MATHEMATICAL REASONING IN LATENT SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

We design and conduct a simple experiment to study whether neural networks can perform several steps of approximate reasoning in a fixed dimensional latent space. The set of rewrites (i.e. transformations) that can be successfully performed on a statement represents essential semantic features of the statement. We can compress this information by embedding the formula in a vector space, such that the vector associated with a statement can be used to predict whether a statement can be rewritten by other theorems. Predicting the embedding of a formula generated by some rewrite rule is naturally viewed as approximate reasoning in the latent space. In order to measure the effectiveness of this reasoning, we perform approximate deduction sequences in the latent space and use the resulting embedding to inform the semantic features of the corresponding formal statement (which is obtained by performing the corresponding rewrite sequence using real formulas). Our experiments show that graph neural networks can make non-trivial predictions about the rewrite-success of statements, even when they propagate predicted latent representations for several steps. Since our corpus of mathematical formulas includes a wide variety of mathematical disciplines, this experiment is a strong indicator for the feasibility of deduction in latent space in general.

1   INTRODUCTION

Automated reasoning has long been considered to require development of logics and "hard" algorithms, such as backtracking search. Recently, approaches that employ deep learning have also been applied, but these have focused on predicting the next step of a proof, which is again executed with a hard algorithm (Loos et al., 2017; Gauthier et al., 2017; Lederman et al., 2018; Bansal et al., 2019b).

We raise the question whether hard algorithms could be omitted from this process and mathematical reasoning performed entirely in the latent space. To this end, we investigate whether we can predict useful latent representations of the mathematical formulas that result from proof steps. Ideally, we could rely entirely on predicted latent representations to sketch out proofs and only go back to the concrete mathematical formulas to check if our intuitive reasoning was correct. This would allow for more flexible and robust system designs for automated reasoning.

In this work, we present a first experiment indicating directly that theorem proving in the latent space might be possible. We build on HOList, an environment and benchmark for automated theorem provers based on deep learning (Bansal et al., 2019b) which is makes use of the interactive theorem prover HOL Light (Harrison, 1996), an interactive proof assistant. The HOList theorem database comprises of over 19 thousand theorems and lemmas from a variety of mathematical domains, including topology, multivariate calculus, real and complex analysis, geometric algebra, and measure theory. Concrete examples include basic properties of real and complex numbers such as $(x^{-1} = y) \Leftrightarrow (x = y^{-1})$, and also well-known theorems, such as Pythagoras' theorem, Skolem's theorem, the fundamental theorem of calculus, Abel's theorem for complex power series and that the eigenvalues of a complex matrix are the roots of its characteristic polynomial.

We focus on *rewrite rules* (or *rewrites* in short). Rewrites are only one of several proof tactics in HOL Light, but they enable powerful transformations on mathematical formulas, as they can be given arbitrary theorems as parameters. For example, the formula $3^2 = z$ can be rewritten to $3 \cdot 3 = z$ by performing a rewrite with the parameter $x^2 = x \cdot x$. Alternatively, a rewrite may diverge (as it operates recursively) or it may return the same formula – in both these cases we consider the rewrite to *fail*. For instance, in the example above, the rewrite would fail if we used equation $x + y = y + x$

as a rewrite parameter instead, since the expression $3^2 = z$ does not contain any $+$ operators to match with.

In our experiments, we first train a neural network to map mathematical formulas into a latent space of fixed dimension. This network is trained by predicting – based on the latent representation being trained – whether a given rewrite is going to succeed (i.e. returns with a new formula). For successful rewrites we also predict the latent representation of the resulting formula. To evaluate the feasibility of reasoning in latent space over two steps, we first predict the latent representation of the result of a rewrite, then we evaluate whether the predicted latent representation still allows for accurate predictions of the rewrite success of the resulting formula. For multi-step reasoning beyond two steps, we predict the future latent representations based on the previous latent representation only - without seeing the intermediate formula. Our experiments suggest that even after 4 steps of reasoning purely in latent space, neural networks show non-trivial reasoning capabilities, despite not being trained on this task directly.

2 RELATED WORK

Our work is motivated by deep learning based automated theorem proving, but is also closely connected to model based reinforcement learning and approaches that learn to predict the future as part of reinforcement learning.

Model based reinforcement learning is concerned with creating a model of the environment while maximizing the expected reward (e.g. Ha & Schmidhuber (2018)). Already early works have shown that predicting the latent representations of reinforcement learning environments with deep learning is sometimes feasible - even over many steps (Oh et al., 2015; Chiappa et al., 2017). This can enable faster training, since it can preempt the need for performing expensive simulations of the environment. Predicting latent representation was also proposed in Brunner et al. (2018) as a regularization method for reinforcement learning.

One recent successful example of model based reinforcement learning is Kaiser et al. (2019), where the system learns to predict the pixel-wise output of the Atari machine. However this approach is based on actually simulating the environment directly in the "pixel space" as opposed to performing predictions in a low dimensional semantic embedding space. More related to our work is Piotrowski et al. (2019), which attempts to learn to rewrite simple formulas. The goal is there again is to predict the actual outcome of the rewrite rather than a latent representation of it. In Dosovitskiy & Koltun (2017), they predict "expected measurements" as an extra supervision in addition to the reward signal.

Graph neural networks have been used for premise selection in higher order logic (Wang et al., 2017) and more recently by Paliwal et al. (2019) as the core component of DeepHOL, a neural theorem prover for higher order logic. In this work, we build upon their neural network architecture, but utilize it for a different task.

3 HOL LIGHT

HOL Light (Harrison, 1996) is an *interactive proof assistant* (or interactive theorem prover) for higher-order logic reasoning. Traditionally, proof assistants have been used by human users for creating formalized proofs of mathematical statements manually. Although they come with limited forms of automation, it is still a cumbersome process to formalize proofs, even when it is already available in natural language. Some large scale formalization efforts were conducted successfully in HOL Light and Coq (Coq), for example the formal proofs of the Kepler conjecture (Hales et al., 2017) and that of the four color theorem (Gonthier, 2008). They required significant meticulous manual work and expert knowledge of the system.

Lately, there have been several attempts to improve the automation of the proof assistants significantly by so called "hammers" (Kaliszyk & Urban, 2015). Still, traditional proof automation lacks the mathematical intuition of human mathematicians who can perform complicated intuitive arguments. The quest for modelling and automating fuzzy, "human style" reasoning is one of the main motivation for this work.

3.1 REWRITE TACTIC IN HOL LIGHT

The HOL Light system allows the user to specify a *goal* to prove, and then offers a number of *tactics* to apply to the goal. A tactic application consumes the goal and returns a list of subgoals. Proving all of the subgoals is equivalent to proving the goal itself. Accordingly, if a tactic application returns the empty list of subgoals, the parent goal is proven.

In this work, we focus on the rewrite tactic (`REWRITE_TAC`) of HOL Light, which is a particularly common and versatile tactic. It takes a list of theorems as parameters (though in this work we only consider applications of rewrite with a single parameter). Parameters must be an equation or a conjunction of equations; possibly guarded by a condition. Given a goal statement T and parameter P_i, the rewrite tactic searches for subexpressions in T that match the left side of one of the equations in P_i and replaces it with the right side of the equation. The matching process takes care of variable names and types, such that minor differences can be bridged. The rewrite procedure is recursive, and hence tries to rewrite the formula until no opportunities for rewrites are left. The rewrite tactic also has a set of built-in rewrite rules, representing trivial simplifications, such as $FST(x, y) = x$. Note that `REWRITE_TAC` uses "big step semantics", meaning that the application of each individual operation can perform multiple elementary rewrite steps recursively. For more details on `REWRITE_TAC`, refer to the manual (HOL Light Rewrite Tactic Reference).

4 REASONING IN LATENT SPACE

We embed higher-order logic statements into a fixed dimensional embedding space by applying a graph neural network to a suitably chosen graph representation of the corresponding formula. The embedding is trained on predicting the outcome (success or failure) of a large number of possible formula rewrite operations. Note that formulas can be quite complex as they are arbitrary typed lambda expressions in higher order logic.

For technical reasons, we will distinguish between two latent embedding spaces $L = \mathbb{R}^k$ and $L' = \mathbb{R}^k$ ($k = 1024$) corresponding to two distinct embeddings for each formula, learned by two different networks.

We have trained three different models. S denotes the set of syntactically correct higher-order logic formulas in HOL Light.

1. Rewrite success prediction $\sigma : S \times S \longrightarrow [0, 1]$,
2. Rewrite outcome prediction $\omega : S \times S \longrightarrow [0, 1] \times L$,
3. Embedding alignment prediction $\alpha : L \longrightarrow L'$.

These networks and their purposes are described in detail in later subsections. Alternatively, we could use a single fixed embedding space with a single model predicting its own future embedding on the rewritten statement. That network could be trained end-to-end and reach better performance without the need of aligning the embedding spaces, removing the need for α. Here, we opted for a more controlled setup that relies on a fixed embedding network σ, trained for the sole task of predicting whether rewriting statement T by P is successful. This way we can rely on a fixed embedding method and run more detailed ablation analyses. Merging σ and ω, is left for future work.

4.1 TRAINING DATA

We start with the theorem database of the HOList environment (Bansal et al., 2019b), which contains 19591 theorems in its theorem database, approximately ordered by increasing complexity. This is split into 11655 training, 3668 validation, and 3620 testing theorems. To generate our training data, we generate all pairs T, P of theorems from the training set, where P must occur before T in the database (to avoid circular dependencies). We then interpret theorem T as a goal to prove and try to rewrite T with P using the `REWRITE_TAC` of HOL Light. This can result in three different outcomes:

1. T is rewritten by theorem P successfully, and the result differs from T.
2. The rewrite operation terminates, but failed to change the input theorem T.

3. The rewrite operation times out or diverges (becomes too big).

In our experiments, we consider only the first outcome as successful, i.e. when the application finishes within the specified time limit and changes the target, as a successful rewrite attempt. Each training example, therefore, consists of the pair (T, P), the success/fail-bit of the rewrite (1 for successful rewrites, 0 for failed rewrites), and, for successful tactic applications, the new formula that results from the rewrite, which we denote with $R(T, P)$.

4.2 BASE MODEL ARCHITECTURE AND TRAINING METHODOLOGY

The rewrite success prediction model $\sigma(T, P)$ is trained on the training set of theorems in the HOList benchmark. The training task is to predict the success or failure of the REWRITE_TAC application.

We used a two-tower network (without weight sharing) with embedding towers $\gamma : S \longrightarrow L$ and π, one for each of the two formulas T and P. Both towers are graph neural networks as described in Paliwal et al. (2019). Both of them embed the supplied formula in a fixed dimensional embedding space $\mathbb{R}^k$. The concatenated embeddings are then processed by a three-layer perceptron c with rectified linear activation, which is followed by p, a single output linear function predicting the logit and is trained by logistic regression on the success/fail-bit of the rewrite. Formally: $\sigma(T, P) = p(c(\gamma(T), \pi(P)))$.
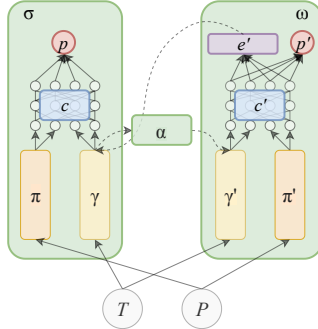

Figure 1: Depiction of network architecture

4.3 OUTCOME PREDICTION MODEL

In addition to our base model, we train $\omega(T, P)$. This model has an identical two-tower embedding architecture as σ, but with a larger combiner network and an extra prediction layer e' to predict embedding vector of the outcome of the rewritten formulas. Here the embedding towers are denoted by γ' and π', the combiner network is c' and the two linear prediction layers are p' and e'. That is: $\omega(T, P) = (p'(c'(\gamma'(T), \pi'(P))), e'(c'(\gamma'(T), \pi'(P))))$.

This model predicts both the success or failure of applying REWRITE_TAC and for successful rewrites, the latent representation $\gamma(R(T, P))$ of the result. While p' is trained to predict the success of rewriting T by P by logistic regression, e' is trained to predict $\gamma(R(T, P))$ by minimizing the squared error.

4.4 EMBEDDING ALIGNMENT MODEL

Since ω and σ produce latent vectors $\gamma(T) \in L$ and $\gamma'(T) \in L'$ in different spaces, we need to align those spaces enable deduction purely in the embedding space. (Merging σ and ω will remove the need for the α, however in our current setup we keep the embedding and deduction components separate).

Given an initial statement T, we predict the approximate value of $\gamma(R(T, P))$, but in order to reason multiple steps in the embedding space alone, without explicitly computing $R(T, P)$, we need $\gamma'(R(T, P))$ to compute the outcome prediction of ω. For that, we train a translation model $\alpha : L \longrightarrow L'$ which predicts $\gamma'(T)$ given an approximation of $\gamma(T)$. Note that α does not see $T \in S$ as an input, it makes its prediction based on the latent space representation $\gamma(T)$ of T alone. This allows us for reasoning multiple steps ahead in the latent space without constructing any of the intermediate formulas explicitly.

4.5 REASONING

After we have trained our three models on the training set theorems (and theorem pairs) of the HOList benchmark, we can use them to perform rewrites in the latent space alone.

We use $\sigma : S \times S \longrightarrow [0, 1]$ as a quality metric for the propagated embedding vector. $\sigma(T, P) = p(c(\gamma(T), \pi(P)))$ was trained for predicting whether theorem P rewrites T. Given an approximation

$\tilde{v}_T$ of the latent representation $\gamma(T)$, we can evaluate $\tilde{\sigma}_T(P)$ defined by $\tilde{\sigma}_T(P) = p(\tilde{v}_T, \pi(P))$ for a large number of tactic parameters P. This is compared with true rewrite successes of T by P to assess the quality of the approximation.

To evaluate multiple steps of reasoning in the latent space, start with formula T_1 and rewrite by theorems $P_1, \cdots P_k$ in that order. For reasoning in the latent space, we only use approximate embeddings vectors of the resulting formulas. To assess the quality of the overall reasoning, the same sequence of rewrites is performed with the actual formulas and the final approximate embedding is evaluated with respect to the formula resulting from the sequence of formal rewrites.

In latent space $L = \mathbb{R}^k$ we from some initial theorem $T_1 \in S$. The following schema indicates the sequence of predictions performed in latent spaces L and L':

$$T_1 \xrightarrow{\gamma'} l_1' \in L' \xrightarrow{e'(\bullet, P_1)} l_2 \in L \xrightarrow{\alpha} l_2' \in L' \xrightarrow{e'(\bullet, P_2)} l_3 \in L \xrightarrow{\alpha} l_3' \in L' \xrightarrow{e'(\bullet, P_3)} \cdots$$

This way, we approximate the following sequence of deductions in the latent space L alone, without producing any intermediate formulas. This is compared with the following formal sequence of rewrites:

$$T_1 \xrightarrow{R(\bullet, P_1)} T_2 \xrightarrow{R(\bullet, P_1)} T_3 \xrightarrow{R(\bullet, P_3)} \cdots$$

That is, l_2 approximates the latent vector of T_2, that is $\gamma(R(T_1, P_1))$ and l_3 approximates the latent vector of T_3, that is $\gamma(R(R(T_1, P_1), P_2))$, by construction.

By a slight abuse of notation we will refer to the operation of one step of approximate deduction in the latent space by $\omega \circ \alpha$, as it is composition α and a subnetwork of ω.

5 EXPERIMENTS

This section provides an experimental evaluation that aims to answer the following question: Is this setup capable of predicting embedding vectors multiple steps ahead? We explore the prediction quality of the embedding vectors (for rewrite success) and see how the quality of predicted embedding vectors degrades after $\omega \circ \alpha$ is used for predicting multiple steps in the latent space alone.

5.1 NEURAL NETWORK ARCHITECTURE DETAILS

Our networks σ and ω both have two towers, which are 16-hop graph neural networks with internal node representations of 128 dimensions. The output of each of the two towers is fed into a layer that expands the dimension of the node representation to 1024 with a fully connected layer with shared weights for each node. This is followed by maximum pooling over all nodes of the network. The two resulting embedding vectors are concatenated along with their element-wise multiplication, and are processed by a three layer perceptron with rectified linear activation between the layers.

Figure 2: Graph showing the distribution of successful rewrites for each parameter P, computed over all (T, P) pairs in the database. Note that a small portion of parameters can be used for many theorems. This allows the Random and Usage baselines to achieve above-chance performance by predicting success based on parameter alone.

The same architecture is used for both σ and ω, but the two networks do not share weights. Also, ω has larger layers in its combiner network c' than σ in c, (1024 units each in c' vs. 128 units in the layers of c). This was necessary for producing good quality predictions of the embedding vector of the outcome of the rewrite, but unnecessary for predicting the rewrite success alone.

σ and ω are trained with $g = 16$ groups of $l + 1 = 16$ instances in each batch: one successful example in each group and l random negatives. However all other instances in other groups are used as negative instances for for each goal as well and they are considered negative regardless of whether they would rewrite it – this is justified by the fact that only a few theorems rewrite any given
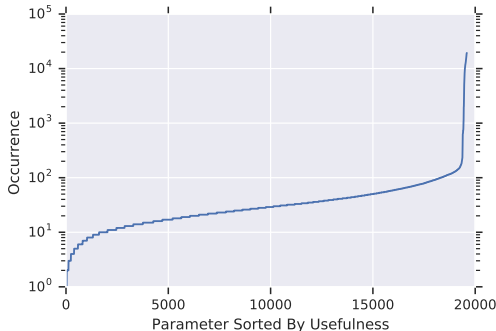
T_i so this introduces only a small amount of uncorrelated label noise. This training methodology is motivated by the fact that evaluating the combiner network is much cheaper then computing the embedding using the graph neural network. Based on Alemi et al. (2016), we expect that hard negative mining would improve our results significantly, but it is left for future work.

5.2 EVALUATION DATASET

In order to measure the performance of our models after multiple deduction steps are performed, we generate datasets $D_0, D_1, \ldots, D_r$ successively by applying rewrites to a randomly selected set of examples. We start with all theorems from the validation set of HOList, denoted by D_0. We create D_i from D_{i-1} by selecting a random subset of 2000 statements from the previous step and 200 random tactic parameters $\mathbf{P}_T$ to rewrite by for each statement. Formally, D_i is defined by $\{R(T, P) \mid P \in \mathbf{P}_T\}$.

5.3 EVALUATION OF REWRITE PREDICTION MODEL

In order to evaluate the performance of σ in isolation we need to compare it with carefully selected baselines:

1. As Figure 2 shows, a few theorems are much more frequently applicable than others. We want to see how the prediction of rewrite success performs based on the rewrite parameter alone if we ignore the theorem to be rewritten. One way to establish such a baseline we just feed a randomly selected theorem T' to σ instead of T to predict its rewrite success.

2. A stronger "baseline" is achieved by utilizing the ground-truth to make the best prediction possible based on knowing P but still being independent of T (the theorem to be rewritten). This is the best achievable prediction that does not depend on T.

3. As we have trained σ and ω only on pairs of theorems from the original database, the models exhibit increasing generalization error as we evaluate them on formulas that with increasing number of rewrites. First, we measure the errors these models make if the theorem for the last step is evaluated directly. This gives an upper bound on the rewrite success prediction by ω, since noisier embedding vectors end up with worse results on average.

4. Finally, we want to measure how rate at which the latent vectors degrade as we propagate them in embedding space as described in Subsection 4.5.

In order to measure the performance of our models after performing a given number of rewrite steps starting from the theorem database, we measure the tactic success prediction quality of σ using predicted embeddings. To do so, we compute the ROC curve of the predictions and use the area under the curve (AUC) as our main metric. Higher curves and higher AUC values represent more accurate predictions of rewrite success. We measure how ROC curves change as we use different approximations of $\gamma(T)$.
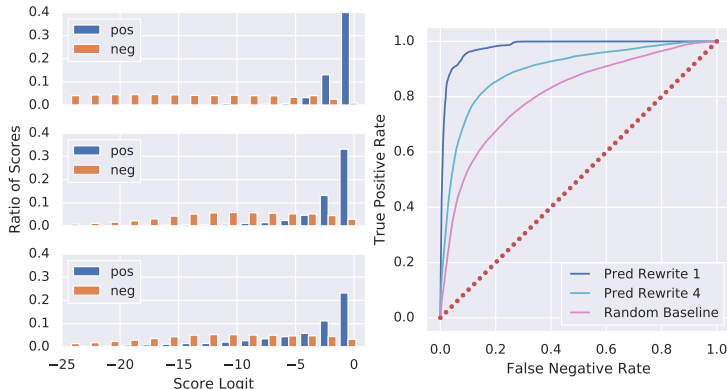

Figure 3: Left: Histograms of scores for first step of rewrite predictions (top), fourth step of rewrite predictions (middle), and random baseline (bottom). Right: Corresponding ROC curves.
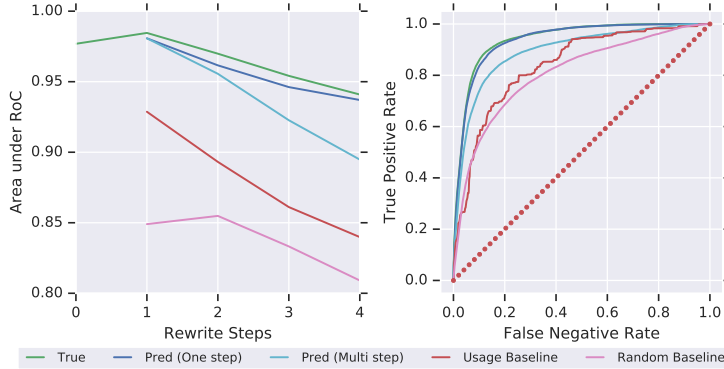
Figure 4: Left: Comparison of Area under ROC curves for each embedding method. Right: Comparison of ROC curves for each embedding method after the fourth rewrite.
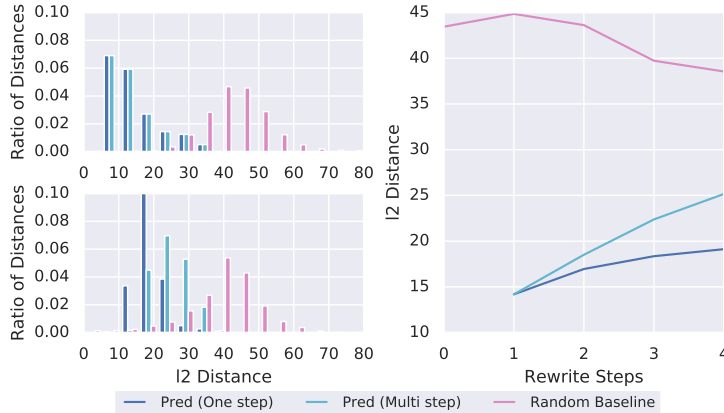

Figure 5: Left: Histogram of l^2 distances between each embedding method and the true embedding for rewrite steps 1 and 4. Right: Plot showing the mean l^2 distances across rewrite steps. **Pred (One step)**: expected error $\|\gamma(R(T, P)) - \alpha(e(\gamma'(T), \pi'(P)))\|_2$ between the true embedding of the rewritten statement $R(T, P)$ and that of the prediction network $\omega \circ \alpha$ one step ahead in D_k. **Pred (Multi step)**: expected error for $\gamma(R(...R(T, P_1), ..., P_k))$ in D_k after k approximate rewrites. **Random baseline**: expected l^2 distance $\|\gamma(T_1) - \gamma(T_2)\|_2$ between the "true" embeddings of two randomly chosen theorems from D_k.

6 ANALYSIS

Figure 3 shows the distribution of the theorem pair prediction score logits of $p(c(\gamma(T), \pi(P)))$, for the those "positive" pairs that rewrite and the "negative" pairs that do not rewrite. Note that the ratio is normalized separately for the positive and negative pairs as negative pairs occur much more frequently than positive pairs.

One can see that the quality of the rewrite success prediction degrades significantly after four steps of reasoning purely in the latent space, but it is still much better than the random baseline. This gives clear evidence that the embedding prediction manages to propagate much useful information over multiple steps in the latent space alone.

In Figure 4 we make further measurements and comparisons on the quality of reasoning in the embedding space. On the left hand side we measure five different metrics: The "True" curve assess the embedding $\gamma(T)$ computed directly from the target theorem T. The "Pred (One step)" curve uses the approximate embedding $\alpha(e(\gamma'(T_{i-1}), \pi'(P_{i-1})))$ for $T_i = R(T_{i-1}, P_{i-1})$. That is, we measure the degradation when performing a single step of embedding prediction. The "Pred (Multi-step)" curve uses multiple steps of predictions completely in the latent space as described in Subsection 4.5. The "Random Baseline" predicts the rewrite success based on the latent vector of a random statement instead of the correct one. "Usage Baseline" is based on the constant prediction that ranks the parameters by how probably they rewrite any statement in the theorem database. This prediction

7
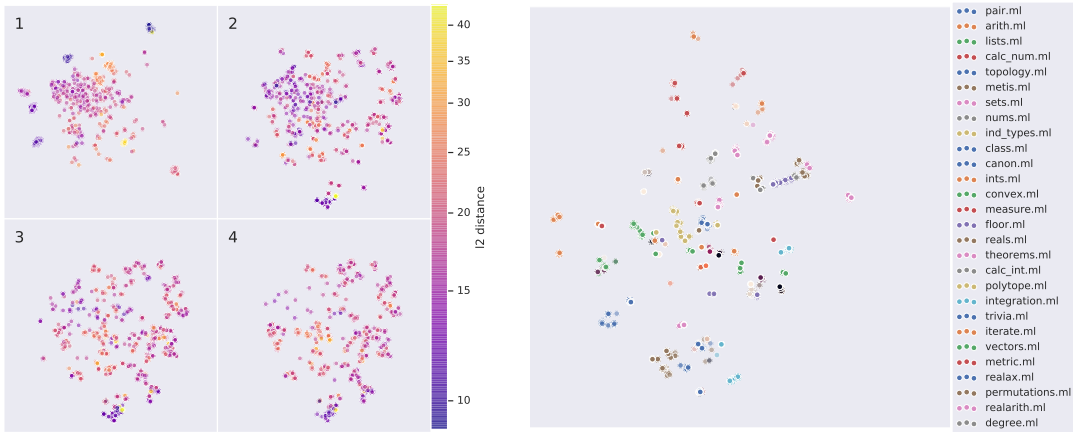
(a) Visualization of the embedding spaces produced by embedding prediction e' in the latent space after $1, 2, 3$ and 4 rewrite steps. The points are colored by their l^2-distance to the true embedding $\gamma(T)$.

(b) Visualization of the embedding space $\gamma(T)$ evolving over four rewrite steps. Theorems are colored by the area of mathematics they originate from, sourced from the theorem database. The brightness of the embeddings corresponds to the rewrite step in which it was generated, with more recent embeddings being darker.

is also independent of T. One can see that our model could perform reasoning for 4 steps in the embedding space and still retain a lot of the predictive power of the original model.

In order to appreciate the above results one should keep in mind that none one of our models α, σ and ω were trained on statements that were already rewritten. All the training was done only on the theorems present in the initial database. The reduction of prediction performance is apparent from downward trajectory of the "Pred (Multi step)" curve, which isolates this effect from that of the error accumulated by the embedding predictions, the effect of which is measured indirectly by the "True" curve.

In Figure 5 we have measure the l^2 distance of the predicted embedding vectors versus that of the true embedding vectors $\gamma(T)$ of formulas after multiple rewrite steps in the latent space. These results are consistent with our earlier findings on success of rewrite predication after rewrite steps in the latent space: while there is some divergence of the predicted embedding vectors from the true embedding vectors (as computed from the rewritten statements directly), the predicted embedding vectors are significantly closer to the true embedding vectors than randomly selected embeddings.

7 CONCLUSION

In this paper we studied the feasibility of performing complex reasoning for mathematical formulas in a fixed (1024) dimensional embedding space. We proposed a new evaluation metric that measures the preservation semantic information under multiple reasoning steps in the embedding space. Although our models were not trained for performing rewrites on rewritten statements, nor were they trained for being able to deduce multiple steps in the embedding space, our approximate rewrite prediction model $\omega \circ \alpha$ has demonstrated significant prediction power as far as 4 approximate rewrite steps performed in the latent space. Although it seems likely that these results can be significantly improved by better neural network architectures, hard negative mining and training on rewritten formulas, our methods showcases a simple and efficient general methodology for reasoning in the latent space. In addition, it proposes an easy to use, fast to train and crisp evaluation methodology for representing mathematical statements by neural networks.

It is likely that such representations prove helpful for faster learning to prove without imitating human proofs like that in DeepHOL-Zero Bansal et al. (2019a), given that premise selection is a closely related task to predicting the rewrite success of statements. Self-supervised pre-training or even co-training such models with premise selection could prove useful as a way of learning more semantic feature representations of mathematical formulas.

REFERENCES

Alexander A Alemi, François Chollet, Geoffrey Irving, Niklas Eén, Christian Szegedy, and Josef Urban. Deepmath-deep sequence models for premise selection. In *Advances in Neural Information Processing Systems*, pp. 2235–2243, 2016.

Kshitij Bansal, Sarah M Loos, Markus N Rabe, and Christian Szegedy. Learning to reason in large theories without imitation. *arXiv preprint arXiv:1905.10501*, 2019a.

Kshitij Bansal, Sarah M Loos, Markus N Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher-order theorem proving. *ICML 2019. International Conference on Machine Learning*, 2019b.

Gino Brunner, Manuel Fritsche, Oliver Richter, and Roger Wattenhofer. Using state predictions for value regularization in curiosity driven deep reinforcement learning. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 25–29. IEEE, 2018.

Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *CoRR*, abs/1704.02254, 2017. URL http://arxiv.org/abs/1704.02254.

Coq. The Coq Proof Assistant. http://coq.inria.fr. URL http://coq.inria.fr.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *ICLR 2017*, 2017.

Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Tactictoe: Learning to reason with HOL4 tactics. In *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46, pp. 125–143, 2017.

Georges Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 2450–2462. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.pdf.

Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. A formal proof of the Kepler conjecture. In *Forum of Mathematics, Pi*, volume 5. Cambridge University Press, 2017.

John Harrison. HOL Light: A tutorial introduction. In *FMCAD*, pp. 265–269, 1996.

HOL Light Rewrite Tactic Reference. URL https://www.cl.cam.ac.uk/~jrh13/hol-light/HTML/REWRITE_TAC.html. Accessed: 2019/09/23.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for Atari. *arXiv preprint arXiv:1903.00374*, 2019.

Cezary Kaliszyk and Josef Urban. Hol (y) hammer: Online atp service for hol light. *Mathematics in Computer Science*, 9(1):5–22, 2015.

Gil Lederman, Markus N. Rabe, and Sanjit A. Seshia. Learning heuristics for automated reasoning through deep reinforcement learning. *CoRR*, abs/1807.08058, 2018. URL http://arxiv.org/abs/1807.08058.

Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, 2017.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in Atari games. In *Advances in neural information processing systems*, pp. 2863–2871, 2015.

Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. *arXiv preprint arXiv:1905.10006*, 2019.

Bartosz Piotrowski, Chad Brown, Josef Urban, and Cezary Kaliszyk. Can neural networks learn symbolic rewriting? 2019.

Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In *Advances in Neural Information Processing Systems*, pp. 2786–2796, 2017.