An Evaluation of Approaches to Train Embeddings for Logical Inference (Student Abstract)

Yasir White[1], **Jevon Lipsey**[2], **Jeff Heflin**[3]
[1]Los Angeles Pierce College
[2]Colorado College
[3]Lehigh University
y.white005@gmail.com, jevonlipsey1029@gmail.com, heflin@cse.lehigh.edu

Abstract

Knowledge bases traditionally require manual optimization to ensure reasonable performance when answering queries. We build on previous neurosymbolic approaches by improving the training of an embedding model for logical statements that maximizes similarity between unifying atoms and minimizes similarity of non-unifying atoms. In particular, we evaluate different approaches to training this model.

Introduction

Backward-chaining is an algorithm for logic-based inference that operates by starting with a goal and systematically working backward through a series of rules and known facts to determine the conditions required to achieve the goal. Traditional backward chaining reasoners often rely on a brute-force approach to explore potential solutions, which can lead to inefficiencies as the complexity and scale of the problem increases. Even relatively small knowledge bases of thousands of statements can results in searches of millions of nodes if they have not been carefully designed by a knowledge engineer. Previous work looked at learning a scoring model to direct the search along promising paths and compared chainbased, termwalk, and unification as a means for solving this problem (Arnold and Heflin 2022). Our work improves on the previous unification approach to learn meaningful embeddings for logical statements.

This work contributes to a general neurosymbolic approach for logical reasoning that has three key components: 1) an embedding model that maps logical statements to vectors, 2) a scoring model that represents the likelihood that a path leads to a successful answer, and 3) a *guided* reasoner, where the choices of a backward-chaining reasoner are scored by the neural model. Prior approaches focused on improving the scoring model and guided reasoner; here we attempt different training methods for the embedding model to learn more useful embeddings and increase the overall performance.

Our Approach

The workflow of our system is shown in Figure 1. Starting with a knowledge base (KB) of facts and rules, we use a forward-chaining reasoner to infer new facts from the existing information. From these new facts, we randomly substitute constants with variables and divide the resulting list into one hundred training queries and one hundred test queries. We then solve the training queries using a backward chaining reasoner, exploring all possible paths to a solution. For each path, we assign $< goal, rule, score >$ tuples to the results. Here, $goal$ refers to the target query, $rule$ denotes the logical statement(s) used in the proof, and $score$ represents the effectiveness of a rule, with a score of 1 indicating a successful rule and 0 indicating a rule that does not lead to a solution. Although our work is currently restricted to Datalog (function-free Horn logic), we believe that many of the ideas can be extended to first-order languages with different/greater expressivity.

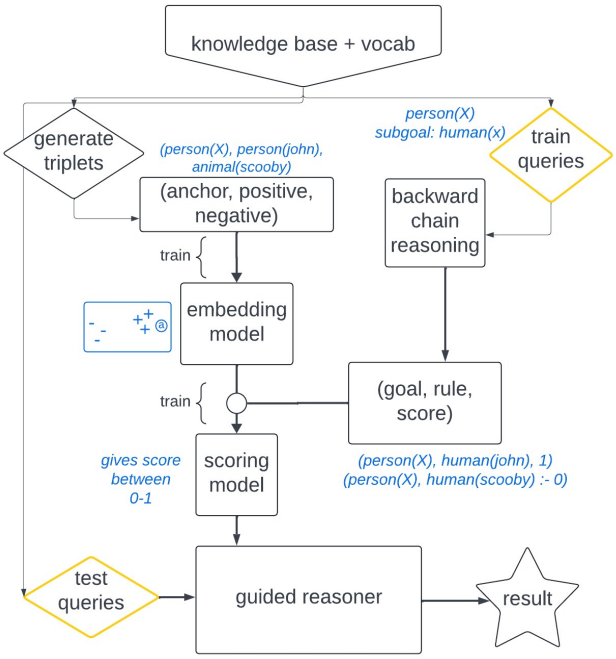

Figure 1: Representation of model structure

We have a two-step training process. First, we learn em-

beddings for atoms using triplet loss. The original approach for training the embedding model involved generating a list of atoms (logical facts) at random, and for each atom, selecting from the same list a unifying (positive example) atom and an atom that does not unify (negative example). Together, the positive, negative, and initial atom, also known as the *anchor*, are mapped onto a 50 dimensional embedding space with Triplet Loss (Vassileios Balntas and Mikolajczyk 2016). Then the goal/rule/score tuples are converted into vectors using the embedding model. Finally, supervised learning with a two-layer neural net is used to train a scoring model. When given an unseen atom to map, the original model tends to be effective, but suffers when answering some queries.

We attempted to improve on this embedding method in two ways. First, we increased the likelihood to generate more outlier atoms, Second, we define specific mutations for the atom to generate optimal training data. We define outlier atoms as atoms that do not appear often from a uniform random generation, but that have significant semantics, and are often very valuable for the embedding model to train on (e.g., $mom(X, X)$). Queries that appear to be outlier atoms often result in the model searching a significant amount of nodes before arriving at a solution. We're able to produce more outlier atoms with a 60% chance by repeating a variable or constant more than once, successfully producing data that can help the model learn that the same substitution must be applied to all occurrences of a variable in an atom.

An important observation we made is that the conditions for defining positive and negative atoms are already well defined. For example, given an anchor atom like $mom(X, john)$, we can convert this into a positive or equivalent atom by replacing X with a constant, such as $mary$. To make it a negative atom, we can replace $john$ with a different constant, such as $jill$. To derive a positive or equivalent logical statement from an anchor, we can change a variable into another variable or constant, or vice versa. On the other hand, to obtain a negative or non-equivalent statement, we can either replace a constant with another constant or alter the predicate itself, such as changing `mom` to a different predicate.

Defining these rules allows us to change the original generation of triplets into something much more effective, and further allows us to control the data our model receives for training. We generate a list of anchor atoms as before, but instead of *finding* the positive and negative atom, we simply generate them by *mutating* the properties of the anchor atom at random.

The initial randomized triplet generation approach typically results in many positive, negative, and anchor triplets that inherently repeat since they can be rearranged for equivalence. It turns out with our mutation approach, we can easily limit and control the amount of repeats and enrich the data while still keeping it organic. This adjustment captures rare but potentially significant semantics that could improve the ability of the model to handle unseen queries.

The other improvement to the embedding model involves a technique that focuses on training with the hardest samples. During training at every n-epochs, we validate the model's performance and continue refining it using half of the data, focusing on the training samples with the highest loss. This forces the model to learn and focus on difficult triplets that will help it solve queries quickly.

Reasoner	*Size*	*Nodes explored*
Standard	250	17,204.0
Previous Unification	250	224.0
New Embeddings	250	90.4
Standard	375	167,297.8
Previous Unification	375	207,489.7
New Embeddings	375	33,280.0

Table 1: Results from three reasoners, Nodes explored is an average taken across 5 different KBs.

Evaluation

To test our improvements, we consider two different KB sizes: 250 statements and 375 statements. For each size, we generate a set of 5 synthetic KBs. The 250 was trained on 100k triplets, and the 375 was trained on 200k triplets, since a larger knowledge base implies more training data is needed to capture any important information. We have three representative reasoners, 1) a standard backward-chaining reasoner, 2) the original unification approach for backward-chaining reasoners, and 3) our combined approaches to improving embeddings for a guided-reasoner. Respectively named, Standard, Previous Unification, and New Embeddings.

In Table 1, we report on the initial results. For each of the 5 KBs, we generated 100 unique queries. We collected the mean nodes explored across the 500 queries (100 per KB) to obtain our *Nodes explored* metric displayed in the table. Results show that our model outperforms other methods significantly, between a 5x and 188x improvement in nodes explored compared to the baseline without learning, and between a 2x and 6x improvement over our previous approach to embeddings. Future work will test larger and more realistic knowledge bases, demonstrating the impact of improvement on real-world knowledge bases and inference performance.

Acknowledgments

This work was conducted as part of an REU site supported by the National Science Foundation under Grant No. CNS-2051037.

References

Arnold, A.; and Heflin, J. 2022. Learning a More Efficient Backward-Chaining Reasoner. In *Proc. of the 10th Annual Conf. on Advances in Cog. Systems (ACS-2022)*. Arlington, VA, USA: Cog. Systems Foundation.

Vassileios Balntas, D. P., Edgar Riba; and Mikolajczyk, K. 2016. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *British Machine Vision Conference (BMVC)*. BMVA Press.