Can a transformer learn to prove by looking at proofs?

Sawinder Kaur
Department of Computer and Information Science and Engineering
Syracuse University

Abstract

A mathematical proof is a sequence proof states derived by application of valid axioms and inference rules. Thus, a derivation of a proof can be perceived as a translation task from one proof state to another. State-of-the art approaches aim to predict the set of inference rules which when applied to one proof state, produces the next proof state. This work aims to learn this translation without taking into account the inference rule at action. Thus, in this work, the proof derivation is defined as a sequence-to-sequence translation task using two slighlty different versions the data. The dataset explored in this work is the Metamath's 'set.mm' partition, restricted to proof lengths of upto 10 and 15.

1 Introduction

The Metamath dataset Megill [2006] comprises of formal mathematical proofs, axioms, and statements encoded in a standardized format for a variety of mathematical domains such as set theory, number theory, and logic. However, this work focuses on automating proof generation for the domain of set theory.

Purpose of a theorem prover: A mathematical proof is a series of logical expressions where each expression is derived from a previous one based on some well-established rules (or axioms/ inference rules) of the domain. The task of proof generation requires searching over the set of inference rules at each proof step and executing them until a terminal condition is obtained. The possibility to apply multiple inference rules at a particular instance makes the problem even harder.

With the expansion of applications of neural networks, the task of theorem proving has also been automated using neural networks. An Automatic Theorem Prover (ATP) takes a conjecture as input and based on the axioms (or inference) rules of the domain, arrives at a conclusion of whether the conjecture is True or False Abdelaziz et al. [2022]. The steps followed to arrive a True conclusion comprise the steps of the proof.

Representation of a proof in Metamath: For each rule, Metamath provides a series of inference rules as a representation of the proof. However, this representation is unable to express or take into account the current state of the proof in terms of a logical expression. Therefore, this work targets the proof stack trace which is a series of stack states where each stack entry contains a well-formed formula (wff). Figure 1 show an example of the proof stack trace for the rule 'mdp'. Here each line represents the state of the current proof stack and transition of one proof stack state to the next is a result of application of a valid inference rule.

State-of-the-art approaches Abdelaziz et al. [2022] aim to learn a valid and correct inference rule that needs to be applied in order to achieve proof state transitions aiming at correct and efficient proof generation. That is, the neural network takes as input the current proof state and predicts the next inference rule, which is followed by application of the predicted rule to update the model state which serves as the next input to the model. This process is repeated until a proof or a stopping condition has reached.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

```
-----------------------------------
mpd $p |- ( ph -> ch ) $.
disjoint variable sets: set()
  wph $f wff ph $.
  wps $f wff ps $.
  wch $f wff ch $.
  mpd.1 $e |- ( ph -> ps ) $.
  mpd.2 $e |- ( ph -> ( ps -> ch ) ) $.

mpd
['wff ph']
['wff ph', 'wff ps']
['wff ( ph -> ps )']
['wff ( ph -> ps )', 'wff ph']
['wff ( ph -> ps )', 'wff ph', 'wff ch']
['wff ( ph -> ps )', 'wff ( ph -> ch )']
['wff ( ph -> ps )', 'wff ( ph -> ch )', '|- ( ph -> ps )']
['wff ( ph -> ps )', 'wff ( ph -> ch )', '|- ( ph -> ps )', 'wff ph']
['wff ( ph -> ps )', 'wff ( ph -> ch )', '|- ( ph -> ps )', 'wff ph', 'wff ps']
['wff ( ph -> ps )', 'wff ( ph -> ch )', '|- ( ph -> ps )', 'wff ph', 'wff ps', 'wff ch']
['wff ( ph -> ps )', 'wff ( ph -> ch )', '|- ( ph -> ps )', 'wff ph', 'wff ps', 'wff ch', '|- ( ph -> ( ps -> ch ) )']
['wff ( ph -> ps )', 'wff ( ph -> ch )', '|- ( ph -> ps )', '|- ( ( ph -> ps ) -> ( ph -> ch ) )']
['|- ( ph -> ch )']
```

Figure 1: Example Proof

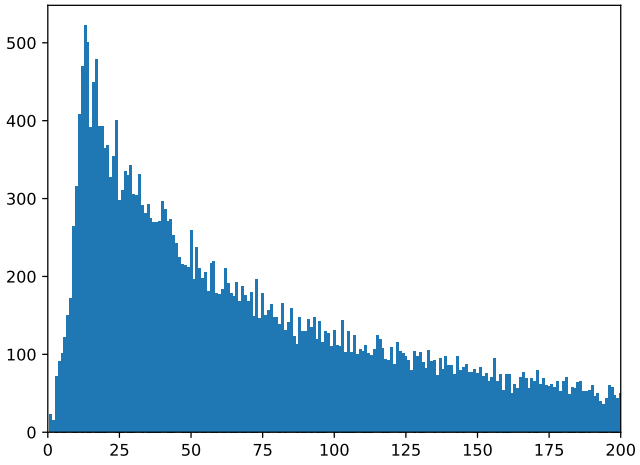

Figure 2: Proof lengths

Objective of this work: This work aims to investigate whether a model can predict the next state of the proof without the information of the rule or not. Specifically, the input to the neural network would be a proof state and the model is expected to predict the next proof state (or proof stack).

Thus, the main contribution and novelty of this work is that it formulates the proof generation problem as a sequence-to-sequence translation task and is independent of the inference rule used for transition from one proof stage to another. Moreover, state-of-the art approaches, which aim as predicting the next inference rule, rely on a theorem solver such as 'Eprover' Schulz [2002] to compute the next proof state. This work's approach has an additional benefit which renders the requirement pf such a solvernon-obligatory.

2 Preliminary Data Analysis

This section describes the terms used in this work and the preliminary data analysis done to extract their corresponding values.

Proof Length: This work considers the number of stack states in the proof stack trace as the length of the proof. Figure 2 shows the distribution of the proof of different length. Although, the dataset does contain proofs of lengths as high as 1400, the scope of this work is restricted to proofs of lengths upto 10 and 15.

Proof Length	Alphabet Size	Max Stack Length	Number of Rules	Dataset length
10	1647	8	1012	6686
15	6163	13	3206	33751

Table 1: Data information

Tokens and Alphabet: Each entry in the proof stack serves as a token and the collection of these tokens form the alphabet of the language under study. This is essential to compute the number of embeddings required to represent the language.

Maximum length of stack (Maximum Sequence length): It can be observed in Figure 1 that the length of the stack can vary for the same proof. The maximum length of the stack defines the maximum sequence length for the translation task at hand. This helps to generate suitable position embeddings. Table 1 provides the details of the alphabet size and maximum stack length for proofs of lengths upto 10 and 15. It can be observed that the maximum stack lengths for proofs of lengths upto 10 is lesser than the ones with length upto 15.

Number of Rules: In Table 1, the 'number of rules', represent the number of rules which have length upto the given proof length.

Dataset length: The dataset length provides the number of seq-to seq translations generated for the proofs of all the rules having compatible proof length. The formulation of the dataset is discussed in Section 4. It is important to note that a single rule contributes to multiple instances in the dataset, thus, the dataset length is larger than the number of rules having compatible proof length.

3 Problem Statement

For a Rule $\mathcal{R}$, if $< S_1, S_2, ...S_n >$ represents a proof stack trace where $S_n = \mathcal{R}$, this works aims to learn the transition $S_i \rightarrow S_{i-1}$ without the information about the inference rule which resulted in this transition.

Thus, if $\mathcal{M}$ represents the model, and S_i^j represents the the j^{th} token in stack state S_i, the objective is to minimize the difference between the predicted and the true tokens:

$$minimize \quad \forall \ell(\mathcal{M}(S_{i-1})^j, S_i^j) \tag{1}$$

4 Approach

The following section describes the dataset formulation for proof generation using the stack traces. The stack trace generally start from a single token which describes a particular literal is a well formed formula. This literal must be a part of the original conjecture (rule $\mathcal{R}$) to be proved. Thus, to make an informed decision at the starting, the stack trace is augmented with the conjecture to be prove. In other works, the starting point of the proof contributes the translation from the conjecture to the first (original) stack stage. Thus, the stack trace for Rule $\mathcal{R}$ is updated to $< \mathcal{R}, S_1, S_2, ..., S_n >$, here, $S_n = \mathcal{R}$. Each of these transitions form one stack stage to the next, contributes to a sample in a translation dataset. This section further explains the formulation of the dataset.

Once the dataset is generated, a transformer-based model $\mathcal{M}$, developed on the lines of 'Prooformer', is used to learn each of the translation task. It is important to note here that the tranlation task doesnot consider the whole proof stack trace at a time, that is, it just aims to learn one translation form S_i to S_{i+1} at a time. In other words, the transformer itself doesnot generate the whole proof.

A proof will be considered complete if starting form the original conjecture, it is possible to reach the same conjecture after repetitive application of the model $\mathcal{M}$. That is, for each stack stage, the model $\mathcal{M}$ has to correctly predict the next stack stage. The Section 4.1 provides the the dataset formulation, the Section 4.2 provides details of the model $\mathcal{M}$ and Section 4.3 explaines the proof generation step.

4.1 Dataset Formulation

As described above, the model $\mathcal{M}$ aims to learn the translation form one stack stage to another. The stacks themselves lack the information of the goal rule or the conjecture to be proved. Thus, the

model $\mathcal{M}$ would learn the translation task without considering the goal rule or the inference rule employed for the transition from one stack stage to another. The table 1 describes the dataset length for proofs different proof lengths.

However, the same transition can be a part of proofs for multiple conjectures. Thus, to disambiguate the translation task, we propose a slightly different variations of the translation task where each proof stack stage of prepended by the conjecture (Rule $\mathcal{R}$) itself. This helps provide the information about the goal to the model $\mathcal{M}$.

Therefore, we propose two translation tasks. The first translation task considers the updated proof stack trace ($< \mathcal{R}, S_1, S_2, ..., S_n >$, here, $S_n = \mathcal{R}$) and the second translation task adds Rule $\mathcal{R}$ to the front of each stack stage in the above-mentioned stack trace.

4.1.1 Translation Task 1

For a Rule $\mathcal{R}$, $< \mathcal{R}, S_1, S_2, ...S_n >$ represents a proof stack trace. Each transition from one proof stack stage to another is an instance of the sequence-to-sequence translation. Thus, each conjecture contributes to multiple instances in the dataset which are described as follows:

$$\begin{gathered}\mathcal{R} \rightarrow S_1 \\ S_1 \rightarrow S_2 \\ \cdots \\ S_{n-1} \rightarrow S_n (= \mathcal{R})\end{gathered}$$

The dataset consists of the collection of the above-mentioned source-target pairs for the all the rules having compatible proof length.

4.1.2 Translation Task 2

For incorporating the information of the Rule $\mathcal{R}$ in the stack, the Rule $\mathcal{R}$ is added to the front of each stack. Thus, for each Rule, the translation task contributes following samples to the dataset:

$$\begin{gathered}\mathcal{R}|\mathcal{R} \rightarrow \mathcal{R}|S_1 \\ \mathcal{R}|S_1 \rightarrow \mathcal{R}|S_2 \\ \cdots \\ \mathcal{R}|S_{n-1} \rightarrow \mathcal{R}|S_n (= \mathcal{R}|\mathcal{R})\end{gathered}$$

Now, each of the stack stage contains a sequence of formulas. To represent them in a numerical format, pytorch embedding generation has been used Pyt. This explores several different embedding sizes.

4.2 Model $\mathcal{M}$

The model used in this work is a Transformer-based model formulated on the same lines as 'Prooformer' (as discussed in the lectures). This original model generated an output of size equal to the embedding size. To map this output to a token in the alphabet, an additional linear layer added to the end output of the transformer model. The size of the layer is 'embedding size X alphabet size'. This is followed by a 'softmax' layer Tra.

Thus, for each output token, the model $\mathcal{M}$ generates a probability distribution over all the tokens in the alphabet. This helps employ cross entropy loss for training.

4.3 Proof Generation

The proof generation step consists of repetitive application of the above learned model $\mathcal{R}$ to generate the next proof stack staring from the rule to be proved. The repeated application of the model stops when the model results in the target rule or when the maximum proof length is reached.

$$
\begin{aligned}
S'_1 &= \mathcal{M}(\mathcal{R}) \\
S'_2 &= \mathcal{M}(S'_1) \\
&\cdots \\
S'_n &= \mathcal{M}(S'_{n-1})
\end{aligned}
$$

The proof generation is successful if S'_n is the same as that of the conjecture $\mathcal{R}$ to be proved.

5 Evaluations

For the evaluations, the aforementioned dataset is divided into disjoint training and test sets split in 80 - 20 ratio. Moreover, all results are averaged over 3 random seed values.

5.1 Metrics for Evaluations

This work considers two metrics of evaluation.

1. **Accuracy of prediction:** The accuracy of prediction refers to the percentage of correct translations done by the model $\mathcal{M}$. More specifically, this is the measure of correctness for the prediction made by the model $\mathcal{M}$. The work considers a translation to be correct if all the tokens present in the target stack are correctly predicted. Thus, the accuracy here is not restricted to correct prediction of one inference rule or one token, but the whole stack.
2. **Completed proofs**: This includes the percentage of rules for which the whole proof could be generated correctly.

5.2 Hyperparameters:

For training these models, a learning rate of 0.00005 has been used with 100 training epochs for proof length 10 and 150 epoches for proof length 15. The number of encoder-decoder layers are restricted to 2 for these evaluations and for parameter 'd-model', which is equal to the embedding size, 4 different values have been evaluated.

5.3 Results for Translation task 1

Tables 2 and 3 demonstrate the efficacy of a transformer based model for leaning the translation task 1 in terms of accuracy of prediction and fraction of proofs completed. The results are shown for datasets compatible with maximum proof lengths of 10 and 15. Following observations can be made from these evaluations:

1. **Increase in accuracy with increase in embedding size:** Generally, increasing the embedding size increases the prediction accuracy of the model. However, it does saturate after a certain point. Additionallys, for embedding size 512, since the data size becomes significantly large, I suspect that the current model capacity is not sufficient to learn the information represented in the data.
2. **Translation task larger proof lengths is harder:** As shown in Table 1, the dataset corresponding to the proofs of length upto 15 is 5 times larger than that the one corresponding to proofs of length 10. Additionally the number of tokens (the number of classes in the classification task) also increase to approximately 5 times. Thus, the translation task with proof lengths upto 15 is significantly difficult as compared to the translation task for proof lengths upto 10. A reduced prediction accuracy for the data corresponding 15 proof length aligns well with this observation.
3. **Number of proofs completed:** As expected, the number of proofs complemented is positively correlated with the accuracy of prediction of the model.

Embedding size Proof Length	64	128	256	512
10	83.88	85.29	85.29	85.29
15	58.16	66.03	73.11	54.43

Table 2: Accuracy of Translation Task - 1

Embedding size Proof Length	64	128	256	512
10	86.05	86.55	86.55	40.65
15	18.50	23.21	30.57	17.06

Table 3: Percentage of proofs completed for translation task -1

5.4 Results for Translation task 2

Tables 4 and 5 demonstrate the results for translation task 2. It is important to note that this task not only provides additional information about the goal conjecture to the dataset instance but also increases the stack length. Thus, there is a trade-off between enabling the model to make an informed decision and increase in complexity of the translation problem. This leads to some interesting observations:

1. **Improvement for proof length upto 10:** For poof lengths upto 10, the benefit of information provided by the addition of goal conjecture overpowers the cost incurred in terms of increased stack length. Thus, an improvement in prediction accuracy can be seen as compared to translation task 1.
2. **Worse results for proof length 15** For proof lengths upto 15, the maximum stack length increase form 13 to 14 and the results shown in table 4 demonstrate that this cost is higher than benefit provided by translation task 2. Thus, a reduced accuracy could be observed for translation task 2. The models used in these evaluations are same as the corresponding models used for Table 2. Thus, increasing the model capacity by adding additional layer may improve the prediction accuracy.

5.5 Gradient norm graphs for three seeds

Figures 3 and 4 depict the variation of ℓ_2-norm of gradient during training the models for three different initializations for translation task 1 and translation task 2, respectively. It can be observed that although initially the value of gradient do increase, eventually a drop in gradient values can be observed which signifies convergence of the model.

6 Code

The project is hosted at DATP Git repository.

Mathematical proof generation is a challenging problem. This work percieves this problem as a translation task from one proof state to another starting form the conjecture to be proved and aims to generate mathematical proofs without taking into account the inference rules. For this objective, two slightly different variations of the translation task have been proposed. Preliminary findings suggest promising outcomes for proof generation, particularly for the task incorporating goal information

Embedding size Proof Length	64	128	256	512
10	95.80	95.80	95.80	95.80
15	18.19	26.40	36.94	50.51

Table 4: Accuracy of Translation Task - 2

Embedding size Proof Length	64	128	256	512
10	99.90	99.90	99.90	99.90
15	23.83	32.81	46.41	17.15

Table 5: Percentage of proofs completed for translation task -2

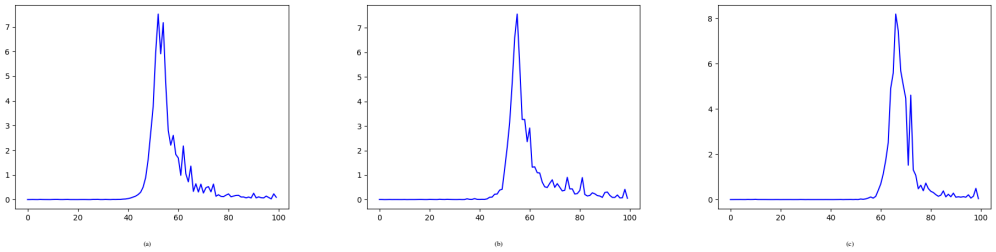

Figure 3: Grad norms for three different seeds for proof length 10 for translation task 1

in the data instances. However, it's worth noting that this study limits its scope to proof lengths of up to 15, representing only a fraction of the available data. Therefore, a comprehensive analysis encompassing the entire dataset is required to fully assess the validity of the hypothesis.

References

[1] Embedding. `https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html#embedding`. Accessed: 2024-04-26.

[2] Language translation with nn.transformer and torchtext. `https://pytorch.org/tutorials/beginner/translation_transformer.html`. Accessed: 2024-04-26.

[3] Ibrahim Abdelaziz, Maxwell Crouse, Bassem Makni, Vernon Austel, Cristina Cornelio, Shajith Ikbal, Pavan Kapanipathi, Ndivhuwo Makondo, Kavitha Srinivas, Michael Witbrock, et al. Learning to guide a saturation-based theorem prover. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[4] Norman Megill. *Metamath*, pages 88–95. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-32888-9. doi: 10.1007/11542384_13. URL `https://doi.org/10.1007/11542384_13`.

[5] Stephan Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2,3):111–126, aug 2002. ISSN 0921-7126.

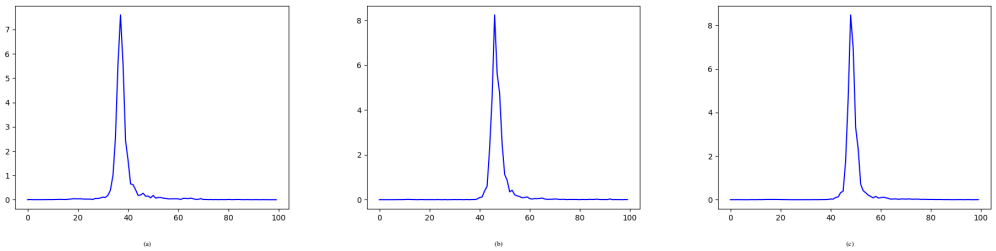

Figure 4: Grad norms for three different seeds for proof length 10 for translation task 2