Formulae embeddings based on proof dependencies

Piotr Piękos

29 czerwca 2019

1 Introduction

The goal of the analysis is to create embeddings for formulas. And contrary to popular papers about embeddings in ATP to create them solely on conjectures and premises dependencies with DeepWalk[**?**] and Word2vec like methodology. Training of embeddings doesn't make use of any semantic or even syntactic or lexical features of the formulae. That fact makes these embeddings promising to analyse, because they could turn out to be totally orthogonal to classic formulae embeddings, and therefore allow to combine them together to make state of the art model stronger.

2 General Description

In DeepWalk[**?**] the methodology is based on random walk on the graph with probabilities of transition determined by the degree of a node. These walks create sentences which are then analysed in a word2vec fashion (like NLP sentences) to create embeddings. My approach is similar, but instead of using degree of a node i use slightly modified version of that which is supposed to better model importance of connection between nodes.

In this case we don't have a classic graph, because we have "grouped edges", directed group of edges represent that "roots" of the connections together are useful for proving the child. Additionally we have multigraph, we can have edges between two the same nodes, but in different "edge groups".

To transform our multigraph of dependencies we assign a score to each connection.

$$s_{i,j} = \sum_{\{G:(i,j)\in G\}} \frac{1}{|G|}$$

So for example if we have theorem 1 and theorem 2 s.t. theorem 2 has 2 proofs which are based on theorem 1, and one of these proofs uses 2 theorems

in total and the other one 3. Then

$$s_{th_1,th_2} = \frac{1}{2} + \frac{1}{3}$$

.

Next we transform these scores into transition probabilities with standard softmax function.

$$p_{i,j} = \frac{\exp s_{i,j}}{\sum_{k \in N(i)} \exp s_{i,k}}$$

where $N(i)$ is the set of neighbours of i

Having transition probabilities we can apply DeepWalk and standard Word2vec methodology. I won't dive into details here.

3 Limitations

Unfortunately Word2vec approach is based on known a priori corpora which means that we can't create embeddings for formulae which we hadn't seen at the moment of making embeddings. That makes it impossible to use these embeddings for proofs of unknown conjectures. This is a significant limitations. Nevertheless it can still be applied for example for

- finding new proofs for conjectures for which we already have proofs
- finding proofs for conjectures that don't have atp proofs (dependencies graph doesn't have to contain only atp dependencies)

4 Best benchmark

The main question to be asked here is: **but is that representation useful?** Do we gain anything by making that auxillary task, in the end even in the embedding training we still have a binary label for a pair (conjecture, premise) that even covers a little with standard prediction of usefulness. What's different is that

- we have window sizes bigger then 1
- these windows are symmetric, we have to predict a context, but we don't care which was proved with what. Although it seems unnatural word2vec context also seems less natural then the more naive "predicting the next word" embedder, which failed.

In order to test if these changes of our setup improved semantics captured by our embeddings we should compare it with a neural network that is based on one-hot encoding. That would represent embedding based on the same knowledge but without complicated tweaks.

Unfortunately I didn't have time for that comparison, but I do realize that's actually a priority.

5 Testing - Premise Selection

In order to find some kind of quality estimation of the embeddings I used these embeddings for premise selection. As it is a rough estimation I created embeddings on the whole dataset, then splitted it into train and test set. So the embeddings are a little leaked and therefore their performance metric is probably a little higher then it would be in a perfectly fair setting.

For the dataset consisting 1135 theorems and 15907 dependencies I created embeddings with the methods described above.

Then i splitted it into 75% training set and 25% test set. On the training test i used concatenation of embeddings of conjecture and premise as features I modeled usefulness of the premise in the proof of the conjecture, I added 3 negative premises (useless) per every positive example.

Then with the embeddings i created premise rankings for the test 25% and tried to prove as many of them with E-PROVER.

I trained 2 models: Logistic Regression, LightGBM I also for a benchmark compared it with 'model' that randomly permutes possible premises. Although i tried it only once so the random permutation model estimator has very high variance.

Results:

Model	PROVED %
Random Permutation	23.5
Linear Regression	23.23
LightGBM	27.1

There is an improvement of the model over the random permutations in the LGBM version, but not in the linear model. Additionally many unclarities in the evaluation process (embedding leak, random permutation variance, test set choice variance, etc.) makes it impossible to say that the progress is significantly different then random.

Further analysis is required.

6 Next Steps

- **Creating better comparison model** - as explained in section 4
- **Training on larger dataset** - dataset of dependencies was relatively small and embeddings are known to have big dataset demands
- **Testing in proof guidance setting** - The goal of embeddings is to make them contain semantic information. If they do in fact contain semantic information they would be useful in proof guidance, as processing time is main focus of optimization.

- **Testing more hyperparameters** - There are a lot of hyperparameters that were chosen ad hoc by intuition. (skip gram training, number of embeddings dimensions, negative samples within embeddings creation, negative samples within premise selection model creation etc. etc. even learning rate was 'optimized' only until i saw that something is training.
- **Merging the two approaches** - Merging that approach with standard (embedding based on parsing tree with RecNN). That would be the last upgrade even after many not mentioned here, but that would be the goal of the whole analysis.

7 Conclusion

Proof dependencies based embeddings seem to be interesting because of their ortogonality and many possible extensions. Nevertheless they also come with a lot of drawbacks and limitations.

Literatura

[1] Bryan Perozzi et al.: DeepWalk: Online Learning of Social Representations `https://arxiv.org/pdf/1403.6652.pdf`