Learning a More Efficient Backward-Chaining Reasoner

Alex Arnold ALEXARNOLD2024@U.NORTHWESTERN.EDU
Northwestern University

Jeff Heflin JEH3@LEHIGH.EDU
Lehigh University, 113 Research Dr., Bethlehem, Pennsylvania 18015

Abstract

Neuro-symbolic AI and deep learning models have been used to mimic logical reasoning. In this paper, we provide a path guidance system for logical inference. We train a feed-forward neural network to provide a backward chaining reasoner with information on which order to attempt subgoals and which rules to apply first. A key element of our approach is learning embeddings for logical atoms that are informed by unification. By using these embeddings to construct a guided reasoner, we saw large reductions in the number of nodes explored by the reasoner.

1. Introduction

Despite the recent breakthroughs in deep learning and neural networks, symbolic learning and knowledge representation still play a vital role in AI advancements and practical uses in domains such as the medical field. Statistical AI and deep learning are adept at finding trends in large data samples, and can be applied without comprehensive preexisting knowledge about the subject matter. While the results they output are often convenient and easy to use, they are also known to be *black boxes* in that it is difficult to generate a convincing explanation of these results. On the other hand, symbolic AI systems can often provide human-readable proofs of their conclusions, but are generally only useful when the problem is already well understood. These algorithms also can not handle edge cases or exceptions well if they are not explicitly coded into the model. Neuro-symbolic AI emerged to combine these two fields in order to harness the power of deep learning while also remaining explainable.

This paper goes in a different direction than previous work in the field in that all actual reasoning will be done by a traditional backward chaining reasoner. Backward chaining is sound with proper implementation, but it is not complete as there are multiple issues that can lead to inefficiency or even infinite loops. In a "pure" version of the algorithm, which rule to apply and which atom to expand next is arbitrary. In languages like Prolog, the path through the search tree is a depth first search, which requires knowledge base (KB) designers to order rules specifically and intentionally in order to optimize the search. If a knowledge base is poorly designed, it can lead to cycles that jeopardize the completeness of the algorithm. These issues require targeted human intervention, and as knowledge bases grow larger, such engineering becomes increasingly difficult. Various heuristics have been used to successfully speed up logical inference, but they often require training on all

axioms in the knowledge base or storing information in a database (Sharma & Goolsbey, 2017). Groundbreaking work such as AlphaGo (Silver et al., 2017) has provided insights into how deep learning can be used to make a seemingly impossible graph search possible. Given the vast size of current knowledge bases, incorrect backward chaining paths can lead to large inefficiencies. The goal of this paper is to create a sub-symbolic representation using atom unification in order to then design a path guidance system for reasoning algorithms, which would prevent some costly incorrect choices that arbitrary searches would take. This is the first step in a larger research program that aims to make it possible to design KBs in a purely declarative fashion, and to avoid optimizing them for specific queries.

2. Background

2.1 Neuro-symbolic AI

One of the major ongoing research areas in neuro-symbolic AI is conducting logical inference using deep learning. Approaches like TransE (Bordes et al., 2013) and work based on it are able to add new facts without inputting new knowledge into the system using novel representations of symbolic relationships. Previous work has also attempted to create neural networks that function as pseudo-reasoners on logic problems. Work like First Order Logical Neural Networks (Kijsirikul & Lerdlamnaochai, 2016) have created feed-forward neural networks that can function on noisier data than traditional reasoners, and therefore can deal with exceptions better than other reasoners. Traditional reasoning techniques like backward chaining have been used as inspiration for recursive neural networks that represent logical operations and replaces steps like unification with differentiable processes (Rocktäschel & Riedel, 2017). Other work has taken inspiration from AlphaGo and NLP systems by using reinforcement policy based learning and attention in order to improve problem solvers (Crouse et al., 2021). Mathematical proofs have also been a subject of research, with automated theorem provers being used to predict which proven statements will be needed to prove a given theorem (Kaliszyk et al., 2015). Our work is most similar to these last two, in that we seek to predict which rules will be needed to solve a given query and the best path through a search tree.

2.2 Meta-reasoning

Our work also falls under the field of meta-reasoning. Meta-reasoning refers to the control of time, effort, and strategies put towards reasoning, and the amount of each is chosen based on some chosen heuristic (Ackerman & Thompson, 2017). Meta-reasoning approaches can take inspiration from meta-cognition, or humans own reasoning about their own thought processes.

2.3 Horn Logic and Unification

In First Order Logic, atomic sentences are the most basic statements. They consist of a predicate or function and a list of arguments that can be either constants (denoted with lowercase characters in Datalog) or variables (denoted with upper case characters in Datalog). For example, the intended interpretation of $daughterOf(arya\_stark, ned\_stark)$ is that "Arya is the daughter of Ned." If

an atom appears in the knowledge base as a fact (or can be inferred through the use of other rules) it is said to be true. We used a subset of first order logic known as *Horn logic*. Horn logic clauses can be written as an implication of the form $A_1 \wedge A_2 \wedge ... \wedge A_i \rightarrow B$, where A_n and B are all positive atoms. The atoms on the left are **body**, while the single positive atom is the **head**. If the body is empty, then the rule is considered a fact. An added benefit of using Horn logic is that contradictions are impossible, as negation is not a valid logical operator.

Datalog is a function-free version of Horn logic that uses a Prolog syntax. Additionally, existential variables are not allowed in the head of the clause.Like Prolog, the head of a rule is written on the left hand side and the ":-" symbol is used for implication. Terms that begin with a capital letter are variable, while all other terms are constants. A short example knowledge base is listed bellow.

Example Knowledge Base

- childOf(arya_stark, ned_stark).
- childOf(ned_stark, rickard_stark).
- childOf(arya_stark, catelyn_stark).
- parentOf(X,Y) :- childOf(Y,X).
- grandparentOf(X,Y) :- parentOf(Z,Y), parentOf(X,Z).

This small knowledge contains four facts and two inference rules. Additional facts, such as **parentOf(rickard_stark, ned_stark)**, can be inferred using reasoning algorithms.

Reasoning in first-order logic (and its fragments) depends on the concept of unification to determine if a one atom can be substituted for another. Formally, two logical expression unify if there is a substitution that makes them syntactically identical. For two Datalog atoms, this means they have to have the same predicate and cannot have different constants in the same term position. Furthermore, a variable cannot appear in the same position as different constants. For example, $p(X, X)$ would unify with $p(a, Y)$ or $p(a, a)$, but not $m(X, X)$ or $p(a, b)$.

2.4 Forward and Backward Chaining Reasoners

Two of the most common reasoners for horn logic knowledge bases are forward and backward chaining (Poole & Mackworth, 2018). Each start with an atomic query, but work towards an answer in opposite ways. Forward chaining starts with all the facts (rules with no body) and uses inference rules to generate facts until it can no longer generate any new facts. Backward chaining starts with the query and attempts to unify it with heads of rules and recursively prove the sub-goals until all sub-goals are proven. Backward chaining tends to be preferable when there is a specific query in mind while forward chaining is more useful to infer all of the new facts from an existing knowledge base. Each can be represented as a tree search problem. Backward chaining starts at the root, while forward chaining starts at the leaves. Forward chaining is less efficient when trying to prove only one query, but is better at inferring many or all possible facts in a knowledge base. Backward chaining will therefore be the algorithm this work seeks to optimize.

2.5 Logical Embeddings

In order to apply neural networks to the problem, one must first find a way to convert facts and rules into a numeric vector form. The result of such a conversion is called an *embedding*. We argue that

logical embeddings have to meet two criteria: they have to be accurate representations, and they have to be applicable to multiple knowledge bases. Structure must be the only feature that embeddings take into account, and they must not take into account any semantic meaning of the symbols themselves. The embeddings of **childOf(arya_stark, ned_stark)** should not rely on the meaning of **arya_stark**, since this can lead to the model over-fitting to the knowledge base used in training (Ebrahimi et al., 2021). Therefore, all names should be represented as meaningless symbols, as that is more representative of how actual reasoners interpret them. The order of arguments in atoms must be preserved to retain properties for unification, but the order of rule bodies is irrelevant, although in contemporary knowledge bases order is integral to backward chaining execution. Previously, rules have been represented using a bag-of-words style approach and "chain-based vectorization" for atoms based on unity (Crouse et al., 2021) and others utilized the semantic meanings of words in the knowledge base to create embeddings of symbols (Socher et al., 2013).

The method used by Crouse et al. (2021) is most similar to ours. Their approach involved breaking down clauses into multiple different "patterns" that are each linear chains from a predicate symbol to a variable or constant. These were intended to provide representations that could be used to determine structural similarity. This approach to vectorization is carefully engineered based on intuitions about first-order reasoning, and the final vector was created by hashing the pattern and setting specific bits of the vector based on how many times each pattern appeared in a clause. If not careful, this approach can either results in vectors that are much larger than needed, or in collisions between different patterns.

3. Methods

In order to created a guided reasoning system, we first created embeddings of atoms that were informed by unification. We then created random knowledge bases to reason on, and represented the rules and facts using the previously created embedding model. A feed-forward neural network was then created to score goal/rule pairs in the random knowledge base by how likely that path was to lead to a solution. A guided reasoner then used this model to choose which paths to attempt first.

3.1 Embeddings that Respect Unification

The first step to generating embeddings based on unification was to create a vector encoding of an atom. This work is most similar to Crouse et al. (2021), who were also inspired by atom unity as a basis for atom representation. However, our key contribution is that the embeddings are actually learned, rather than engineering. Since the semantic content of the atoms are not important, we created random atoms from a pool of ten predicates ($p0, p1, \ldots p9$) with a random arity from one to four, ten variables ($X0, X1, \ldots X9$), and 100 constants ($a0, a1, \ldots a99$).[1] Unary and binary predicates were set to be more common to reflect conditions in real-world knowledge bases. Other work has allowed semantically similar predicates such as "grandpaOf" and "grandfatherOf" to unify (Rocktäschel & Riedel, 2017), but this is not possible in classical logic (where predicates must match exactly) and will not be considered here since the random predicates lack any meaning at all.

1. More complex KBs could be supported by starting with a larger pool, but this will lead to larger training times

Individual predicate arity was held constant across all experiments, so if $p3$ had an arity of two when generating embeddings, it retained the same arity when training and testing on a specific KB. To construct the input representation, we concatenate a one-hot encoding of the predicate with a one-hot encoding of each argument in the atom. If there were less than four arguments, the encoding was padded with zeros to keep a consistent length for all encodings. Each encoding had a length of 560 (10 for the representation of the predicate and 110 each for five possible arguments).[2] The output embedding space had a dimensionality of 20, resulting in a significant reduction in dimensionality that should capture the most important commonalities between atoms. An example of this process on a shortened atom is shown below in Figure 1.

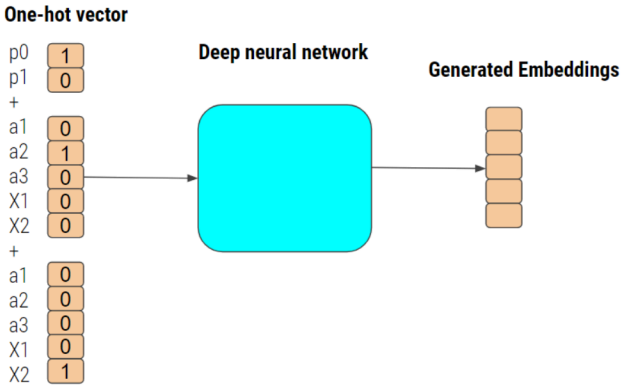

Figure 1. One-hot encoding of an example atom, **p1(a2,X2),** with two possible predicates, variables, and constants and a maximum arity of two

This approach can be used without loss of generality because any knowledge base can be canonicialized into a sufficiently large pool of standard predicates, variables, and constants. Their names do not matter so long as they are unique, so a straight-forward renaming can be applied without changing the semantics of the KB. Real world knowledge bases have a much larger number of predicates, variables, and constants, but an appropriate pool can be generated, and embeddings can be learned.

To learn our embeddings, we constructed a feed-forward neural network with one hidden layer. Our learning goal was that the cosine similarity of the embeddings of two atoms that unify would be closer than two atoms that did not unify. To achieve this, we used triplet loss as the loss function for the model. Triplet loss is a classless loss function, and has found success in the fields of computer vision and facial recognition (Chechik et al., 2010). It has also been used successfully to represent relationships in knowledge bases (Bordes et al., 2013). Triplet loss maximizes the distance between an anchor and a negative example and minimizes the distance between the anchor and a positive example. In this case, the anchor can be any atom. The positive example atom will be chosen to unify with the anchor atom, while the negative example will be chosen so that it does not unify with the anchor atom. In order to construct the triplets, a set of atoms was randomly generated and all

2. Our training data did not make full use of this input representation, as it had no predicates with arity greater than four.

possible unifying pairs were found to create the anchors and positive cases, and the anchor atom was "corrupted" by randomly changing the arguments in order to create a negative case. If this process was not able to create a non-unifying case with the same predicate, a random atom was selected to be the negative case.

3.2 Knowledge Bases and Reasoners

We generated random knowledge bases using the same predicates as those used to train the atom embedding model. They were kept to a limited rule length to reflect real world knowledge bases that do not tend to have overly long or complex rules (Russell & Norvig, 2009). Generated facts also only contained constants to retain a manageable computation time. Due to the properties of Horn logic stated above, these random knowledge bases could not contain contradictions that would complicate the generation process. A random knowledge base of size 150 was created with 80% facts and 20% rules with body lengths of up to 4, with smaller rules being more likely than larger rules to reflect conditions in many real world knowledge bases.

In order to create a model that predicts the best path, both forward and backward chaining reasoners were needed to construct training examples. All possible facts that could be inferred from a knowledge base were inferred using forward chaining, and a random fact was selected and random arguments replaced with variables to create new queries. The main advantage of using Datalog is that the deductive closure of any knowledge base is finite since Datalog lacks functions. This guarantees that forward chaining will eventually complete and find all possible facts. Backward chaining was then used to find all possible solutions to the query and generate a representation of the search tree. In the tree, a node consists of subgoals that remain to be proven. We choose one subgoal at random and then find all rules that apply to this subgoal (i.e., the head of the rules unify with the subgoal). We also selected the rules in a random order for each subgoal. This process is repeated recursively until a solution is found or a depth limit is reached. We extracted all (g, r) pairs from the tree, where g was a chosen subgoal and r was the rule that was chosen to expand the node. If a (g, r) pair was extracted from a path that eventually led to a solution (regardless of depth), we assigned it a score of 1. Otherwise, we assigned it a score of 0. We used a dynamic depth limiter to prevent infinite cycles. Since paths to shallower solutions are more efficient than those to deeper ones, so deeper paths were less pertinent than shallower ones. When the reasoner finds a solution, it only searches up to 1.5 times deeper in the search tree from then on.

The output of this process is a set of training examples, each consisting of a subgoal g, a rule r, and a binary classification of whether the rule ultimately leads to a solution of the original problem. Note, that this process will produce many more negative examples (where the rule does not lead to a solution) than positive ones. This an example of a class imbalance problem, which will be addressed in the next section. Additionally, we cannot sample the entire answer space, and the initial queries used in the test set were not present in the training data generated from this process. That being said, it is possible for them to share sub-trees within each search tree.

3.3 Binary Classifier for Path Guidance

Given our set of training examples, we can learn a binary classifier that determines whether a particular rule is a good step in the search for a solution to reasoning problem. This model would determine a "score" for how likely it is that a particular rule will lead to an overall solution for each subgoal. We used the data generated from the previous process to train this model, where the inputs to model are the subgoal atom and the rule used, and the target is a binary variable indicating whether the subgoal/rule combination led to a solution. All inputs were constructed using the embeddings generated with the model we introduced in 3.1.

An important question is how to create an embedding of a rule, given that we have embeddings for its constituent atoms. For a (g, r) pair, our algorithm constructs the embedding by concatenating the unity embedding of subgoal g, the unity embedding of the head of rule r, and the sum of unity embeddings of each atom in the body of r. We hypothesize that adding them together will still capture the semantics of all of the body atoms while also having the desirable property of ignoring the ordering of the atoms in the body. Since our embedding process of each atom tends to place unifiable atoms close together, our rule embeddings will also tend to have close embeddings if they only differ in a variable renaming. We selected a supervised learning approach and used feed-forward neural networks. The network contained three linear layers and sigmoid activation functions.

As mentioned above, our training data has a class imbalance problem, where there were many more negative examples than positive examples. It is well-known that unbalanced classes can lead to problems in machine learning. We used a common solution, which is to oversample the examples (add additional copies of them).

In a reasoner guided by these results, both the query/subgoal and rule used would be chosen in accordance to the model above. The score given by the binary classification will rank all possible subgoal/rule combinations, and the reasoner will search the tree in that order as opposed to from left to right as in a standard reasoner or randomly as in our data generation reasoner. To test if this is more efficient, we evaluated against a traditional depth first search backward chaining reasoner. We then utilized the same methods with embeddings generated via an autoencoder as a baseline. Our evaluation metric was the number of nodes explored, and runtime was not considered.

4. Experiment and Results

4.1 Unification Embeddings

To learn the embeddings of atoms, we created a dataset with 53k atom triplets (anchor, positive example, negative example). The model was trained for 30 epochs and achieved a training loss of 0.2022. We then compared the embeddings of unifying pairs with those of non-unfying pairs (Figure 2). As desired, unifying atoms tended to have a very high cosine similarity, while non-unifying atoms tends tended to have a cosine similarity closer to 0.

A Kolmogorov-Smirnov two sample test was run to determine if the two samples are significantly different. With a test statistic of 0.755 and $p < 0.001$, there is statistically significant evidence that the two distributions are different. This indicates that the embeddings were able to sufficiently

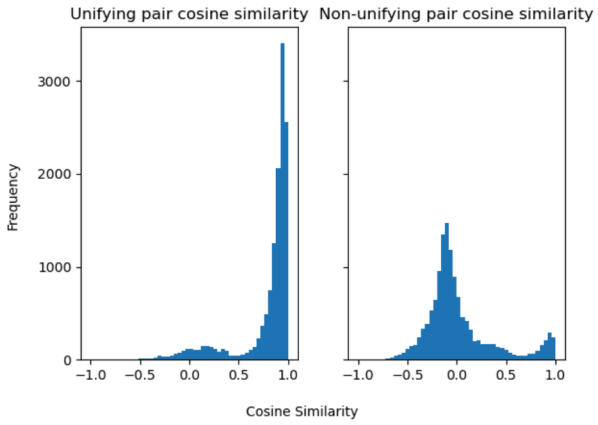

Figure 2. Cosine similarity of unifying and non-unifying atom pairs

capture whether two atoms unify, as well as some representation of their structure. These embeddings were then used to represent all the atoms in subsequent experiments.

4.2 Path Guidance

To evaluate our our approach, we generated a random knowledge base with 150 statements using the approach described in Section 3.2. The same knowledge base was used to train the rule classifier and to test the reasoners. The model was trained for 500 epochs and achieved a training loss of 0.31434.We note that because we are not training on all possible queries, the system still needs to learn to generalize. We argue that such a process could still be useful for large, frequently-used KBs that might benefit from more efficient reasoning, as the training process would be a one-time, up-front cost that can be fully automated. Nevertheless, our long-term goal is to learn from a collection of KBs, and generalize to a previously unseen KB.

4.2.1 Experiment 1

Two reasoners were constructed for the purpose of testing. Neither had cycle checking, but both utilized a depth limiter to prevent infinite cycles. The standard reasoner always picked the leftmost goal to expand and chose rules from the knowledge base in the order they appear in the knowledge base. In the guided reasoner, the order was chosen by using the model to score each goal/rule combination and sorting them from largest to smallest score. Each reasoner still performed a depth-first search. Using the forward-chaining reasoner, we generated 100 novel random queries that were not used to train the model, and each query was evaluated by both reasoners. The guided reasoner utilized path guidance for goals that were of depth three or less then switched over to using the traditional reasoner as the current implementation is still not optimized for a large number of nodes. We calculated the number of nodes explored by the guided reasoner minus the number

nodes explored by the standard reasoner. Figure 3 show the differences by query, sorted from best improvement over the standard reasoner to worst.

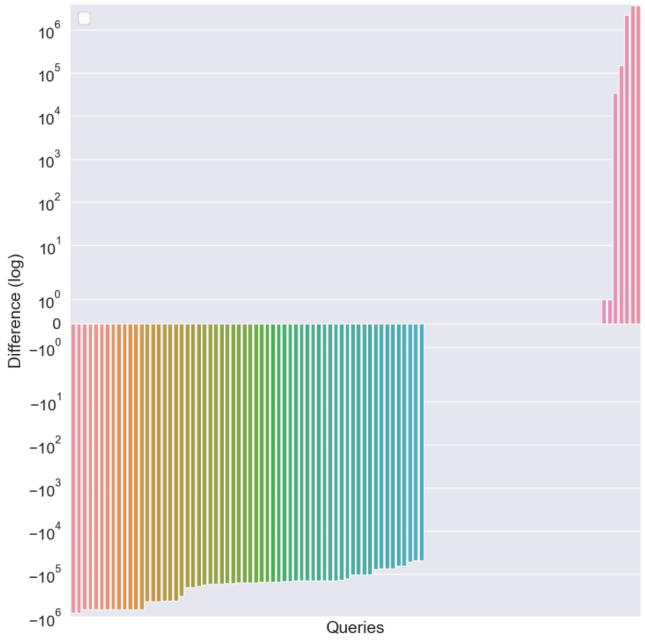

Figure 3. Change in nodes explored between standard and guided reasoner, plotted on a logarithmic scale.

The traditional reasoner on average explored $199,175.1$ nodes with a median of $140,888.5$, while the guided reasoner on average explored $122,919.4$ with a median of 5.0. This indicates that the guided reasoner was often able to find solutions with a small search, but there were occasionally significant outliers. In 62% of cases the guided reasoner explored fewer nodes than the traditional reasoner, while it explored the same number 31% of the time and only performed worse 7% of the time. However, it must be noted that the three worst examples had increases of over 1000%.

While the reasoner seemed to get lost in a rabbit hole a few times, for the majority of cases it provided a significant improvement compared to a standard reasoner, often with over a 99% decrease. There is certainly more work to be done to improve the guidance system and avoid costly drops in efficiency, but these preliminary results are promising.

4.2.2 Experiment 2

To evaluate the contribution of our unification-based embedding approach, we decided to compare against an alternative form of generating embeddings. We chose an autoencoder (Rumelhart et al., 1985) as our baseline. An autoencoder is a neural network that learns encodings of unlabeled data by trying to map them to a lower-dimensional vector, and then minimize the loss when recovering the original data by inverting the mapping. It was not provided any additional information about the properties of the atom, such as atoms it unifies with. So that our experiment would only have

one independent variable, we used the same training set, represented inputs using the same one-hot vectors, and output a vector in an embedding space with the same dimensionality as our proposed embedding approach (20).

A third reasoner was constructed that utilized the autoencoder model to provide inputs to the network that evaluated goal/rule pairs. The autoencoder was trained for 200 epochs and achieved a training loss of 0.00701. Both reasoners were trained on the same 3798 training examples. The autoencoder reasoner was also trained for 500 epochs and achieved a training loss of 0.33144. One hundred new queries that were not used to train either model were generated. We calculated the number of nodes explored by the autoencoding guided reasoner minus the number nodes explored by the unification embedding reasoner, as shown in Figure 4.

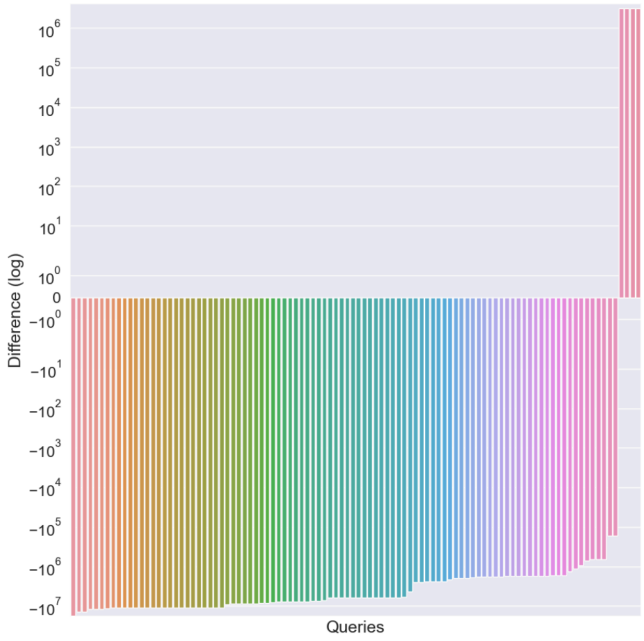

Figure 4. Change in nodes explored between autoencoder and unification embedding guided reasoners

The guided reasoner based on autoencodings performed much worse than our proposed guided reasoner on the majority of queries, with the former exploring one million or more additional nodes than the latter in 98% of queries. These results provide additional evidence embeddings that respect unification capture beneficial information about atoms and rules that can improve path guidance.

These results were also compared against the standard reasoner. We found the minimum solution depth found by one of the three reasoners and compared it against the average nodes explored for that depth (Figure 5). We found that across the board, the autoencoder reasoner performed worse than the standard reasoner, while the unification-based reasoner performed either similarly or better than the standard reasoner regardless of solution depth. Overall, the standard reasoner explored $224,918.55$ on average, the autoencoder reasoner explored $6,171,486.49$, and the unification

guided reasoner only explored $187,885.66$. Although we expected our unification-aware approach to perform better, we were still surprised by the how poorly the autoencoder performed. We suspect that its performance could be improved by significantly increasing the number of training examples for the goal/rule evaluation stage.

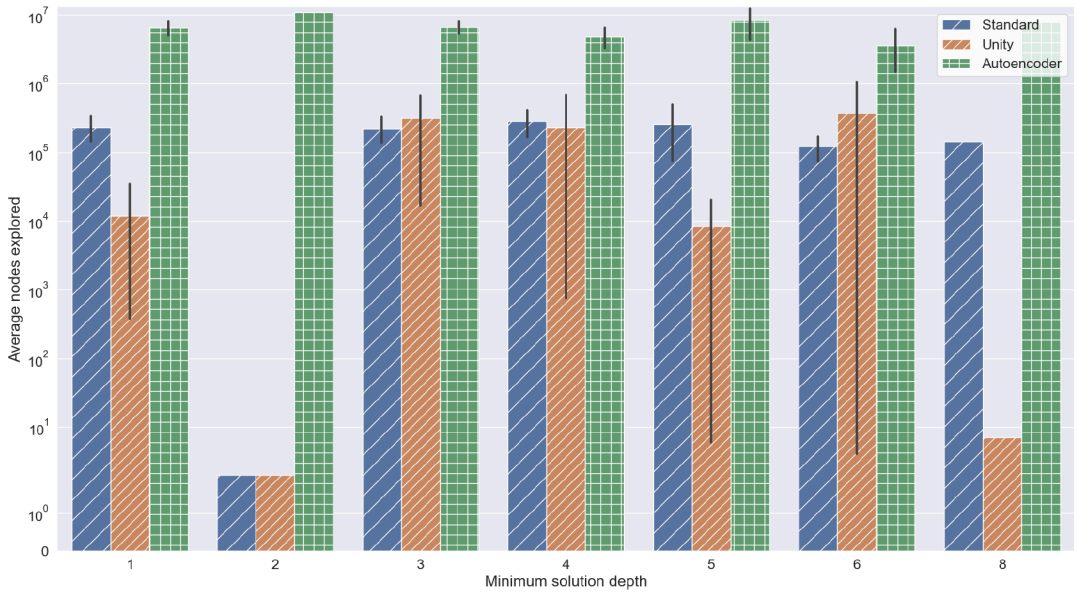

Figure 5. Minimum solution depth vs average number of nodes explored found by the reasoner

5. Conclusion and Future Work

In this work we introduced a model for generating atom and rule embeddings that respect unification and used those embeddings to create a guided reasoner that is able to reason more efficiently on the majority of queries compared to standard reasoners. This reasoner did not require manual optimization of knowledge bases to see marked improvements in the number of nodes explored. This work was a preliminary step towards a larger goal of creating models that can generalize across much larger knowledge bases through the use of transfer learning and training on multiple knowledge bases. We believe these results are a promising first step towards this overall goal.

Additionally, other training approaches like regressors and reinforcement learning should be considered. Training and testing on larger real world knowledge bases is essential as well. We are also interested in expanding to full first-order logic that includes more complex rules.

Acknowledgements

We thank Dr. Mohamed Trabelsi for his help in the design of the project. This work was conducted as part of an REU site supported by the National Science Foundation under Grant No. CNS-2051037.

References

Ackerman, R., & Thompson, V. A. (2017). Meta-Reasoning: Monitoring and Control of Thinking and Reasoning. *Trends in Cognitive Sciences*, *21*, 607–617. From `https://www.sciencedirect.com/science/article/pii/S1364661317301055`.

Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. *Advances in Neural Information Processing Systems, 26*.

Chechik, G., Sharma, V., Shalit, U., & Bengio, S. (2010). Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, *11*, 1109–1135.

Crouse, M., et al. (2021). A Deep Reinforcement Learning Approach to First-Order Logic Theorem Proving. *35th AAAI Conference on Artificial Intelligence, AAAI 2021*, *7*, 6279–6287.

Ebrahimi, M., Eberhart, A., Bianchi, F., & Hitzler, P. (2021). Towards bridging the neuro-symbolic gap: deep deductive reasoners. *Applied Intelligence*, *51*.

Kaliszyk, C., Urban, J., & Vyskocil, J. (2015). Efficient semantic features for automated reasoning over large theories. *Twenty-fourth International Joint Conference on Artificial Intelligence*.

Kijsirikul, B., & Lerdlamnaochai, T. (2016). First-Order Logical Neural Networks. *International Journal of Hybrid Intelligent Systems*, *2*.

Poole, D. L., & Mackworth, A. K. (2018). *Artificial intelligence: Foundations of computational agents*. Cambridge University Press, 2 edition.

Rocktäschel, T., & Riedel, S. (2017). End-to-end differentiable proving. *Advances in Neural Information Processing Systems, 31*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation*. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Russell, S. J., & Norvig, P. (2009). *Artificial intelligence: A modern approach*. Prentice Hall, 3rd edition.

Sharma, A., & Goolsbey, K. (2017). Identifying useful inference paths in large commonsense knowledge bases by retrograde analysis. *Proceedings of the 31st AAAI Conference on Artificial Intelligence*.

Silver, D., et al. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*.

Socher, R., Chen, D., Manning, C. D., & Ng, A. Y. (2013). Reasoning with neural tensor networks for knowledge base completion. *Advances in Neural Information Processing Systems*, (pp. 1–10).