Team ClosedAI: Fine-Tuning Llama3-8B for Math Question Answer Verification

Kaartikeya Panjwani (kp3291)
New York University
M.S., Computer Science
kp3291@nyu.edu

Shrish Singhal (sks9405)
New York University
M.S., Computer Science
sks9405@nyu.edu

Nikhil Soni (ns6062)
New York University
M.S., Computer Science
ns6062@nyu.edu

Abstract

The Math Question Answer Verification Kaggle competition aimed to predict whether the provided answer to a math question was correct. We fine-tuned the Llama3-8B model using LoRA to adapt it for this task. Our approach involved preparing structured input prompts and leveraging efficient parameter updates to train the model on a subset of the dataset. With improved training strategies, our model achieved an updated validation accuracy of 82.5%, demonstrating significant improvement in performance. This report details our methodology, results, and key insights.

1 Introduction

The Math Question Answer Verification competition tasked participants with developing a binary classification system to evaluate whether a given answer to a mathematical question was correct. This required leveraging large language models (LLMs) capable of reasoning and understanding detailed solutions.

We used Llama3-8B, a state-of-the-art pretrained language model, and fine-tuned it using Low-Rank Adaptation (LoRA). The competition provided a dataset containing mathematical questions, expected answers, detailed solutions, and a correctness label (‘is_correct‘). Our approach involved preprocessing the dataset to create structured prompts, fine-tuning the model efficiently with limited resources, and evaluating its performance.

2 Dataset

The datasets provided for the competition included a training set and a test set with the following characteristics:

```
DatasetDict({
    train: Dataset({
        features: ['question',
                   'is_correct',
                   'answer',
                   'solution'],
        num_rows: 1000000
    })
    test: Dataset({
        features: ['question',
                   'is_correct',
                   'answer',
                   'solution'],
        num_rows: 10000
    })
})
```

2.1 Data Sampling and Splitting

- We initially sampled **50,000 rows**, splitting it into:
  - **40,000 rows** for training.
  - **10,000 rows** for validation.

Subsequently:

- We sampled **110,000 rows** randomly from the original 1,000,000-row training dataset.
- The sampled data was split into:
  - **100,000 rows** for training.
  - **10,000 rows** for validation (testing).

Each sample consisted of:

- **question**: The math problem posed to the student.
- **is_correct**: A binary label indicating whether the provided answer matched the expected answer.
- **answer**: The expected answer to the question.
- **solution**: A detailed explanation of how the answer was derived.

Through experimentation, we found that changing the prompt order to `Question, Solution, Answer` improved the model's contextual understanding. Additionally, incorporating examples in prompts based on chain-of-thought prompting further boosted accuracy.

The 'Output' field was left blank during training, allowing the model to generate either 'True' or 'False' based on the input.

3 Model Description

3.1 Base Model

We were instructed to use Llama3-8B, a large language model designed for text generation and reasoning tasks. While Llama models excel in performance on language tasks, adapting them for mathematical reasoning required fine-tuning.

3.2 Fine-Tuning Technique

We employed Low-Rank Adaptation (LoRA), a parameter-efficient fine-tuning method that modifies a small subset of the model parameters. This approach allowed us to train the model on consumer-grade GPUs while maintaining computational efficiency.

3.3 Tools and Frameworks

- **Hugging Face Transformers**: For model loading and fine-tuning.
- **PyTorch**: For implementing training and inference pipelines.
- **Colab Pro**: To leverage more advanced GPUs like the Nvidia A100 for faster computation and lesser training and testing times.

4 Experimentation and Hyperparameters

Our experimentation involved systematic changes to model configurations, prompting strategies, and training parameters.

4.1 Initial Experiments

- **Default Configuration:** Training with 40,000 rows for 500 steps achieved an accuracy of **63%**.
- Researched LoRA parameters, quantization strategies, and chain-of-thought prompting techniques to enhance performance.

4.2 Progressive Improvements

- **Hyperparameter Adjustments:**
  - Experimented with 'r' values (16, 32, 64, 128)
  - Experimented with 'lora_alpha' values (16, 32, 64, 128).
  - Experimented with learning rates (ranging from 10^{-7} to 10^{-3}).
  - Experimented with different schedulers (linear, cosine, polynomial).
  - Experimented with lora dropout by keeping it non-zero to 0.1.
  - Experimented with weight decay by using the values 0.01, 0, and finally 0.001.
  - Experimented with 'per_device_train_batch_size' values ranging from 2 to 6.
  - Experimented with 'gradient_accumulation_steps' values ranging from 4 to 8.
  - Reduced max output tokens to 5 for concise predictions as the output was only 'True' or 'False'.
- **Prompts:**
  - Started with default prompt provided in the starter notebook.
  - Experimented with different aspects (Question, Answer, Solution) of the sample prompt, and ran training and validation after reordering these aspects to observe the effect of ordering.
  - Experimented with removing the 'Answer' and treating it as a redundant attribute since it was present as part of 'Solution'
  - Added example in the prompt for the model to further interpret the observations in the dataset.

  An example of a structured input prompt:

  Problem: Find the value of y if 4y - 7 = 9.
  Solution: Adding 7 to both sides, we get 4y = 16. Then, dividing by 4, we find y = 4.
  Answer: 4
  Output: True
- **Training Configurations:**

1. First we trained our model with 40,000 rows in training data on 500 max_steps and r = 16 and tested it on 10,000 rows with a batch_size of 16 on validation set taken from training set which gave us an accuracy of 63%.
2. Next we increased the number of observations to train our model to 100,000 rows with 5000 max_steps, increasing r and lora_alpha to 32 and batch_size of 32 with $lr = 5e - 6$, achieving 81% validation accuracy.
3. As we observed a significant increase in accuracy by increasing both r and lora_alpha, we further increased both of them to 128 and got an 82.5% accuracy on the validation set and the full test set provided to us.
4. We then trained the same model on additional 50,000 rows for 2,000 steps but with a lower learning rate of $5e - 6$ with polynomial scheduler, which decreased our model accuracy to 82.5% on validation set.
5. Studying these observations, the final training was conducted on a training set of 100,000 rows on 5000 max_steps, with a r and lora_alpha of 128 that achieved an accuracy of 82.5% on validation set of 10,000 rows and a 83.33% accuracy on the test set of 5,000 rows given in the competition and eventually 82.52% percent accuracy on the full test set (we first trained on polynomial scheduler but came to the conclusion that linear worked better)

4.3 Hyperparameters

Final hyperparameters:

Hyperparameter	Value
Learning Rate	2e-5
r	128
lora_alpha	128
Batch Size	32
Number of Epochs	3
Max Steps	5000
Quantization	4-bit
Scheduler	Linear

Table 1: Hyperparameter settings for fine-tuning.

5 Results

5.1 Performance Metrics

The model achieved an updated validation accuracy of 82.5%.

5.2 What Worked

- Using a larger sample (100,000 rows) for training improved the model's ability to generalize, boosting accuracy to 82.5%.
- LoRA fine-tuning with $\alpha = 128$ and $r = 128$ effectively enhanced parameter efficiency and model capacity.
- Structured prompts combining 'question', 'answer', and 'solution' provided contextual understanding to the model.

5.3 What Didn't Work

- Further training the model a second time with a lower learning rate did not change the accuracy.
- The model tended to overpredict 'True' for ambiguous or poorly phrased questions.
- We tried different schedulers like the polynomial and cosine schedulers but eventually the linear scheduler worked the best.
- Limited training steps constrained the model's performance, indicating a need for further optimization.
- Learning rate more than 1e-3 and below 1e-7 did not increase the accuracy.

5.4 Analysis

The improved accuracy indicates that sampling a larger subset of the data and performing additional validation yielded better generalization. However, the false predictions suggest areas for further refinement, particularly in handling complex or ambiguous problems.

6 Conclusion

This competition demonstrated the challenges and potential of adapting general-purpose LLMs for domain-specific tasks. With refined sampling and training, we significantly improved model accuracy from 63% to 82.5%. Future work could focus on expanding the dataset, leveraging task-specific embeddings, and exploring advanced fine-tuning strategies.

Limitations

The primary limitations include:

- False predictions indicate potential issues in model understanding of complex reasoning.
- Resource constraints such as fixed free resources on both Google Colab and Kaggle limited the scope of training iterations.

Acknowledgements

We thank professor Gustavo Sandoval for guidance and the Kaggle platform for providing the dataset and competition environment.

Links

- **Training and Inference Notebook**: https://github.com/kaartikeya15/Deep-Learning-Midterm/blob/038c5588746d759aaf032b703c20dad9f37fee33/Final_Notebook.ipynb
- **Model**: https://drive.google.com/file/d/1Z_nwsNsuDmT6SOcb0RO26YUE7KJ0YZk7/view?usp=sharing