving

Paper by

A. Campero, A. Pareja, T. Klinger, J. Tenenbaum & S. Riedel, 2019

Presented by Kaifu Wang

March 23, 2020

- **Rule Induction**: Given a knowledge base of person relations, can we build a learning algorithm to learn a target rule such as "if X is the father of Y and Y is a parent of Z, then X is the grandfather of Z"?
- **Theory Learning**: Given a knowledge base of animals, how to automatically develop an animal taxonomy, so that it can use (minimum) logical rules to explain the facts in the KB?

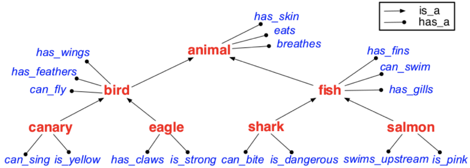


Figure 1: Animal Taxonomy. Constants are in red and blue, relations are indicated with lines and arrows.

- How to design differentiable representation for predicates and rules?
- How to generate candidate set of logic rules and evaluate them?
- How to supervise the learning without direct annotation of the rules?

Background: Terminology

- **Atom:** A predicate applied to a list of terms (variables or constants), e.g.,

  `fatherOf(X,Y)`

- **Rule:** In this paper, we only consider logic rules of form $h \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_k$ where h and b_i's are head and body atoms, e.g.,

  $$\texttt{grandfatherOf(X,Z)} \leftarrow \texttt{fatherOf(X,Y)} \wedge \texttt{parentOf(Y,Z)}$$

- **Fact:** A given atom whose terms are all constants, e.g.,

  `brotherOf(Mario, Luigi)`

- **Forward Chaining:** Given background facts, match them with the body of a rule to derive new facts.
- **Backward Chaining:** Given goal atom (to be proved), find the rule that can conclude it and recursively try to prove the body atoms of the rule.

- **Symbolic**
  - Inductive Logic Programming: learn interpretable rules from data and exploit them for reasoning.
  - (Kakas, Kowalski, and Toni 1992) Abductive Logic Programming: learn consistent explanatory facts as well as rules.
  - Learning hard logic rules, not robust to noisy input.
- **Neuro-Symbolic**
  - (Rockaschel and Riedel, 2017) A differentiable prover using backward chaining. Learning the representation of the true facts.
  - (Evans and Grefenstette, 2018) ∂_{ILP}: Rule induction using forward chaining. Generate candidate rules using templates. Learning the weights (correctness) of candidate rules.

This paper: neuro-symbolic, forward-chaining, learning the embeddings.

We first introduce the model for rule induction. In this case, the learner's input includes a set of background facts and a set of labeled target facts. For example, in the task of learning the predicate $\texttt{even}(X)$ for integer X using the successive relation of integers, we have

$$\begin{aligned} \text{Background} &= \{\texttt{zero}(0), \texttt{succ}(0,1), \texttt{succ}(1,2), \ldots, \texttt{succ}(9,10)\} \\ \text{Target Positive} &= \{\texttt{target}(0), \texttt{target}(2), \ldots, \texttt{target}(10)\} \\ \text{Target Negative} &= \{\texttt{target}(1), \texttt{target}(9)\} \end{aligned}$$

Proposed Method:

1. Initialize the representation of predicates.
2. Generate candidate rules (proto-rules) using manually-designed task-specific templates.
3. For each candidate rule and each pair of facts, perform K times forward chaining (K is a hyperparameter).
4. Compare the inferred facts with the labeled target facts, and backpropagate the loss.

Model: Representation

- **Constants** are represented as integers.
- **Atom** $a = (\theta, s, o)$ where θ is the embedding of the predicate (to be learnt), and s, o are subject and object of the atom respectively.
- **Rule** $r = (a_h, a_{b_1}, a_{b_2})$ where a_h is the head atom and a_{b_i} are the body atoms (In this paper, rules are restricted to have at most two body atoms).
- **Fact** $f = (\theta, s, o, v)$ where $v \in [0, 1]$ represents the belief that the atom (θ, s, o) is true.

Model: Candidate Rule Generation

For example, in the task of learning $\texttt{even}(X)$, the following template is used:

$$P_1(X) \leftarrow P_2(X)$$
$$P_1(X) \leftarrow P_2(Z) \land P_3(Z, X)$$

where $P_i \in \{\texttt{even}, \texttt{zero}, \texttt{succ}\}$.
(This template is probably designed by an analogy of the structure of the true logic rule, which is known to the human. Is this candidate set of rules too small?)

Model: Forward Chaining

Given a pair of facts $f_i = (\theta_{f_i}, s_{f_i}, o_{f_i}, v_{f_i})$ and rule $r = (a_h, a_{b_1}, a_{b_2})$:

- Constant Matching: check if the terms of f_1, f_2 can be assigned to the rule (do not check predicates). For example, given rule

  `grandfatherOf(X,Z)` $\leftarrow$ `fatherOf(X,Y)` $\wedge$ `parentOf(Y,Z)`

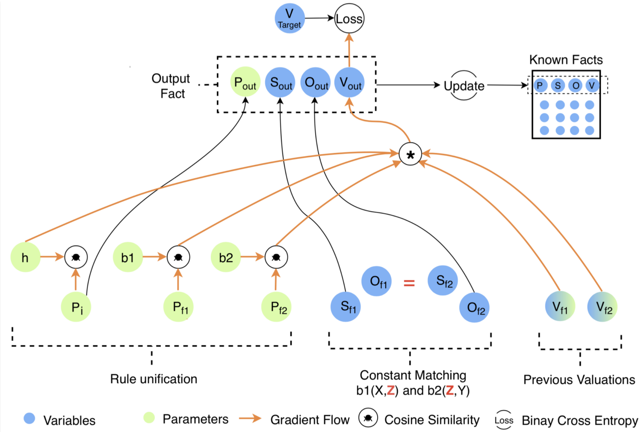
  Then fact pair `fatherOf(Alice, Bob)`, `fatherOf(Bob, Cook)` is matched, but pair `fatherOf(Alice, Bob)`, `fatherOf(Lee, Cook)` is not.
- If matched (denote the matched subject and object for the rule as $s_{\text{out}}, o_{\text{out}}$), for each predicate p, we generate a candidate output fact $f = (\theta_p, s_{\text{out}}, o_{\text{out}})$.
- Compute the score of f using a soft form of conjunction by:

$$v_{\text{out}} = \cos(\theta_h, \theta_p) \cdot \cos(\theta_{b_1}, \theta_{f_1}) \cdot \cos(\theta_{b_2}, \theta_{f_2}) \cdot v_{f_1} \cdot v_{f_2}$$

  So now we have an inferred fact $(\theta_p, s_{\text{out}}, o_{\text{out}}, v_{\text{out}})$.

Model: Backpropagation

For each candidate rule, we match it with constants of all pairs of given facts. If matched, perform K step forward chaining. If the predicate and arguments of the inferred fact matches one of the target labeled fact, loss is computed and backpropagated.



Model: Theory Learning

The goal is given a set of facts, we wish to learn some logical rules and core facts so that the observations can be recovered by the rules and core facts.

Proposed Method:

- Fix a set of core facts, we initialize the scores of all the other facts as 0.5, i.e., we forget the truth value of all the other facts.
- Add a regularization term to reduce the size of core facts. Overall, the loss becomes

$$\sum_{i \in I, f \in F, i \sim f} \text{Cross-Entropy}(v(f), v(i)) + \lambda \sum_{i \in I} v(i)$$

  where I is the set of inferred facts, F is the set of all observed facts and $\sim$ indicates if the predicates and arguments of two facts match.
- Train the model so that it can best recover the other observations.

Experiment: Rule Induction

Table 1: ILP percentage of successful runs. $|I|$ is the number of intentional predicates.

Task	$\|I\|$	Recursive	∂ILP	Ours
Predecessor	1	No	100	100
Even-Odd	2	Yes	100	100
Even-succ2	2	Yes	48.5	100
Less than	1	Yes	100	100
Fizz	3	Yes	10	10
Buzz	2	Yes	35	70
Member	1	Yes	100	100
Length	2	Yes	92.5	100
Son	2	No	100	100
Grandparent	2	No	96.5	100
Relatedeness	1	No	100	100
Father	1	No	100	100
Undirected Edge	1	No	100	100
Adjacent to Red	2	No	50.5	100
Two Children	2	No	95	0
Graph Colouring	2	Yes	94.5	0
Connectedness	1	Yes	100	100
Cyclic	2	Yes	100	100

- Perform better than ∂_{ILP} in most of the tasks.
- The Fizz and Buzz tasks basically aim to find if an integer can be divided by 3 and 5 using successive relations between integers. Neither of the two methods perform perfectly.
- Fail in the tasks Two Children and Graph Colouring. The author claims that this is because there is a powerful local minima that attracts most of the points in the space.

Experiment: Theory Learning

Table 3: Theory Learning Results. Succ is the percentage of successful initializations; Acc stands for the accuracy of the recovered facts; Const is the number of constants.

	Taxonomy			Family		
	# Preds	# Const	# Facts	# Preds	# Const	# Facts
Observed Data	4	36	145	6	10	30
Target Theory	4	36	40	4	10	28
	% Succ	% Acc	# Induced Facts	% Succ	% Acc	# Induced Facts
Algorithm	70	99	69	100	96	30.8

- Two tasks: Animal Taxonomy and Kinship Theory.
- For animal taxonomy: successfully recover the theory in 70% times. In average, use 69 core facts. The optimal size of core facts is 40.
- For kinship theory: no compression but pollute the known facts. This is because the learnt rule deduces incorrect core facts.

Conclusions

- Contributions
  - Differentiable rule induction using predicate embedding and forward-chaining.
  - Indirect supervision for learning logic rules.
- Limitations
  - Need manually-designed task-specific templates to generate rules.
  - Types of rules are restricted (at most two body atoms).
  - Need to consider all possible fact-rule pairs, not scalable.
- Questions
  - Is it a good idea to encode logical rules only using predicate embeddings?
  - What are the conditions for labeled facts to make sure that we can learn a correct logic rule?
  - For more complex problems, it is necessary to removing some restrictions of the rules, hwo to ensure scalability?

Thank you!

Questions?