Math Question Answer Verification Competition

Dhanesh Baalaji Srinivasan
ds7636@nyu.edu

Charan Kandasamy Raja
ck3740@nyu.edu

Sai Chandra Kaushik Siddavarapu
ss17413@nyu.edu

1 Introduction

The Kaggle Competition challenges the participants to improve the reliability of mathematical problem-solving systems. The focus of this competition is on using **Supervised Fine-tuning (SFT)** with the **Llama3.1-8B model** to assess the accuracy of answers provided for mathematical questions. Unlike conventional methods that prioritize generating answers, this task highlights the critical step of verifying their correctness.

The goal of the competition was to confirm that the answers to math problems were accurate. Building a model that could evaluate the given solution and assess its correctness was the challenge assigned to the participants. With the help of hyperparameter tuning and prompt engineering, the Llama3.1-8B model optimized with **LoRA (Low-Rank Adaptation)**, our team was able to achieve an accuracy of **82.04%**. We go into detail on the model design, experiments, results, difficulties, and the dataset preparation.

2 Dataset Description

The competition dataset consisted of:

- **Training Data**: 1,000,000 samples containing a math question, a provided answer, a detailed explanation, and a binary correctness label (`is_correct`).
- **Test Data**: 10,000 samples, with hidden labels used for leaderboard evaluation.

It is important to note that the `is_correct` labels in the test set are placeholders, all set to `True`. We were expected to fine-tune models that predict this variable accurately based on the information in the dataset.

Each sample included:

- **Question**: The math problem to solve.
- **Answer**: The provided solution to evaluate.
- **Solution**: A detailed reasoning process for deriving the answer.
- **Correctness Label**: A binary label (`True/False`) indicating the answer's validity.

3 Model Description

Our model architecture was based on the Llama3.1-8B transformer, augmented with LoRA adapters for efficient fine-tuning.

3.1 Base Model: Llama3.1-8B

Llama3.1-8B is a transformer-based architecture designed for a wide range of natural language processing (NLP) tasks.

3.2 Fine-Tuning with LoRA (Low-Rank Adaptation)

To adapt the model for the task of mathematical question answer verification, we employed LoRA, a parameter-efficient fine-tuning method. LoRA works by introducing low-rank updates to the model's pre-trained weight matrices, which reduces the computational cost and memory usage during training.

3.3 Frameworks and Libraries Used

- **PyTorch:** Used as the primary deep learning framework for implementing and fine-tuning the model, enabling GPU acceleration.
- **Transformers (from Hugging Face):** Provided pre-trained Llama3.1-8B model and the tokenizer integration.
- **Datasets (from Hugging Face):** Used for loading and processing the competition dataset hosted on Hugging Face.
- **Unsloth:** A library that facilitated optimized handling of Llama models, including rank-

stabilized LoRA and efficient inference with 4-bit quantization.

- **TRL (from Hugging Face):** Enabled supervised fine-tuning (SFT) of the Llama3.1-8B model with utilities like `SFTTrainer`.
- **NumPy:** Used for operations like dataset shuffling and sampling.
- **Pandas:** Used for generating our submission file to Kaggle.

3.4 Incorporating Reasoning via Prompt Engineering

The fine-tuning process incorporated customized prompt engineering, guiding the model to emphasize reasoning steps and logical deductions while limiting responses to binary labels only i.e `True` or `False`.

4 Experimentation

4.1 Hyperparameter Settings

To address the challenges of the Math Question Answer Verification Competition, we fine-tuned the Llama3.1-8B model using LoRA (Low-Rank Adaptation) techniques. Our goal was to optimize the model's performance in classifying answers as correct (`True`) or incorrect (`False`).

To guide the model more effectively, we modified the prompt stating that *"You are a great mathematician and you are tasked with finding if an answer to a given maths question is correct or not. Your response should be 'True' if correct, otherwise 'False'. The response should be only 'True' or 'False'. Below is the Question, Answer, Explanation for the solution and a boolean indicating if the provided answer is correct or wrong. True implies that it is correct and False implies that it is wrong."* to ensure binary outputs. Additionally, we changed the scheduler to **cosine** for better learning rate adjustments during training.

We also incorporated the **solution column** from the training dataset to impart a sense of reasoning to the model.

- **Key Hyperparameters:**
  - **Rank (R):** Adjusted to 16.
  - **LoRA Alpha:** Used the same value as Rank (R) which is 16
  - **Learning Rate:** Tested at 2e-4 and 1e-4 to balance convergence speed and stability.
  - **Learning Rate Scheduler:** Cosine scheduler.
  - **Batch Processing:**
    - `per_device_train_batch_size=2`
    - `gradient_accumulation_steps=4/8`
  - **Number of Training Steps:** Tested 2000 steps, 1 epoch and Early stopping when the gap between training and validation loss increased too much.
  - **Dataset Size:** Varied sample size from 30,000-100,000 for the experiments.
- **Observed Effects of Hyperparameter tuning:**
  - **Rank and LoRA Alpha:**
    - Higher values (R=32, Alpha=32) improved model convergence and accuracy. But takes a long time to train, because the higher rank increases the number of trainable parameters.
    - Lower values (R=16, Alpha=16) reduced computational costs but slightly hindered performance.
  - **Learning Rate:**
    - A rate of 2e-4 resulted in faster convergence.
    - Reducing the rate to 1e-4 lead to a smoothed training curve.
  - **Training Steps:**
    - Shorter training steps sometimes does not reach convergence
    - Prolonged training (e.g., 2000 steps) sometimes can lead to overfitting. We track that by using a validation set. We stop the training when the training loss decreases and validation loss increases.

4.2 Methods Tried

1. Fine-Tuning with Solution context and default scheduler (Model 1):

- **Configuration:**
  - **Rank (R):** 16
  - **LoRA Alpha:** 16
  - **Learning Rate:** 2e-4

- **Max steps:** 2000
- **Batch Size:**
  - `per_device_train_batch_size=2`
  - `gradient_accumulation_steps=4`
- **Scheduler:** Linear

- **Results:**
  - Accuracy: 80.592%.
  - Loss: Converged at ∼0.58.

- **Insights:**
  - We can see that adding the solution context during training and inference pushed us by over 5% baseline accuracy resulting in an accuracy over 80%.

2. Running for 1 Epoch, using Validation sets, lower learning rate, Sampling 100k, and using a cosine scheduler (Model 2):

- **Configuration:**
  - **Rank (R):** 16
  - **LoRA Alpha:** 16
  - **Learning Rate:** 1e-4
  - **Max steps:** 1 epoch
  - **Batch Size:**
    - `per_device_train_batch_size=2`
    - `gradient_accumulation_steps=4`
  - **Scheduler:** Cosine

- **Results:**
  - Test Accuracy: 81.85%.
  - Training Loss: Converged at $\sim$ 0.473 and Validation loss converged at $\sim$ 0.468

- **Insights:**
  - We can see that running for 1 epoch had an Incremental accuracy improvement, with increased computational demands. We also felt that 100k to be a large enough training set. Training took around 6 hours using an NVIDIA A100.

3. Reducing Training Samples, Increasing gradient accumulation steps, Modifying the prompt (Model 3):

- **Configuration:**
  - **Rank (R):** 16
  - **LoRA Alpha:** 16
  - **Learning Rate:** 1e-4
  - **Max steps:** 2000
  - **Batch Size:**
    - `per_device_train_batch_size=2`
    - `gradient_accumulation_steps=8`
  - **Scheduler:** Cosine (`lr_scheduler_type=cosine`)

- **Results:**
  - Accuracy: 82.08%.
  - Training Loss: Converged at $\sim$ 0.389 and Validation loss converged at $\sim$ 0.491

- **Insights:**
  - The increase in effective batch size by increasing the number of gradient accumulation steps helped along with the modified prompt.
  - Reducing the number of samples also helped us accelerate the training process. We were wary of overfitting by performing early stopping once the training loss decreased and validation loss increased.

These experiments demonstrate the impact of carefully tuned hyperparameters and iterative adjustments on model performance and generalization.

4.3 Ablation Studies

- Excluding the `solution` column during inference dropped accuracy to 69.8% even with LoRA.

- Excluding the `solution` column totally dropped accuracy to 56% even with LoRA.

5 Results

Here are the results obtained from the experiment.

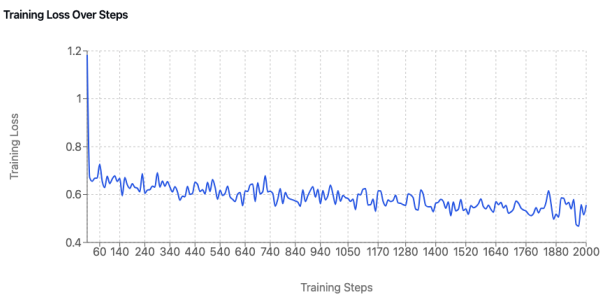

Figure 1: Training loss curve for Model 1.

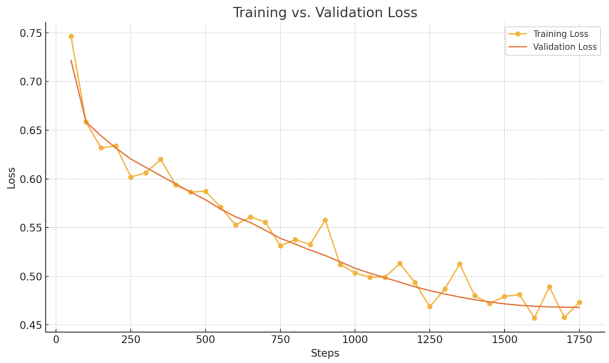

Figure 2: Training/Validation loss for Model 2.

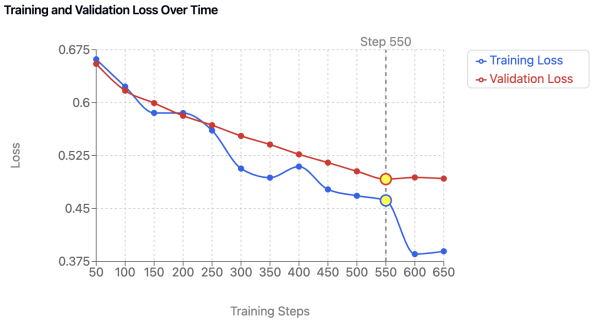

Figure 3: Training/Validation loss for Model 3.

6 Link to Training Notebook and Model Weights

- **Google Drive:** Download Model Weights
  Download the trained model weights.

7 Conclusion

The Llama3.1-8B model fine-tuned with LoRA demonstrated robust performance, achieving 82.04% accuracy. One avenue for future work is optimal sampling of the training data where we score the quality of data and sample high quality training data, which can improve our test accuracy.

References

- T. Hu, X. Shen, J. Wang, and D. A. Koutra. (2021). LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*. Available at: https://arxiv.org/abs/2106.09685

- B. Zhang, Y. Li, and P. Jiang. (2023). Unleashing the Potential of Prompt Engineering in Large Language Models: A Comprehensive Review. *arXiv preprint arXiv:2310.14735*. Available at: https://arxiv.org/abs/2310.14735