Deep Learning - Fall 2024 - Midterm Project Report

Adel del Valle
Tandon School of Engineering, NYU
MS in Computer Science
ad7082@nyu.edu

Jayanth Rao
Tandon School of Engineering, NYU
MS in Computer Science
jr6594@nyu

Joshua Alfred Jayapal
Tandon School of Engineering, NYU
MS in Computer Science
jj3811@nyu.edu

Abstract

This document compiles the methods and Supervised Fine Tuning (SFT) approaches performed on the Llama3.1 8B LLM to improve its correctness of solutions to math problems. The task is to fine-tune the LLM to predict whether a given answer for a mathematical problem is the right solution. This binary classification problem initially imposed a lot of challenges like poor accuracy of predictions, and we have documented our hyperparameter tuning search to find the right set of parameters to improve its accuracy. Finally, we were able to achieve a testing accuracy of 84.171% on the Kaggle contest submission, and proved that our tuning approach helped the model to learn the context and make better predictions.

1 Introduction

Over the past half-decade, we have seen a meteoric rise in the complexity, utility, and reliability of large language models (LLMs). These LLMs have shown promise in various fields such as automated reasoning or educational utility, but have also demonstrated limitations in more "basic" tasks. One such task is computational mathematics, where LLMs are limited by their linguistic training data. Alexandre Hamel (2024) notes in his paper that fields such as creative writing or historical analysis do not usually have a "binary" right or wrong answer. He claims that LLMs such as ChatGPT do not truly know if an answer *is correct*, only whether the answer *looks correct*.

We can extend this notion to the problem at hand, where we aim to improve the performance of the LLaMA-3.1 model by Meta when it comes to solving and verifying said solutions to high-school level math questions. This paper goes into depth on the usefulness of Low-Rank Adaptation (LoRA) as an approach to training LLMs. Introducing LoRA provides task-specific improvements without compromising on computational complexity. We also look into the hyperparameters of the Supervised Fine-Tuning (SFT) process, and how we improve the reasoning capability of the LLM by introducing the problem explanation as a learning mechanism.

Through our experiments, we show that it is possible to improve the performance of LLaMA-3.1 by utilizing the LoRA technique and by diving deeper into hyperparameter tuning and prompt engineering to better recognize correct solutions to various mathematical problems. The remainder of the document is as follows. Section 2 describes the structure and distribution of the dataset provided in the competition. Section 3 describes the LLaMA-3.1 8B model and a short description of its modules that must be fine-tuned. Section 4 describes a list of hyperparameters that we have modified in various combinations and their corresponding testing results on the competition, followed by a conclusion in Section 5. Also, you can find our notebook source code and model weights in the Appendix section.

2 Dataset

The competition dataset was a collection of high-school-level mathematical questions, along with their answers and solutions. The question and solution columns comprised large text sequences with numerals and symbols. The answer column contained a one-line answer for the given question prompt. The target variable is the "is_correct" column which has a True value if the corresponding answer is the right expected value for the corresponding problem prompt. The solution column further extends the method behind the answer calculated, which can be used to improve the model's understanding and decision-making. The training dataset contains 1,000,000 records, while the test set contains 10,000 records.

Dataset Split	Number of Samples
Train	1,000,000
Test	10,000

Table 1: Data Distribution of the given dataset.

3 Model Description

The LLaMA-3.1 model is the latest in a family of LLMs produced by Meta AI to achieve efficient performance over a range of natural language processing tasks Meta (2024). The 8B model, consisting of eight billion trainable parameters, is a medium-sized version that was mandated in the competition.

The LLaMA-3.1 8B is a decoder-only architecture and it caters to text generation tasks through autoregressive sampling. The model is comprised of 56 transformer layers, each integrated with multi-head attention and feedforward network modules. It is considerably lightweight and can run on systems with GPUs containing 8 to 16GB of VRAM. The model is versatile in applications like code generation, and language translation, and we intend to fine-tune it to understand and classify whether a given answer to a math problem is correct or not. We intend to fine-tune this model using Low Rank Adaptation (LoRA) and hyperparameter tuning.

LoRA is a compute-efficient method to fine-tune LLMs Hu et al. (2021). LoRA introduces an additional matrix layer to the LLM (considerably lower rank), and the weights in this matrix are optimized. The layers of the LLM are frozen during the process, and fine-tuning the adaptation layer is faster, and is possible on low end GPUs. This method may degrade the modelś accuracy by a small amount, but it is an efficient method to tune LLMs according to the given problem or context.

4 Experimentation

This section discusses the various approaches of LoRA and hyperparameter tuning to improve the model performance beyond its baseline performance. The section begins with the list of tuned parameters along with their explanations, and the corresponding testing accuracy results and discussions. All experiments were scripted as a .ipynb notebook, and run on Colab T4 and A100 GPUs with 15GB GPU RAM.

4.1 Data Splitting

Since our testing dataset had dummy placeholders for the target variable, we extracted a small subset from the training dataset to validate our model before making the submission. The validation set consists of 10,000 samples to mirror the test set, and the accuracy of the model is preemptively calculated with this subset of data.

4.2 Hyperparameter tuning

Below are the list of hyperparameters to our interest that we decided to tune to improvise the given model.

- **LoRA rank (R):** It is the dimensionality of the low rank matrices. The higher the rank, the more information the matrix can store, but can increase computation requirements. We aim to select the optimal rank from the range [8, 16, 32, 64, 128], balancing the model's capacity to store information with computational efficiency.

- **LoRA alpha (α):** It is a scaling factor that determines the contribution measure of the LoRA matrix. a higher α (if larger than r) will make the model rely more on the low rank matrix rather than its actual weights, leading to overfitting. We also intend to select an optimal α from the same range as r.

- **Gradient accumulation steps (GAS):** It is the number of mini-batches the gradient accumulates before making the weight updates through backpropagation. A right balance in the accumulation steps can help faster convergence with limited memory requirements.

- **Weight decay (λ):** a regularization parameter to penalize complex weights during training. We chose random weight decay values within the range [0.01, 0.05] to prevent overfitting.

- **Learning rate (LR):** the rate at which the weights are updated during training. A higher rate can overshoot the parameters to instability, while a lower rate requires more training steps as convergence happens very slowly. We pick random rates within the range [1e-5, 1e-3] during our tests.

- **Batch size (BS):** Batch size is the number of training samples that the model receives and

#	R	α	GAS	λ	LR	BS	WS	MS	Epochs	Train Loss	Val Acc	Test Acc
1	16	16	2	0.05	3e-4	8	10	-	2	0.7542	0.452	0.4461
2	16	16	2	0.05	5e-4	2	5	30	-	0.7892	0.3843	0.3729
3	16	16	2	0.01	2e-4	8	10	400	-	0.7401	0.4811	0.4838
4	16	16	4	0.01	2e-4	8	10	500	-	0.7198	0.5254	0.5166
5	32	32	32	0.01	2e-5	2	10	400	-	0.6024	0.6291	0.6165
6	64	128	32	0.01	1e-5	4	35	500	-	0.5887	0.7696	0.6994
7	64	128	32	0.01	1e-5	4	30	350	-	0.5892	0.7012	0.6994
8	64	64	32	0.01	1e-4	8	40	400	-	0.4785	0.8567	0.8449
9	64	80	32	0.01	1e-4	8	25	250	-	0.5394	0.8528	0.8397
10	64	64	32	0.01	1.2e-4	8	25	250	-	0.4934	**0.8811**	**0.8451**
11	64	64	32	0.05	2e-3	8	50	-	1	0.5032	0.7993	0.7704
12	64	64	32	0.013	1.4e-4	8	35	350	-	0.4396	0.8811	0.8398

Table 2: Hyperparameters of the model and their corresponding performances with the test set.

learns before updating its weights. We also try different batch sizes (in powers of two) to find the right balance for the model to train.

- **Warmup steps (WS):** Number of iterations after which the current learning rate is reset to the initial learning rate value if the current rate is too small or large.

- **Max steps (MS):** Maximum number of steps or iterations that the model will train for. This has more control than epoch training, especially when it comes to fine-tuning LLMs with a large dataset.

- **Training Prompt:** The prompt used to train the model and respond in a specific pattern. We were provided with an original prompt that did not utilize the Solution/Explanation field in the dataset. We amended this prompt to simplify the language and be more direct while also incorporating the Solution/Explanation field.

All the aforementioned parameter combinations were given for fine-tuning the model, and the model was tested against our validation set of samples, and assessed its testing performance through submissions.

4.3 Results & Discussion

This section explains the twelve submission attempts to improve the LLaMA-3.1 8B model, each attempt with a different combination of hyperparameter settings to fine-tune the model. Table 2 compiles the parameters used in every attempt and the corresponding validation and test accuracy during submission.

Submissions that caused very poor results helped us to configure our parameters for the next iteration. For instance, whenever the α factor exceeded the rank of the matrix, there was a decrease in the testing accuracy, as seen in submissions 6,7, and 9. A lower rank and α value in the initial runs were too small, with fewer parameters that gave low scores. It is to be noted that there was a significant increase in the accuracy of the model after the 8th submission when the prompt was redesigned by adding the "solution" column to the prompts. The context for the answer provided more detail to the model to effectively assess the correctness of the answer. Moreover, an improved batch size and max steps (MS) included more samples and iterations, gradually contributing to the model's performance.

After the test runs, it was evident that the higher magnitudes of parameters R, α, BS, MS, and LR improved the accuracy of the model. The best of the twelve attempts was submission 10, which gave a testing accuracy of 84.51% on the competition, and the final testing accuracy turned out to be 84.171% when assessed with the remaining 50% of the test samples. However, the training time in each of these attempts took a long time than usual, but since training time was not under the scope of the competition, we improved the parameters to their optimal values that gave the highest performance, with a tradeoff of large training times.

5 Conclusion

Implementation of SFTs can help LLMs adapt to task-specific applications and this competition was

a valuable opportunity to face practical challenges and explore its potential. This competition focused on fine-tuning an LLaMA-3.1 8B to assess whether a given answer and the solution context are the right expected answer for the given math problem prompt. This classification task was accomplished by utilizing LoRA. With various test runs with combinations of hyperparameters, we were able to produce a fine-tuned model that rendered a final testing accuracy of 84.171% on the Kaggle competition, proving that our attempts improved the model beyond the baseline performance.

References

Alexandre M. Hamel. 2024. Math, chatgpt, and you: The problem with mathematical accuracy in large language models. *Dartmouth Digital Commons*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Meta. 2024. Introducing llama 3.1: Our most capable models to date.

A Appendix

This section provides the mathematical formulations used in the implementation of LoRA, QLoRA, Gradient Descent, AdamW, and related methods.

A.1 Low-Rank Adaptation (LoRA)

LoRA adapts a pretrained model by introducing low-rank matrices to approximate the weight updates during fine-tuning. The weights W of a given layer are decomposed as:

$$W' = W + \Delta W \quad \text{where} \quad \Delta W = AB$$

Here:

- $A \in R^{d \times r}$ and $B \in R^{r \times k}$, where $r \ll \min(d, k)$, represent low-rank matrices.
- W is frozen during fine-tuning, and only A and B are trained.

A.2 Quantized LoRA (QLoRA)

QLoRA applies 4-bit quantization to reduce memory requirements while fine-tuning large models. Quantization approximates full-precision weights W by:

$$W \approx \text{DeQuant}(\text{Quant}(W)) + \Delta W$$

Where:

- $\text{Quant}(W)$: Quantizes W into a lower precision (e.g., 4-bit integers).
- $\text{DeQuant}(\cdot)$: Restores approximate full-precision values.
- ΔW: Low-rank matrix added via LoRA.

A.3 Gradient Descent

Gradient Descent minimizes a loss function $L(\theta)$ with respect to model parameters θ using:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t)$$

Where:

- η: Learning rate.
- $\nabla L(\theta_t)$: Gradient of the loss function at iteration t.

A.4 AdamW Optimizer

AdamW extends the Adam optimizer by incorporating weight decay for better regularization. Updates for parameters θ are:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(\theta_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L(\theta_t))^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \lambda \theta_t$$

Where:

- β_1, β_2: Exponential decay rates for moment estimates.
- λ: Weight decay coefficient.
- ϵ: Small constant for numerical stability.

A.5 Cross-Entropy Loss

The classification task leverages cross-entropy loss to compare predicted probabilities $p(y|x)$ with true labels y:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log \hat{y}_{ij}$$

Where:

- N: Number of samples.
- C: Number of classes (here $C = 2$).
- y_{ij}: True label for class j of sample i.
- $\hat{y}_{ij}$: Predicted probability for class j of sample i.

A.6 Training Configuration

Key hyperparameters used in the training:

- **Learning Rate** (η): Adjusted dynamically using a linear scheduler.
- **Batch Size (BS)**: Number of samples processed in parallel per iteration.
- **Warmup Steps (WS)**: Number of initial steps with a linearly increasing learning rate.
- **Gradient Accumulation Steps (GAS)**: Steps over which gradients are accumulated before performing an update.

A.7 Implementation Details

The complete implementation is available in the attached notebook, including fine-tuning code, inference scripts, and the final model weights. Please find the notebook containing the model training and inference code sections, and the model weights below:

- **Project Notebook**
- **Model Weights**