

End-to-End Differentiable Proving

Reading Notes

This paper introduce a prover using vector representation.

What can this NTP do?

Use knowledge from KB like a pro-log prover

Even if a KB with corrupted ground atom

e.g.,[grandpaof] is a corrupted version of [grandfatherof]

According to authors words, the 4 aims of this NTP is :

1. Neural network for proving queries to a knowledge base
2. Proof success differentiable w.r.t. vector representations of symbols
3. Learn vector representations of symbols end-to-end from proof success
4. Make use of provided rules in soft proofs
5. Induce interpretable rules end-to-end from proof success

In one line of his own word : **Let's neuralize Prolog's Backward Chaining using a Radial Basis Function kernel for unifying vector representations of symbols!**

What is the structure of the model?

Three kinds of modules:

Unification module:perform matching score calculation for terms in vector space.

OR module & AND module : perform backward chaining

Hard concepts which I have come across when reading

What is the relation between modules and net?

Answer, modules are the nodes of the net. The net is construct with recursion.

What is the meaning of the net which create by each module?/ What does the network learn?

Answer, since the module is a part of the net.The word "create" means it adds itself to the net. From the Training Objective it should have some learnable parameters but I don't see any in the codes. It is still confusing to me right now.

So what does the net mean in prover perspective?

Answer, the net is the proof.

where is the gradient descent?

Answer, I don't know where it is. From the first sentence in 3.Differentiable Prover, they calculate the gradient of proof successes derivative w.r.t vector representations of symbols so if it use BP, it should apply gradient to vector representations.

Besides, there are several implementation problems need to be clarified. I try answering it after reading the incomplete code from Github:

1. How to get the vector representation of a term? Is it learned from training(by autoencoder)? No, just use trained word2vec.
2. What is the measure of similarity between vectors? Sigmoid(L1 (A,B))
3. What is the net actually in states? As it is said the the current net of prover as part of the prover state.
4. Where will the gradients be applied? No variable.

The most valuable idea to me

The use of vector representation for reasoning rules is interesting. **They explore this ideas further in later work (Language Models as Knowledge Bases?)**

However the efficiency of OR and AND modules are questionable. Struggles are mentioned in 4.3 Computational Optimizations **They improve it in later works(Learning Reasoning Strategies in End-to-End Differentiable Proving).**

unsolved problems after reading codes

I can't find the trainable parameters to in their code. Theoretically, they should train the network which create the vector representation .

Draft Notes which is helpful for me at reading time. they starts below:

Abstract

Object:

Knowledge Base

Methods:

dense vector representation of symbols.

replace symbolic unification with a differentiable computation on vector representations of symbols.

Results:

A network can infer facts from in incomplete knowledge base.

e.g.,

(i) place representations of similar symbols in close proximity in a vector space, (ii) make use of such

similarities to prove queries, (iii) induce logical rules, and (iv) use provided and induced logical rules for multi-hop reasoning.

Background

Introduction of the KB which used in this paper.

They only consider function-free first-order logic rules.

Use rule and AND OR operation to do backward chaining.

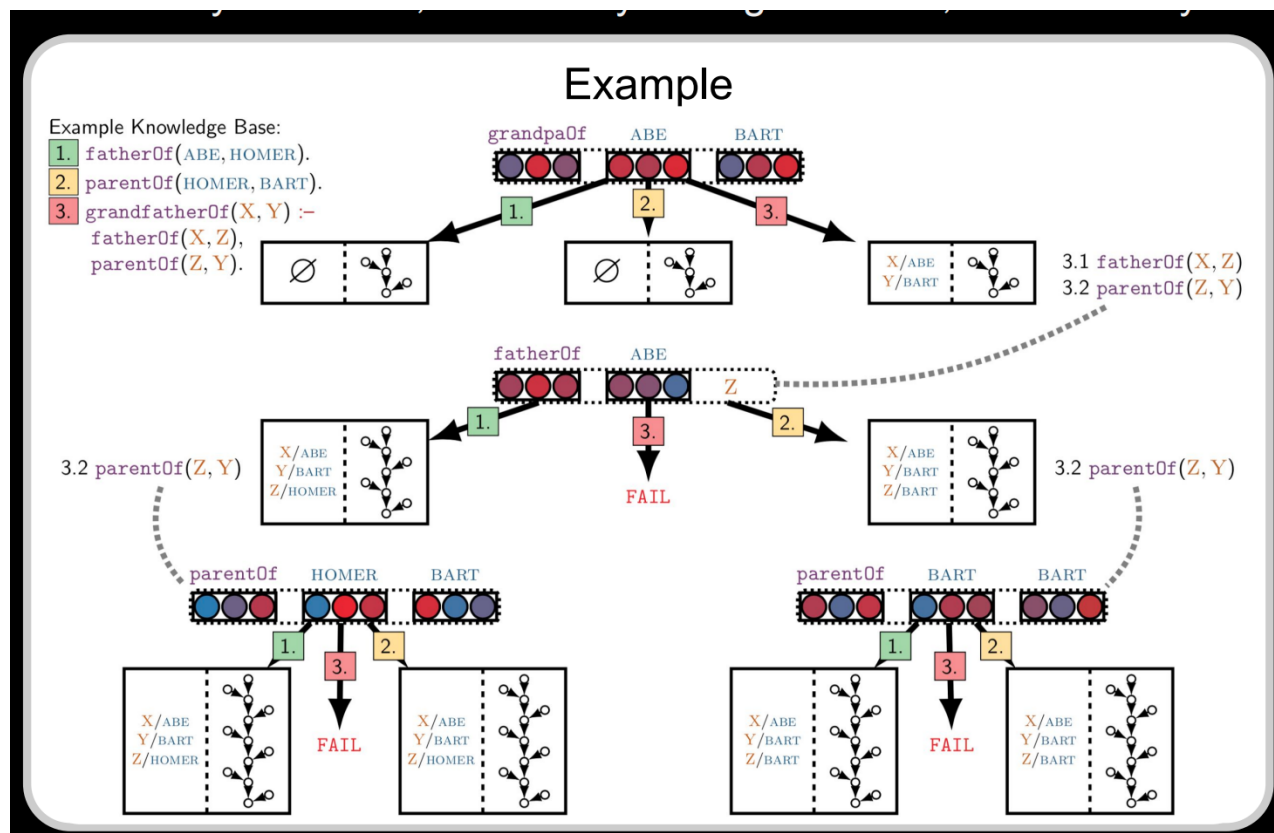
Differentiable prover(construction of the net)

End to End network

Each module takes as inputs discrete objects (atoms and rules) and a proof state, and returns a list of new proof states

A proof state $S = (\psi, \rho)$ is a tuple consisting of the substitution set ψ constructed in the proof so far and a neural network ρ that outputs a real-valued success score of a (partial) proof.

The substitution set is kinds of alias set which give terms their aliases in the proof. e.g., In the the application of module 3, X has substitution ABE



From my perspective, the net can output partial proof success score for all input. e.g.:

step 0:

1,2,3 are three rules in Knowledge Base

Use unification module to process the input so that it can be a unified vector

Use OR module to create query into database.

1 and 2 output $S = (\psi, \rho)$, $\psi = []$, $\rho = \text{net}$, the input of net is the query

3 output $S = (\psi, \rho)$, $\psi = [X/ABE, Y/BART]$, $\rho = \text{net}$, the input of net is the query. This will create two new query.

step 1:

deal with new query with module

step n: no new query generate.

This paper neuralize the backward chaining by replace module with neural net. (I'll check this in code)

All symbol are in vector representation, so that it may be aware of similar symbol.

Unification Module

if unify(A,B) not equal FAIL, proving process will continue.

This module works as a equal notation in theorem prover. It can check if the right hand side and the goal are equal. :

order matters, *i.e.*, if arguments match a line, subsequent lines are not evaluated.

1. $\text{unify}_{\theta}([], [], S) = S$
2. $\text{unify}_{\theta}([], _, _) = \text{FAIL}$
3. $\text{unify}_{\theta}(_, [], _) = \text{FAIL}$
4. $\text{unify}_{\theta}(h : H, g : G, S) = \text{unify}_{\theta}(H, G, S') = (S'_{\psi}, S'_{\rho})$ where

$$S'_{\psi} = \left\{ \begin{array}{ll} S_{\psi} \cup \{h/g\} & \text{if } h \in \mathcal{V} \\ S_{\psi} \cup \{g/h\} & \text{if } g \in \mathcal{V}, h \notin \mathcal{V} \\ S_{\psi} & \text{otherwise} \end{array} \right\}, \quad S'_{\rho} = \min \left(S_{\rho}, \left\{ \begin{array}{ll} \exp \left(\frac{-\|\theta_{h_1} - \theta_{g_1}\|_2}{2\mu^2} \right) & \text{if } h, g \notin \mathcal{V} \\ 1 & \text{otherwise} \end{array} \right\} \right)$$

OR Module

search for rule in KB, create multiple AND Module:

1. $\text{or}_{\theta}^{\mathcal{R}}(G, d, S) = [S' \mid S' \in \text{and}_{\theta}^{\mathcal{R}}(\mathbb{B}, d, \text{unify}_{\theta}(H, G, S)) \text{ for } H :- \mathbb{B} \in \mathcal{R}]$

AND Module

First apply substitution:

tions to variables in an atom if possible. This is realized via

1. $\text{substitute}([], _) = []$
2. $\text{substitute}(g : G, \psi) = \left\{ \begin{array}{ll} x & \text{if } g/x \in \psi \\ g & \text{otherwise} \end{array} \right\} : \text{substitute}(G, \psi)$

For example, $\text{substitute}([\text{fatherOf}, X, Z], \{X/Q, Y/i\})$ results in $[\text{fatherOf}, Q, Z]$.

Then apply AND:

this module is implemented as

1. $\text{and}_{\theta}^{\mathfrak{K}}(_, _, \text{FAIL}) = \text{FAIL}$
2. $\text{and}_{\theta}^{\mathfrak{K}}(_, 0, _) = \text{FAIL}$
3. $\text{and}_{\theta}^{\mathfrak{K}}([], _, S) = S$
4. $\text{and}_{\theta}^{\mathfrak{K}}(G : G, d, S) = [S'' \mid S'' \in \text{and}_{\theta}^{\mathfrak{K}}(G, d, S') \text{ for } S' \in \text{or}_{\theta}^{\mathfrak{K}}(\text{substitute}(G, S_{\psi}), d - 1, S)]$

where the first two lines define the failure of a proof either because of an unstream unification

Success score(aggregation)

The parameterized rules are used as given rules. The vector representation of it optimized in training time too.

iii success score of proving a goal G using a

$$\text{ntp}_{\theta}^{\mathfrak{K}}(G, d) = \arg \max_{\substack{S \in \text{or}_{\theta}^{\mathfrak{K}}(G, d, (\emptyset, 1)) \\ S \neq \text{FAIL}}} S_{\rho}$$

where d is the depth of the proof and the initial state is

Optimization(How to pass the gradient)

There two kinds of loss, basic loss and auxiliary loss.

The corrupted atoms are generated by resampling at training time.

Objective-Basic loss

A typical log likelihood loss:

$$\mathcal{L}_{\text{ntp}_{\theta}^{\mathfrak{K}}} = \sum_{([s, i, j], y) \in \mathcal{T}} -y \log(\text{ntp}_{\theta}^{\mathfrak{K}}([s, i, j], d)_{\rho}) - (1 - y) \log(1 - \text{ntp}_{\theta}^{\mathfrak{K}}([s, i, j], d)_{\rho})$$

The unification success score is set to zero to avoid early abort which resulting in no parameter updates.

All training atoms are ground atoms.

Neural link prediction-Auxiliary loss

At the beginning of training all subsymbolic representations are initialized randomly. When unifying a goal with all facts in a KB we consequently get very noisy success scores in early stages of training.

This is the reason for Auxiliary Loss in training time.

The form of the Auxiliary Loss is similar to formalization.

$$\mathcal{L}_{\text{ntp}\lambda_{\theta}^{\mathfrak{s}}} = \mathcal{L}_{\text{ntp}_{\theta}^{\mathfrak{s}}} + \sum_{([s,i,j],y) \in \mathcal{T}} -y \log(\text{complex}_{\theta}(s,i,j)) - (1-y) \log(1 - \text{complex}_{\theta}(s,i,j))$$

limitation on optimization

The limitation is the neural network is used to representing the KB and the prover module. In this paper, the NTP have no early abort except unification abort.

NTPs as described above suffer from severe computational limitations since the neural network is representing all possible proofs up to some predefined depth. In contrast to symbolic backward

Optimization methods

Batch proving

batch version unification model

K-max Gradient Approximation

in order to deal with the lack of gradient problem from max function, a K max heuristic method for gradient computation is proposed.

Only K unification successes are kept after unification process.