PREDICTING ONE-STEP PROOFS IN FORMAL MATHEMATICS WITH TRANSFORMER-BASED NEURAL NETWORKS

Xiao Wang
Department of Computer Science
Syracuse University
xwang99@syr.edu
Project Code: `https://github.com/WangXiaoShawn/-Deep-Automated-Theorem-Proving`

1 INTRODUCTION

1.1 MOTIVATION

In the era of digital transformation, artificial intelligence (AI) has emerged as a catalyst for innovation and growth. One of the most notable AI models that has attracted considerable attention is the transformer model, which serves as the basis for ChatGPT, a large language model developed by OpenAI. Sanmarchi et al. (2023).

This paper focuses on exploring the potential of transformer models in the Automated Theorem Proving (ATP) domain, specifically for the first step proof of first-order logic, as it is crucial in establishing the groundwork for all subsequent reasoning and deductions.

My research suggests proof verification can be treated as a sequence-to-sequence (seq2seq) problem, given the inherent logical chain relationships in mathematical proofs. Whalen (2016).

This article aims to investigate the potential of transformer models in mathematical proof , address the challenges associated with utilizing transformer models in the field of mathematics, and provide insights into the methodology and techniques employed in developing the transformer model for Set.mm. Additionally, it offers a glimpse into the future of AI in mathematical research.

1.2 METAMATH DATA SET: SET.MM

Metamath is a computer language and accompanying software designed for archiving, verifying, and examining mathematical proofs.

Distinct from some other systems, Metamath does not integrate a specific set of axioms into its framework. Instead, the language is characterized by its simplicity and robustness, with minimal hard-wired syntax. Within Metamath, users express axioms, theorems, and proofs in a database consisting of text files. Each proof step must be validated using an axiom or a previously proven theorem to establish a theorem, ensuring complete transparency.

The databases Set.mm generated in this manner contain human-readable axioms and theorem statements. Proofs in this repository are compressed to conserve space, but tools are available to easily decompress them, yielding a human-readable sequence of every proof step. Megill (2019)

1.3 NEURAL NETWORKS: TRANSFORMER-BASED NEURAL NETWORKS

Transformer neural networks for sequence-to-sequence (seq2seq) tasks are powerful models designed to handle a variety of problems, such as machine translation, summarization, and question-answering. These models are based on the transformer architecture, which employs self-attention mechanisms to process input sequences in parallel rather than sequentially, as seen in traditional recurrent neural networks (RNNs). This parallel processing enables faster training and better handling of long-range dependencies within the input data. Beltagy et al. (2020)

In exploring set.mm, a Transformer model is composed of an encoder and a decoder. The encoder handles the input sequence, which signifies an unproven theorem, whereas the decoder produces the output sequence, representing the initial step of the theorem's proof. Both parts employ self-attention and positional encoding to efficiently capture the relationships among elements within the sequences.

2 METHODS

2.1 TRANSFORMER ARCHITECTURE

My models are built based on the model developed by Katz (2023), which only has one encoder in the model. In the first design (Figure 1), the model added an decoder, and sef-attention layers make the model meet the design of Vaswani et al. (2017).

Positional encoding is a key component in Transformer design Vaswani et al. (2017), and I find the Trigonometric positional encoding Vaswani et al. (2017) and Relative position encoding Shaw et al. (2018) are two different approaches to encoding positional information in a sequence. I want to explore if the model could benefit from these approaches and achieve better performance when I combine these two positional encoding approaches together. Therefore, I derive the positional encoder by combining the Trigonometric Positional Encoder and Relative Positional Encoder and put it into my models.

Inspired by Jonas Gehring's work Gehring et al. (2017) on improving the performance of LSTM tasks using a convolutional architecture, I want to explore if the convolution layer would boost the performance of my model. Hence, I decided to integrate the convolution layer into the Second Transformer architecture (Figure 2) to investigate its impact on the model's performance and compare with the first baseline model.

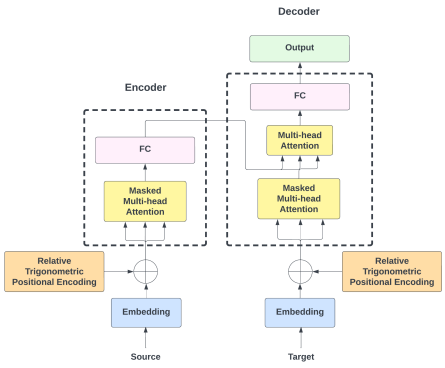

Figure 1: The First Architecture

Figure 2: The Second Architecture

In the upcoming section, I will present the equations used in my architecture to provide a deeper understanding of how the model functions.

2.1.1 Equations defining the architecture

Embeddings

Similar to other models that transform sequences, I employ learned embeddings to convert the input tokens and output tokens into d_{model} vectors.

$$E(x) = W \cdot x \tag{1}$$

where $E(x)$ is the embedding vector for input token(s) x, W is the embedding matrix of shape $(vocab\_size, d\_model)$, where $vocab\_size$ is the size of the vocabulary and d_model is the size of the embedding vector, x is the index (indices) of the token(s) in the vocabulary.

Convolution Layer

$$Conv(x) = ReLU(x * W_{\text{conv}} + b_{\text{conv}}) \tag{2}$$

In the convolution layer, the input vector x is convolved with a set of learnable filters, represented by the parameters W_{conv} and b_{conv}. The output is then passed through a ReLU active function, applied element-wise.O'Shea & Nash (2015)

Position-wise Feed-Forward Networks

The position-wise feed-forward network Vaswani et al. (2017) takes as input a sequence of embeddings, and applies a fully connected feed-forward network to each position independently. The feed-forward network consists of two linear transformations with a ReLU activation in between:

$$FFN(x) = max(0, W_1 \cdot x + b_1) \cdot W_2 + b_2 \tag{3}$$

where W_1, W_2, b_1, and b_2 are learnable parameters.

Fully Connected layer

$$y = W \cdot x + b \tag{4}$$

where W is the weight matrix, x is the input vector, b is the bias vector, and y is the output vector.

Relative Trigonometric Positional Encoder

The Relative Trigonometric Positional Encoder in my design adds the relative position embeddings to the input sequence after applying the Trigonometric Positional Encoder. And it is built based on Vaswani et al. (2017).

The formula for combining the trigonometric positional encodings and relative position encoding is given by:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{5}$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{6}$$

where pos is the position of the token in the sequence, i is the dimension index, and d_{model} is the dimensionality of the model.

The formula for computing the relative position encoding is given by:

$$RE_{(i,j,2k)} = sin\left(\frac{i-j}{10000^{\frac{2k}{d_{model}}}}\right) \tag{7}$$

$$RE_{(i,j,2k+1)} = cos\left(\frac{i-j}{10000^{\frac{2k}{d_{model}}}}\right) \tag{8}$$

where $RE_{i,j,2k}$ and $RE_{i,j,2k+1}$ are the $2k$-th and $(2k+1)$-th elements of the relative positional encoding for the pair of positions (i, j), d_{model} is the dimension of the model, and k ranges from 0 to $\lfloor (d_{\text{model}}/2) - 1 \rfloor$.

Formula for computing the combined positional encodings:

$$PE^{combined}_{pos,i} = PE_{pos,i} + RE_{i,j,k} \tag{9}$$

where $PEpos, i$ is the i-th element of the trigonometric positional encoding for position pos, REi, j, k is the k-th element of the relative positional encoding for the pair of positions (i, j), and k is chosen such that $i - j = k$.

By combining these two types of positional encodings, I can capture both the absolute position and the relative position information of the tokens in the sequence.

Multi-Head Attention Vaswani et al. (2017)

Query, Key, and Value projections:

$$Q = W_q \cdot X,\ K = W_k \cdot X,\ V = W_v \cdot X \tag{10}$$

Where W_q, W_k , and W_v are learnable matrices that transform each input into a query, key, and value. And X is input matrix.

Scaled Dot-Product Attention:

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V \tag{11}$$

Multi-Head Attention:

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h) \cdot W^O \tag{12}$$

where each $head_i$ is computed as $head_i = Attention(Q_i, K_i, V_i)$. The Q, K, and V in the $Attention$ equation are the query, key, and value matrices, respectively. W^O is the final output projection matrix. The $MultiHead$ operation first computes h attention heads, each of which computes the attention score using the $Attention$ function. The resulting attention scores are concatenated and passed through a linear projection given by W^O to produce the final output.

Multi-Head Attention for decoder Vaswani et al. (2017)

First, apply the positional encoding to the embedded target sequence *X*:

$$Q = RelativeTrigonometricPositionalEncoder(X) \tag{13}$$

where Q is the query matrix with positional information added

Then generate the Masked Multi-Head Attention using the query matrix Q, memory vector M, and the attention function $Attention$:

$$Q, \_, \_ = Attention(Q, M, M) \tag{14}$$

In this equation, Q represents the target sequence after applying the attention mechanism. The memory vector M is the output of the encoder, and the attention function $Attention$ is an instance of the Masked Multi-Head Attention mechanism.

In summary, the equations describe the process of adding positional information to the target sequence and then applying the Masked Multi-Head Attention mechanism using the memory vector from the encoder. The output is the updated target sequence containing attended information from the source sequence.

Output Layer

$$readout = W_r \cdot output + b_r \tag{15}$$

where $readout$ is the output after applying the readout layer, W_r is the weight matrix of the readout layer, b_r is the bias vector of the readout layer, and $\cdot$ denotes matrix multiplication.

2.2 Loss Function

I choose cross-entropy as the loss function in my sequence-to-sequence model due to its advantageous properties. In seq2seq models, the decoder generates a probability distribution over the vocabulary for each position in the output sequence. The objective is to maximize the likelihood of the correct tokens at each position. Cross-entropy loss is well-suited for this purpose, as it quantifies the dissimilarity between the predicted probability distribution and the true distribution, which is represented as a one-hot vector corresponding to the correct token.Brownlee (2019)

During the training process, the cross-entropy loss is computed for each position in the output sequence and subsequently averaged over the entire sequence length. This methodology promotes the generation of accurate tokens at every position in the output sequence, leading to improved overall performance of the model. The formula for cross-entropy loss is as follows:

$$H_{(p,q)} = -\sum_i p\left(i\right) log\left(q(i)\right) \tag{16}$$

In this equation, $H(p, q)$ denotes the cross-entropy loss between two probability distributions p and q. The summation is over all possible events i. $p(i)$ represents the true probability of event i, and $q(i)$ represents the predicted probability of event i.

2.3 Optimizer

The optimizer chosen for my sequence-to-sequence model is Adaptive Moment Estimation, is a popular optimization algorithm that combines the advantages of two other well-known optimization

techniques: AdaGrad and RMSProp. It adapts the learning rate for each parameter during training, resulting in faster convergence and improved performance. Kingma & Ba (2014)

2.4 Deep Learning Library

In my paper, I would use the PyTorch library Imambi et al. (2021) in python to build my transformer models.

3 Experiment and Results

In this research, my objective was to evaluate and compare the performance of two distinct architectures to identify the superior model. To ensure a reliable and unbiased comparison, I took the following steps:

Dataset: I used the same dataset for both models, ensuring that they were exposed to an equal amount of information and faced similar challenges during training and testing. This helps eliminate variations in performance that could arise from differences in the data.

Dataset size: I maintained the same dataset size for both architectures, making sure that neither model had an advantage in terms of data availability.

Hyperparameters: I determined the optimal hyperparameters by fine-tuning the first model and then applied these parameters to the second model for comparison purposes. This approach ensures that both models are compared under the same optimal conditions, providing a fair evaluation of their performance and effectiveness.

By maintaining consistency, I created a controlled environment for the comparison of the two architectures. This approach allowed me to isolate the effects of the architectural design and accurately assess the performance of each model, ultimately determining the superior architecture.

3.1 dataset train/validate/test split

I extract 22,000 theorems from the set.mm file for my experiment. To ensure a fair distribution of complexity across the dataset, I first shuffle these theorems, as theorems proven later generally have longer proof steps. After shuffling, I divide the dataset into three parts: 18,000 theorems for the training set, 2,000 theorems for the validation set, and the remaining 2,000 theorems for the test set. This division allows me to effectively train, fine-tune, and evaluate my model on distinct subsets of theorems.

3.2 Preparing Theorem Data for Model Training

During the parsing process, I create a sequence that consists of an input prompt and a corresponding output. The input prompt is a combination of the theorem's assertion and hypothesis, which together provide the necessary context and assumptions for proving the theorem.

The output, on the other hand, represents the first step in the proof of the theorem. This first step contains essential elements for the proof, which are the assertion, hypothesis, and proof of the first theorem in the overall proof sequence. This step serves as a starting point for the proof, from which subsequent steps will be derived.

By structuring the input and output, I can train the model to learn the relationship between the given theorem (with its assertion and hypothesis) and the theorem for the first step in its proof.

3.3 training procedure

First, the device and model hyperparameters, learning rate schedule, optimizer, and loss function are set up. The number of epochs, updates, and other training-related variables are also set.

For each epoch, the training data is shuffled and iterated through. For each training example, an input and output sequence is created. If the conjecture is longer than the maximum length, the example is skipped.

And the gradients are zeroed, and the input sequence is passed through the model. The loss is computed based on cross-entropy in the output region. Backpropagation is performed, and the model parameters are updated.

Moreover, the Noam learning rate schedule is applied, and the one-step-ahead prediction accuracy in the output region is calculated periodically.

At the end of each epoch, the model is evaluated on the validation set and test set. If the conjecture is longer than the maximum length, the example is skipped. The one-step-ahead prediction accuracy is calculated on both sets.

Overall, this training procedure aims to optimize the model's parameters using cross-entropy loss and the Noam learning rate schedule while monitoring its accuracy on both the training and validation sets. By adjusting the hyperparameters and observing the model's performance, I can fine-tune the model to achieve better accuracy. With the same training procedure, I will run my models (the first architecture2 and the second architecture1) three times each separately to ensure model stability.

3.4 The First Architecture Result

3.4.1 Hyperparameters

The main goal for tuning the hyperparameters of the model is to boost the model performance on my tasks, and I also want to decrease the usage of the computing resources at the same time. The hyperparameters were chosen based on their performance on the validation set, with the goal of selecting the hyperparameters that would maximize the validation accuracy. Hyperparameters that resulted in the highest accuracy were ultimately selected for the final model. These are the parameters in the first model:

$max\_len = 128$: The maximum length of the input and output sequences. Sequences longer than this value will be skipped during training and evaluation.

$d\_model = 256$: The dimensionality of the token embeddings and the Transformer model's hidden states.

$nhead = 16$: The number of attention heads used in the multi-head attention mechanism.

$nlayers = 10$: The number of layers (or blocks) in the Transformer's encoder and decoder stacks.

$batch\_size = 10$: A larger batch size reduces the noise in the gradient estimate, leading to a more stable training process. However, a larger batch size requires more memory and may slow down the training process. It's best to choose a batch size that is as large as possible without causing memory issues. Here will accelerate the GPU x10 speed.

$num\_epoch = 20$: The training loop is set up to run for 20 epochs. This means that the algorithm will perform 20 passes through the entire training dataset, updating the model parameters at each iteration. After 20 epochs, the training process will stop and the model will be evaluated on the validation and test dataset.

Noam learning rate schedule settings:

$use\_noam = True$: A flag to indicate whether to use the Noam learning rate schedule during training.

$base\_lr = 0.0005$: The learning rate controls the step size in gradient descent. A high learning rate may cause the algorithm to overshoot the minimum and diverge, while a low learning rate may cause the algorithm to converge slowly. The base learning rate here is the initial learning rate and the maximum learning rate during training while using the Noam schedule.

$warm\_up = 500$: The number of warm-up steps used in the Noam learning rate schedule.

3.4.2 experiment result

Repetition 1:

The figure (Figure 3) and accuracy metrics (Table 1) presented are derived from the experiment conducted, and here is the result of this repetition:

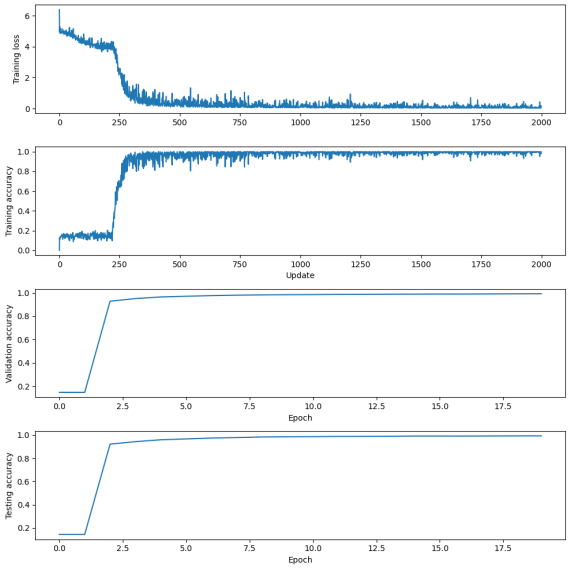

Figure 3: Baseline model exp1

Table 1: repetition 1 overall performance metric

Part	Accuracy
Average training accuracy	87.81%
Average validation accuracy	89.68%
Average testing accuracy	89.49%

Repetition 2:

The figure (Figure 4) and accuracy metrics (Table 2) presented are derived from the experiment conducted, and here is the result of this repetition:

Table 2: repetition 2 overall performance metric

Part	Accuracy
Average training accuracy	87.88%
Average validation accuracy	90.36%
Average testing accuracy	90.22%

Repetition 3:

The figure (Figure 5) and accuracy metrics (Table 3) presented are derived from the experiment conducted, and here is the result of this repetition:

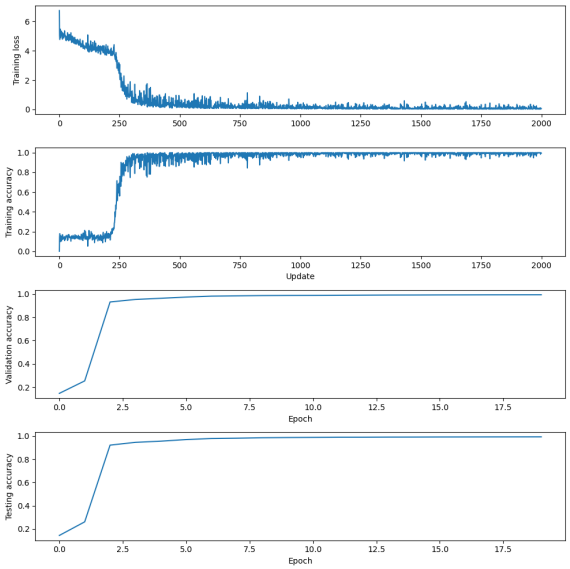

Figure 4: Baseline model exp2

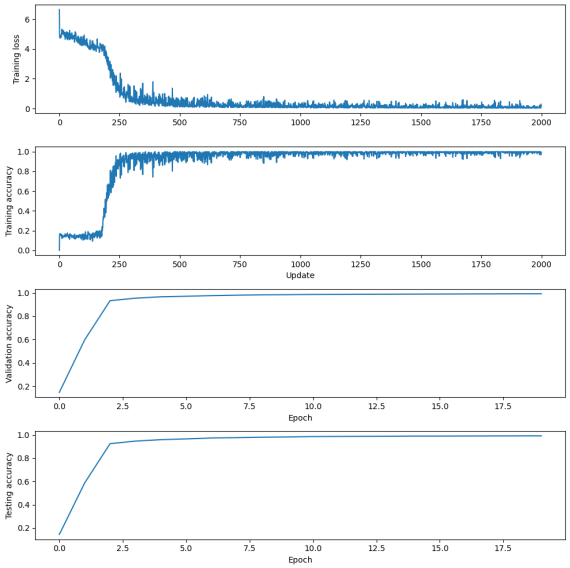

Figure 5: Baseline model exp3

Table 3: repetition 3 overall performance metric

Part	**Accuracy**
Average training accuracy	89.32%
Average validation accuracy	91.97%
Average testing accuracy	91.67%

3.5 THE FIRST ARCHITECTURE RESULT

3.5.1 ADDITIONAL LAYERS

For the convolution layer, the kernel size is set to 3 and the padding is set to 1, which means that the convolution operation will have a receptive field of size 3 and will output a vector of the same size as the input. After that, the output will be passed to the normalization layer for further use.

3.5.2 HYPERPARAMETERS

To perform a controlled experiment, I kept other Hyperparameters in the second model the same as the first model. And here are the Hyperparameters for the second model:

$max\_len = 128$: The maximum length of the input and output sequences. Sequences longer than this value will be skipped during training and evaluation.

$d\_model = 256$: The dimensionality of the token embeddings and the Transformer model's hidden states.

$nhead = 16$: The number of attention heads used in the multi-head attention mechanism.

$nlayers = 10$: The number of layers (or blocks) in the Transformer's encoder and decoder stacks.

$batch\_size = 10$: Use GPU accelerated x10 speed.

$num\_epoch = 20$: The training loop is set up to run for 20 epochs. This means that the algorithm will perform 20 passes through the entire training dataset, updating the model parameters at each iteration. After 20 epochs, the training process will stop and the model will be evaluated on the validation and test dataset.

$max\_examples = 1000$: During each epoch, the model will be updated 1000 times using random samples from the training data. Once 1000 updates have been performed, the training loop moves on to the next epoch.

Noam learning rate schedule settings:

$use\_noam = True$: A flag to indicate whether to use the Noam learning rate schedule during training.

$base\_lr = 0.0005$: The base learning rate, which is the initial learning rate and the maximum learning rate during training when using the Noam schedule.

$warm\_up = 500$: The number of warm-up steps used in the Noam learning rate schedule. During these warm-up steps, the learning rate increases linearly from 0 to the base learning rate. After the warm-up steps, the learning rate decreases proportionally to the inverse square root of the number of updates.

3.5.3 EXPERIMENT RESULT

Repetition 1:

The figure (Figure 6) and accuracy metrics (Table 4) presented are derived from the experiment conducted, and here is the result of this repetition:

Table 4: repetition 1 overall performance metric

Part	Accuracy
Average training accuracy	71.89%
Average validation accuracy	74.30%
Average testing accuracy	73.12%

Repetition 2:

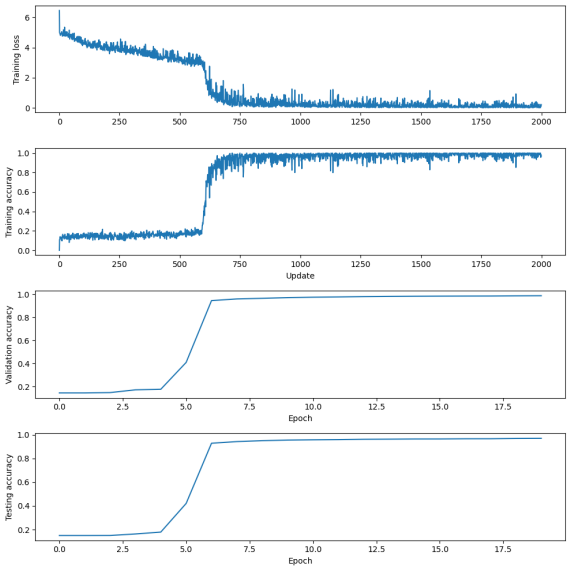

Figure 6: conv exp1

The figure (Figure 7) and accuracy metrics (Table 5) presented are derived from the experiment conducted, and here is the result of this repetition:

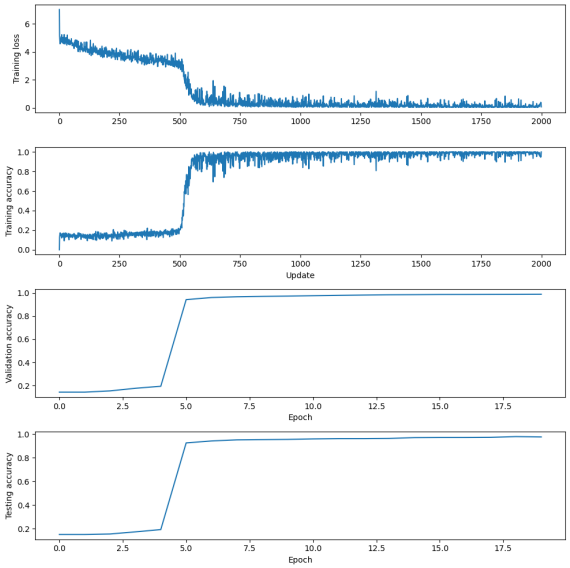

Figure 7: conv exp2

Table 5: repetition 2 overall performance metric

Part	Accuracy
Average training accuracy	75.63%
Average validation accuracy	77.36%
Average testing accuracy	76.14%

Repetition 3:

The figure (Figure 8) and accuracy metrics (Table 6) presented are derived from the experiment conducted, and here is the result of this repetition:

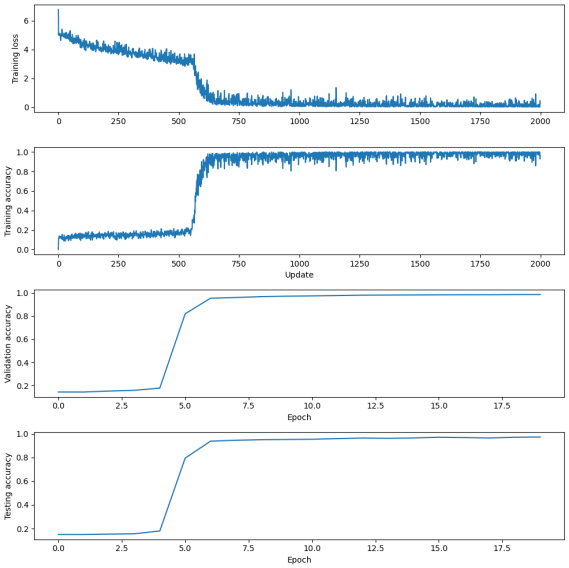

Figure 8: conv exp3

Table 6: repetition 3 overall performance metric

Part	Accuracy
Average training accuracy	73.17%
Average validation accuracy	76.40%
Average testing accuracy	75.13%

3.6 Assessing Theorem Prediction Performance

Upon completing the training phase, I evaluate the model's performance by generating predictions for the next theorem. My observations indicate that the model has effectively internalized the underlying patterns and structures present in the target theorem. 9 Although the generated theorem exhibits minor discrepancies when compared to the ground truth, an individual with domain expertise can readily discern its identity.

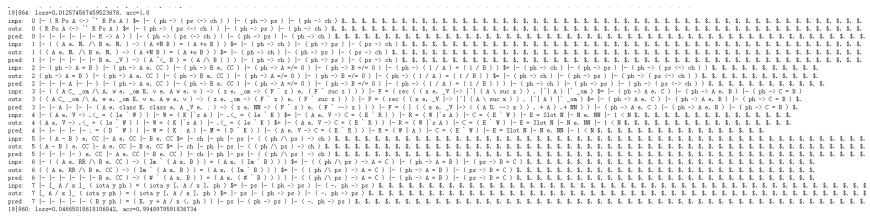

Figure 9: Theorem Prediction check 1

Moreover, I extract the token that possesses a 100% accuracy rate for the subsequent step in the prediction process, and discover that it perfectly aligns with the desired target value.9 These findings, which highlight the model's ability to accurately reproduce essential patterns and achieve precise

token-level predictions, provide strong evidence that my model has successfully met its intended goals.

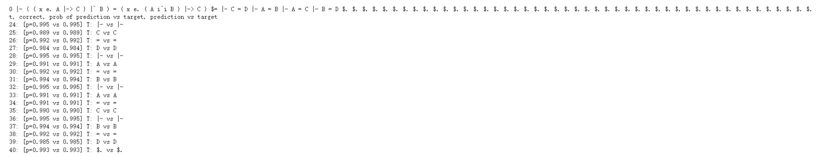

Figure 10: Theorem Prediction check 2

4 CONCLUSION

I embarked on a research project where they aimed to explore the effectiveness of the transformer model, which is considered one of the most popular deep learning architectures used for natural language processing tasks. I directly applied this model to the set.mm dataset without making any significant modifications to the data, in order to obtain a more comprehensive understanding of its performance.

I primary objective was to conduct experiments to provide proof for the Metamath. The dataset(set.mm) I used comprised only the features inherent to the theorem, such as assertion hypothesis, and did not require additional processing. By implementing the transformer model directly on the dataset, I was able to assess the performance of this model on mathematical theorems, which is a relatively unexplored area in natural language processing.

The results are quite impressive, as the first model demonstrates superior accuracy compared to the second sophisticated model. This indicates that the first architecture is more optimal, as it not only guarantees accuracy but also substantially reduces the required training resources compared to the second more complicated model.

To explain my result, I hypothesize that, in the context of the set.mmMegill (2019) dataset, which is a formalized mathematics database, every record plays a vital role in understanding and proving each theorem. Traditional use cases for convolutional layers, such as image processing or natural language processing, involve data where some spatial or local patterns can be compressed or ignored without significantly affecting the overall understanding. However, this does not hold true for the set.mmMegill (2019) dataset.

My next research endeavor will focus on improving the quality of axiom proofs by utilizing a comprehensive dataset processing technique. To achieve this, I will first implement a labeling mechanism that distinguishes between free and constrained variables within the dataset.Whalen (2016) This labeling process will allow us to identify which variables can be replaced and which must be constrained, thus optimizing the data for machine learning algorithms. By replacing free variables with substitutions, I anticipate that the model's ability to identify axioms accurately would be improved. This approach is particularly promising as it enables us to enhance the quality of the data.

Moreover, while generating sequences from my observations, I occasionally encountered challenges in accurately identifying specific theorems, despite having high precision. To address this problem, I intend to create a new model that can effectively discern the appropriate theorem associated with the output sequence. This improvement will substantially benefit my proofing process.

Undertaking this project presented us with a significant challenge, requiring me to learn mathematical proofs from the ground up, while also studying several complex machine learning models. Throughout the project, I had to integrate my understanding of mathematical concepts into deep learning, which made training models in this direction more abstract compared to other areas.

REFERENCES

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Jason Brownlee. A gentle introduction to cross-entropy for machine learning. *Machine Learning Mastery. https://machinelearningmastery. com/cross-entropy-for-machine-learning*, 2019.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pp. 1243–1252. PMLR, 2017.

Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. *Programming with TensorFlow: Solution for Edge Computing Applications*, pp. 87–104, 2021.

Garrett Katz. "transformers in pytorch", cis 700: Deep learning for automatic theorem proving, May 2023. URL `https://colab.research.google.com/drive/1hIwuwZelSsSbqgJDsHJv2-peAx9wO4AL?usp=sharing`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Norman D. Megill. *Metamath: A Computer Language for Mathematical Proofs*. Lulu Press, Morrisville, North Carolina, 2019. URL `http://us.metamath.org/downloads/metamath.pdf`.

Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

Francesco Sanmarchi, Davide Golinelli, and Andrea Bucci. A step-by-step researcher's guide to the use of an ai-based transformer in epidemiology: an exploratory analysis of chatgpt using the strobe checklist for observational studies. *medRxiv*, pp. 2023–02, 2023.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Daniel Whalen. Holophrasm: a neural automated theorem prover for higher-order logic. *arXiv preprint arXiv:1608.02644*, 2016.