Premise Selection For Automated Theorem Proving in Metamath

Sahil Gupta
Department of Electrical Engineering and Computer Science
Syracuse University
`sgupta47@syr.edu`

Abstract

Humans use interactive theorem provers like Metamath to manually prove theorems in mathematics. The fundamental step to prove any theorem in Metamath, starting from the goal statement, is to assign a label (premise selection) of an hypothesis, axiom, or an already proved theorem to the sub-goals at every step of the proof. In this work, we use neural networks to predict and automate the process of premise selection to enable automatic proof construction. We do further analysis to understand the bottlenecks of this approach and suggest methods to overcome them.

1 Introduction

As mathematical research delves into increasingly intricate territories, the conventional peer review system is becoming inadequate. To address this, a new paradigm is emerging where research papers include formal, computer-verifiable proofs, enabling the assessment of correctness to be automated and offloaded to machines [8]. Recently, deep learning has been applied in automated theorem proving focusing primarily on premise selection and proof guidance. DeepMath [7] uses CNNs and RNNs, FormulaNet [12] uses GNNs and Loos et al. [9] compare RNNs, LSTM, CNNs and WaveNets for premise selection. In this work, we explore the use Transformer networks [11] for premise selection.

1.1 Dataset

We use Metamath's set.mm [1], aka "Metamath Proof Explorer (MPE)" database to create our own dataset. MPE uses classical logic and Zermelo–Fraenkel set theory with the axiom of choice (ZFC). It has over 26,000 completed worked out proofs in its main section [2].

Data preprocessing We use metamath-py package [3] in python to parse the set.mm file. The file has a collection of rules (or theorems) that we want to learn from and prove. An example of the rule is 'ax-mp'. In this work, we keep the scope limited to the provable assertions (represented by $p in Metamath) to create our dataset. Since all the provable assertions in set.mm are already proved, every rule has an associated proof tree that contains information about the complete proof which is constructed in proving that rule. We iterate over the proof tree and extract data of the form <xi, yi> where xi is a sequence of tokens that represents the conclusion of the proof step and yi is the corresponding label of the consequent of the rule that is used to justify this proof step. We get a total of 2.9M such input output pairs in our dataset. One example of such a pair is [('|-', '(', 'ph', '->', '-.', '-.', 'ph', ')'), 'notnot']. We shuffle and split our dataset into 80% train and 20% test sets. We plot the distribution of top 10 labels by count in Figure 1.

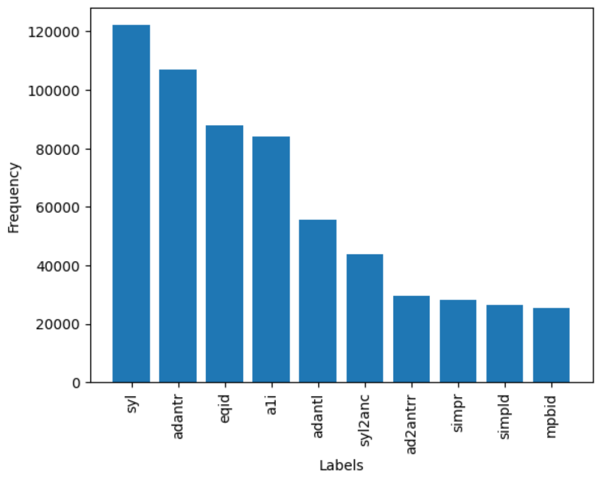


Figure 1: Frequency of the top 10 labels in the dataset.

The codebase and dataset referenced and utilized in the research presented in this paper are publicly accessible and can be obtained from the following source: `https://github.com/sahil7gupta/atp-metamath-project`

1.2 Our approach

We formulate the problem as a supervised learning problem. For a given datapoint in our dataset, we have xi which is a sequence of tokens in the proof step, and yi is the correct label that we are trying to predict. For this purpose we use transformers [11] to learn the relation between tokens, and we use multi-class classification to predict the correct label. The scope of this work is limited to premise selection. To automate a complete proof of a theorem however, we would need to additionally implement beam search which is discussed briefly in the future work. We discuss our approach in detail in the next section.

2 Methodology

Our neural network architecture is composed of 3 components: Input embedding layer, a series of transformer encoder blocks, and a classification head layer. The architecture is shown in Figure 2a

Input embedding layer Each token in xi is initialised with a random learnable embedding. These embeddings are added to sinusoidal positional embedding as defined by Vaswani et al. in [11]. The resulting embeddings are passed to the Transformer Encoder Block.

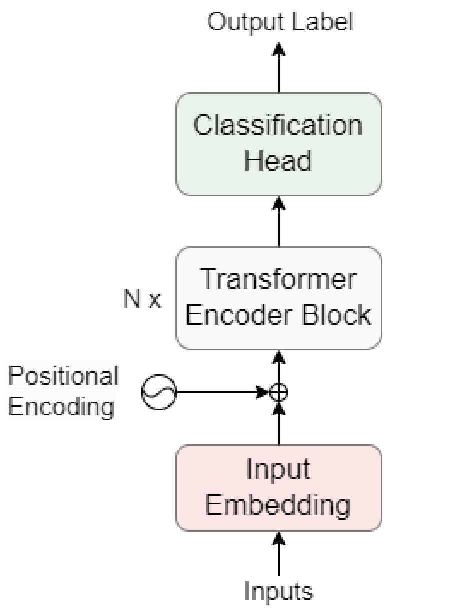
Transformer Encoder Blocks We use a series of encoder blocks as defined by Vaswani et al. in [11], to compute the attention between the tokens. Figure 2b shows the network layers in one such encoder block. The output of the previous encoder block directly are fed into the input of the next block. The output of the last block is passed to the classification head.

Classification Head The classifier head is a Pytorch linear layer that maps the output of the encoder block to the output class labels.

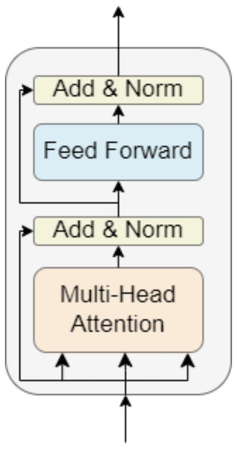
Loss function We use a pytorch's cross entropy loss [4] to train the network. It computes the cross entropy loss between output logits of the classification head and yi.

$$\ell(x,y) = L = \{l_1, \dots, l_N\}^\top, l_n = -\sum_{c=1}^{C} \log \frac{\exp(x_{n,c})}{\sum_{i=1}^{C} \exp(x_{n,i})} y_{n,c}$$

where x is the output of the classification head, y is the target, C is the number of classes, and N spans the minibatch dimension



(a) 3 components: Input embedding layer, a series of transformer encoder blocks, and a classification head layer.



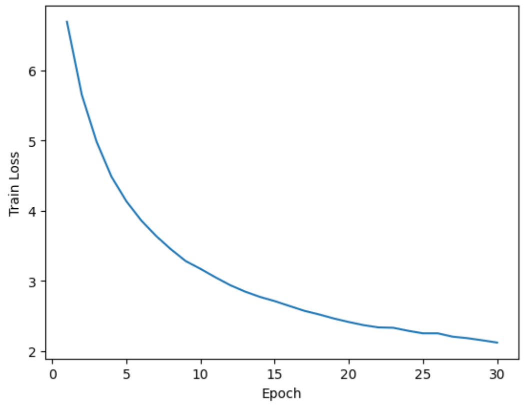
(b) Transformer Encoder Block

Figure 2: Model architecture diagram

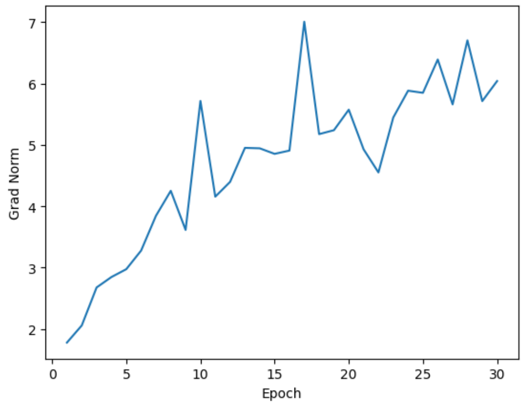
Optimiser We use Adam optimiser with a learning rate of 0.001 for training.

3 Results

Hyperparameters We set the following values of the hyperparameters: batch_size = 32, block_size = 200 (the context length) , d_model = embeds_size = 40, epochs = 30, num_heads = 8. We used Optuna [5] on various hyperparameters to discover the optimal hyperparameter values. In particular we tried, batch_size = [32, 256, 32] (*[low, high, step]*), num_heads = [1, 50, 3], head_size (embedding size for a single attention head) = [10, 100, 10], number of encoder blocks = [1, 20], optimizers = ["Adam", "RMSprop", "SGD"], and learning rate = [1e-5, 1e-1]. But we did not see any significant increase in accuracy due to Optuna hyperparameter search compared to our default setting.

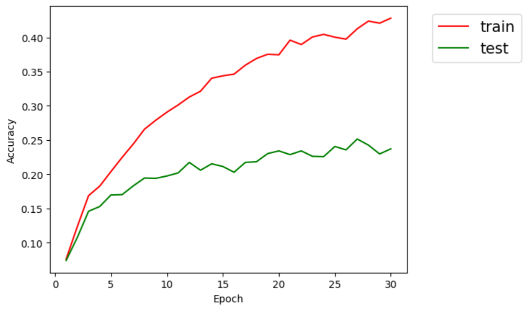


(a) Training Loss

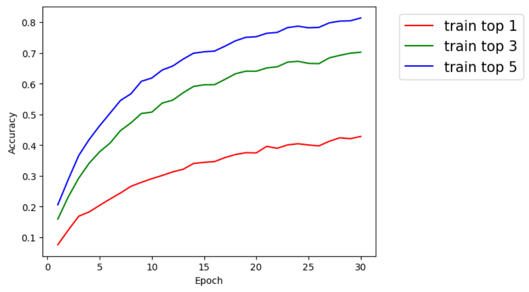


(b) Gradient Norm

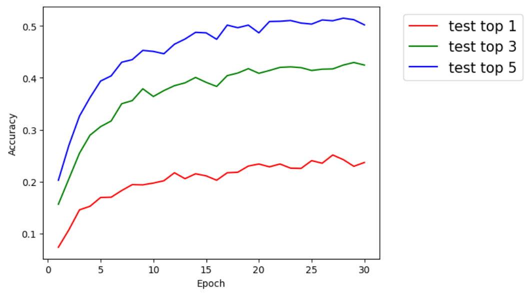
Figure 3: Plot of (a) Training Loss (b) Gradient Norm with training iterations



(a) Test and Training Set Accuracy - Top 1 Label



(b) Training Set Accuracy



(c) Test Set Accuracy

Figure 4: Experiment Repetition 1 - Accuracy

Due to hardware constraints, out of 2.9M input-output pairs, we select a subset of the first 10K input-output pairs to perform our experiments. We use Pytorch [10] for training our model. The results of the first run are shown in Figure 3 and Figure 4.

We see in Figure 3a that the training loss decreases with increasing epochs. This indicates that the model is indeed converging to a local optimum value. In Figure 3b, the gradient norm is increasing with the training iterations. This could be inferred in multiple ways. It could be caused by either an exploding gradient problem or an increase in the parameter values starting from small random values. We suspect that it is the latter but would need to conduct further evaluations to confirm this hypothesis.

For the baseline results of the 10K dataset, we calculate the frequency of Top 1 label: 4.24%, Top 3 labels: 7.62%, Top 5 labels: 10.13%. In Figure 4a, we plot the training vs test accuracy (Top 1 label). The test accuracy stops increasing by iteraton 30 and converges to 22.4% which is significantly better than the baseline. The training accuracy reaches to 43.5% and can be seen to increase further with more iterations. We also plot the Top 3, and Top 5 label accuracy for the training 4b and the test sets 4c. We see that the training accuracy reaches >80% for the Top 5 labels, and the test accuracy reaches >50% which is significantly better than the baseline of 10.13%.

For reproduciblility, we perform 2 additional experiments, the results are shown in Appendix Figure 6, and Figure 7.

To compare the performance on increasing the dataset size, we also perform experiments on 3 different subsets of the dataset with varying sizes. We choose the first 10K, first 100K and the complete 2.9M input-output pairs and report the Top 5 accuracy for these datasets. The results are shown in Figure 5. We see a trend of increase in test accuracy with the increase in dataset size. Specifically, for 10x the dataset size, we see a 10% increase in accuracy. Using this as basis, we hypothesize that even when using the complete dataset, the model is capable of learning more with an increase in data and discuss methods to scale this approach in the next section.

4 Conclusion and Future Work

1. Proofs in Metamath contains on an average of 71 proof steps. Considering the best (top 5) accuracy of our model, i.e. 73%, we can estimate the complete proof accuracy to

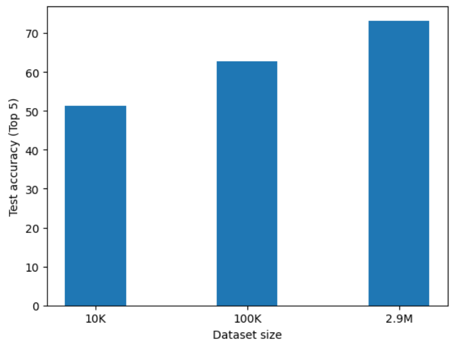


Figure 5: Test accuracy (Top 5 labels) increases on increasing dataset on the same model size

be $1.97\text{x}10^{-10}$ when implemented in conjunction with a beam search. This is quite less compared to the state-or-the-art approaches.

2. We see a correlation between increase in dataset size with increase in accuracy for the same model size. So we could use data augmentation approaches to generate more data and that could lead to improving accuracy.
3. There is a lot of geometry in the data that has not been explored. For example: if we interchange the token 'ph' with 'ps' in set.mm, all the proofs will still be valid. We could use geometric deep learning [6] approaches to learn these symmetries in our model without generating more data.
4. Our model treats tokens like '->', '(', ')' representing mathematical symbols and tokens like 'ph', 'ps' representing variables with no distinction. But humans have prior knowledge about their meanings and thus treat them in a different way when manually proving theorems. We could also encode this information as a prior in the model itself.

References

[1] `https://github.com/metamath/set.mm/`.

[2] `https://us.metamath.org/mpeuni/mmset.html/`.

[3] `https://github.com/garrettkatz/mmpy/`.

[4] `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`.

[5] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

[7] Geoffrey Irving, Christian Szegedy, Alexander A Alemi, Niklas Een, Francois Chollet, and Josef Urban. Deepmath - deep sequence models for premise selection. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[8] Ruy Queiroz Lev D. Beklemishev. *Logic, Language, Information, and Computation*. Springer Berlin, Heidelberg, Philadelphia, PA, USA, 2011.

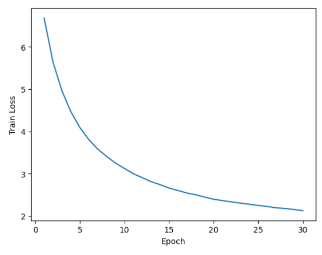
[9] Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search, 2017.

[10] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

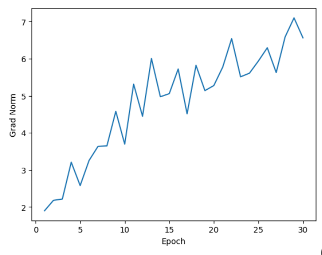
[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[12] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

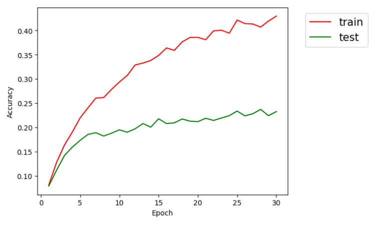
5 Appendix



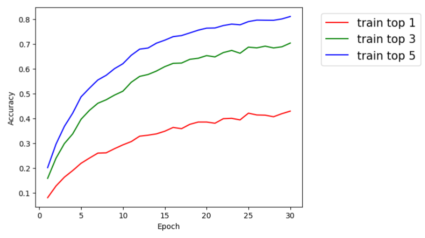
(a) Training Loss



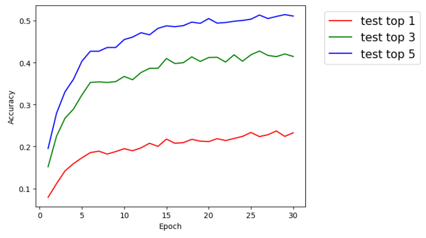
(b) Gradient Norm



(c) Test and Training Set Accuracy - Top 1 Label

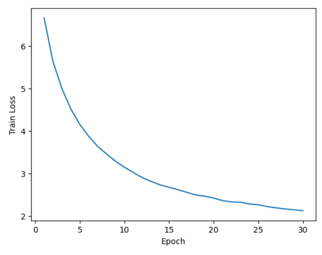


(d) Training Set Accuracy

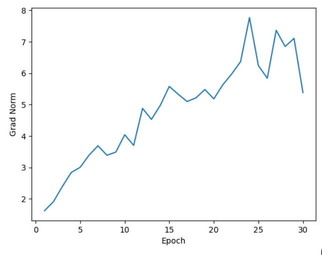


(e) Test Set Accuracy

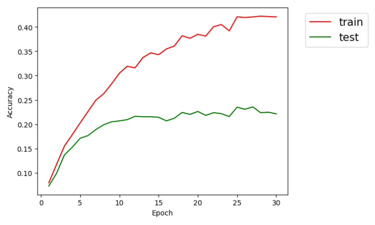
Figure 6: Experiment Repetition 2



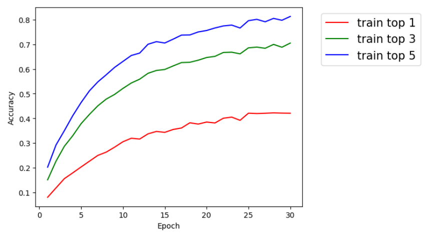
(a) Training Loss



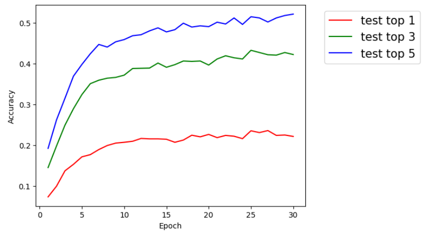
(b) Gradient Norm



(c) Test and Training Set Accuracy - Top 1 Label



(d) Training Set Accuracy



(e) Test Set Accuracy

Figure 7: Experiment Repetition 3