Computer-Assisted Propositional Proof Systems with LSTM Text Generators

Omar Afifi
University of Michigan, LSA Statistics.
Ann Arbor, United States
afifio@umich.edu

***Abstract*—We consider the task of generating proofs in propositional systems using Long-Short-Term Memory Recurrent Neural Networks. We discuss our data selection and model design methodology, and we explore how the performance might improve with more data and larger models.**

I. Introduction

A. Overview and Motivation

Since the advent of transformers, there has been no shortage of progress in the area of generative language models. Current state-of-the-art models are impressive at producing realistic text in a variety of contexts. However, even state-of-the-art models like ChatGPT lack the ability to effectively 'think' through more advanced mathematical proofs. In this project, we further work in this area by using LSTMs to prove theorems in propositional logic. We propose a novel tokenization procedure that allows our model to generate realistic proofs and achieve impressive predictive accuracy, even with very little data.

B. Propositional Logic and Encoding Mathematics

Many statements in mathematics can be encoded as finite logical sentences, e.g:

$$\forall xy \exists z \forall t (t \in z \leftrightarrow t = x \vee t = y)$$

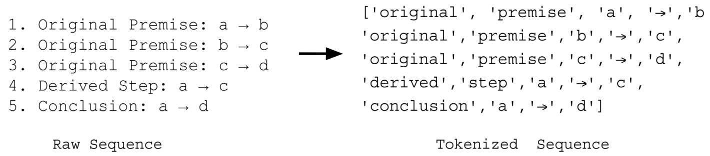
In English, this means: "For every x and y, there is a z such that, for every t, t is an element of z if and only if t is x or t is y". I.e. this sentence defines the union of the sets $\{x\}$ and $\{y\}$. Finite sentences with variables and quantifiers are considered first-order sentences, but for simplicity, we will restrict ourselves to propositional formulas, which are sentences without variables or quantifiers, e.g. $\neg(p \vee q) \leftrightarrow (\neg p \wedge \neg q)$. (This is DeMorgan's law). Given a collection of propositional sentences, one can try to derive a proof of some other sentence using deductive rules: Consider the following very simple proof:

Premise 1: $p \rightarrow q$
Premise 2: p
Conclusion: q

Humans attempting to derive q will realize that, if p implies q, and p is true, then q must also be true. However, text generators choose q based on conditional likelihood rather than genuine awareness of the 'truth value' of the sentence. If models can be demonstrated to instead learn how the symbols and sentences interact via deductive rules, then this can be a gateway to much broader accomplishments in computer-assisted proof and mathematics, since many problems are first-order expressible. (Although it is important to note that many theorems can be proven to not be first-order expressible.)

C. Prior Work & Recent Developments

Automated theorem proving has a long history, starting with Hilbert's program and eventually the first successful computer-assisted proof of a major theorem (the four-coloring theorem) at UTA in the 1980s. Traditionally, these methods used more deterministic models, rather than stochastic ones: Primarily, they tried to explore proof trees using traditional pruning algorithms like backtracking. Currently, the most impressive research combines language models with a tree-pruning approach: Lamont et al. made progress exploring a tree-pruning approach in conjunction with GPT-type models to try and simplify the decision space the models have to learn on. [1] Other approaches simply try to wield the full power of transformers, such as in recent work from Polu and Sutskever, in which they trained transformers on MetaMath data [2]. Both deterministic and stochastic approaches have their benefits: deterministic approaches are guaranteed to be correct but pose a lot of complexity difficulties because proof trees grow incredibly fast. Transformers are less of a computational burden but are inherently less reliable and notoriously data-hungry.



```
1. Original Premise: a → b
2. Original Premise: b → c
3. Original Premise: c → d
4. Derived Step: a → c
5. Conclusion: a → d
```

```
['original', 'premise', 'a', '→','b
'original','premise','b','→','c',
'original','premise','c','→','d',
'derived','step','a','→','c',
'conclusion','a','→','d']
```

Fig. 1. Text is parsed into its logical syntactic units. After Tokenization, the data is passed into an embedding layer before being fed into the LSTM.

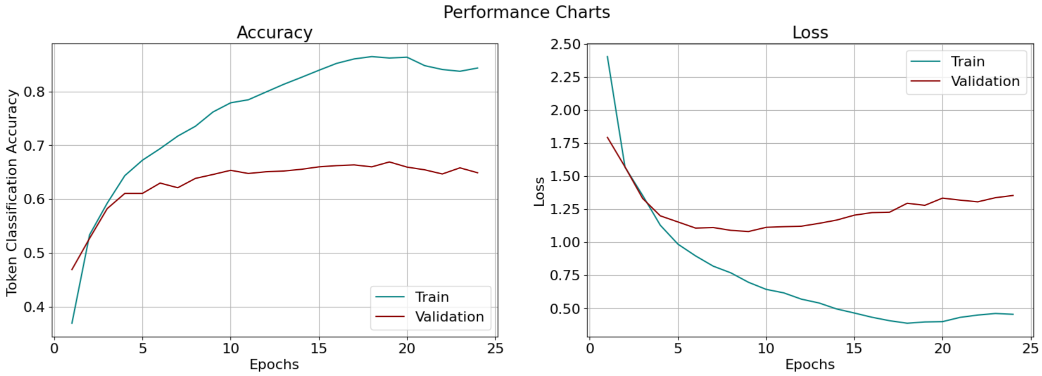


Fig. 2. The model converges after 25 epochs: train and validation loss and accuracy charts.

II. Method

To train a theorem-prover, we utilized a dataset of 1600 propositional proofs of the form observed in figure 1, sourced from Hugging Face [3]. Each sample is a complete proof consisting of premises, deductive steps, and a conclusion. The data was cleaned for duplicates in order to ensure accurate validation metrics, and each proof sequence was broken down into multiple sequences, where sequence s_{i+1} contains s_i plus the next token in the parent sequence. After this, there were around 10,000 sequences in total. The input sequences were tokenized by parsing them into their logical syntactical units: E.g. "→" and "premise" are both tokens (See fig. 1). This differs from conventional character or word-level tokenizations, but we believe that it allows the model to focus on the actual symbolic relationships, which is what we want.

We selected a Long-Short-Term Memory RNN as our model for two primary reasons: The first is that, with 1600 samples, we did not have enough data to train a transformer. Secondly, the separate long and short term gates allow the model to focus on current derivation steps while still maintaining long-term awareness, making it preferable to a vanilla RNN for this task. The sequences we train on are also very long (sometimes 100+ time steps), so LSTMs help mitigate the stability issues caused by vanishing gradients in vanilla RNNs.

It is computationally challenging to select hyper-parameters via grid-search for deep-learning, so we selected them by experimentation and careful research; The most interesting to select was the hidden dimension: Initially, the model had 8 hidden dimensions, but could not capture the nuance in the data, causing it to keep repeating the same sequences when asked to generate proofs. After struggling to resolve this, we increased the hidden dimension to 120, after which the model began to generate coherent output. (See fig. 3.) This also boosted validation accuracy by upwards of 15%. We also found that increasing the learning rate to .01 and clipping the gradients helped the model avoid premature convergence and improved the stability of the parameters. These adjustments also helped mitigate the over-fitting which is a natural concern when large models are trained on small datasets. We found that keeping the number of LSTM units small was more effective in improving validation accuracy, while dropout layers, although effective against over-fitting, were too aggressive and drastically reduced performance during inference.

Proof when $h = 8$	Proof when $h = 120$
Original Premise: $a \to b$ Original Premise: $a \to b$ Derived Step: $b \to b$ Derived Step: $b \to b$ Derived Step: $b \to q$ Derived Step: q Derived Step: o Derived Step: o Derived Step: o Derived Step: o Derived Step: o Derived Step: o ⋮	Original Premise: $a \to b$ Original Premise: $a \to c$ A.D.J.E: b Derived Step d Derived Step $b \vee c$ Conclusion: $c \wedge d$

Fig. 3. The model begins generating intelligible output after increasing the hidden dimension.

III. Results

Because of the small amount of data, the model converges after only around 20 epochs, achieving a training and validation accuracies at token prediction of around 85% and 65%, resp. (See fig. 2.) Figure 2 seems to suggest that less epochs may be ideal, but training for more does not seem to hurt the validation accuracy and tended to produce more stable outcomes in our experiments. In order to assess the model's practical utility, we also built an additional test set consisting of 10 sequences of premises that we uploaded to Hugging Face. [4] Once the training was complete, the model was asked to complete these derivations. In some instances it was able to correctly complete the proofs, such as a simple application of Modus Ponens (See figure 4.).

```
Original premise: a→b
Original premise: a
Derived step: b
Conclusion: b
```

Fig. 4. The model correctly applies Modus Ponens to complete a derivation. (Black Text: input, Red Text: output

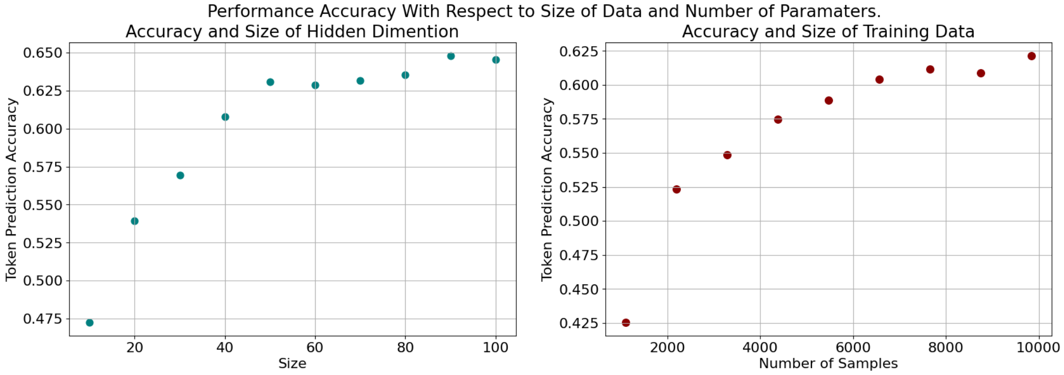


Fig. 5. **Right**: Improvement in accuracy when fixing the data set and increasing the hidden dimension. **Left**: Improvement in accuracy when fixing the model's size and increasing the amount of training data.

The model's success at this task suggests that deep language models are actually capable of learning deductive logic, although further work will need to be done to ensure that these outcomes can be obtained consistently and without dependence on training or sample variability, since our model did not always produce correct outputs across multiple training instances.

IV. Conclusion

Doing deep-learning with 1600 observations is incredibly challenging. As such, these results are incredibly promising due to both the accuracy of the model at token prediction at its ability to generate intelligible derivations. Our success also demonstrates that the right tokenization procedure can help produce the desired learning outcomes in the model, and that language models are capable of learning deductive reasoning. Nonetheless, there is room for improvement: Figure 5 shows how the model's validation accuracy improves with respect to both the training data size and the hidden size, while holding the other constant. There is an upward trend that slows when either factor is held constant. Of course, we cannot extrapolate and say that there are no performance bottlenecks that can't be conquered with just more data and larger model. However, based on these results and current research in the field, it seems reasonable that improvements can be made by scaling up these factors simultaneously. Consequently, building better, larger datasets and more complex models is the most pressing area of further research. Another possible avenue is to extended this work to first-order systems, which will require more data and more advanced models that combine the power of generative machine learning with formal deduction. I also believe that there is a lot of potential for a deep reinforcement-learning approach (as opposed to an NLP-focused one), since algorithms like PPO or MCTS allow for heuristics that can actively prune the decision space and simplify the guesswork the model needs to do.

References

[1] Sean Lamont and Michael Norrish. 3D-Prover: Diversity Driven Theorem Proving With Determinantal Point Processes. cs.AI, 2024. URL: https://arxiv.org/html/2410.11133v1

[2] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. CoRR, abs/2009.03393, 2020. URL https://arxiv.org/abs/2009.03393.

[3] Ergotts, Propositional Logic, URL: https://huggingface.co/datasets/ergotts/propositional-logic

[4] Omar Afifi, Propositional Proofs, URL: https://huggingface.co/datasets/afifio/PropositionalProofs

All of our code can be found on github: