Supervised Fine-Tuning of Meta-LLaMA-3.1-8B for Solution Verification to Math Problems

Nidhushan Kanagaraja nk3755@nyu.edu
Stavan Christian snc8114@nyu.edu

Abstract

Computational optimization for fine-tuning Large-language models (LLMs) to instruct tasks has been well explored within the deep learning community. This work is aimed at highlighting one such optimization technique called Low-Rank Adaption (LoRA) for fine-tuning LLMs. Particularly, we fine-tune the pre-trained Meta-Llama 3.1-8B model, provided by the Unsloth community, to the instruction task of verifying given solutions to mathematical problems. In this work, we provide the details of the experimental methodologies with the model hypermeter space and LoRA configuration that we employed for fine-tuning and thereafter concluding remarks based on the observations and results obtained.

1 Introduction

The latest Large Language models contain billions of parameters and thus are computationally expensive to train. More often than not, instead of training an LLM for specific or contextual tasks such as medical diagnostics and summarization, LLMs are trained on a large public corpus dataset for language models and then later fine-tuned for the specific objective task. This is done by training the model on a relatively smaller custom training data for that specific task. Even on this relatively smaller training data, fine-tuning LLMs would require updating billions of parameters which is computationally slow and expensive. Simply put, LoRA provides a much more efficient technique for fine-tuning by freezing the pre-trained weight matrix, say $P_{m,n}$ where m, n are the dimensions of the matrix, and creating a separate "delta/update" weight matrix, say $W_{m,n}$, which is representative of the fine-tuning updates to the pre-trained weight matrix. This so-called update weight matrix is computed as a matrix multiplication of two matrices, $U_{m,k}$ and $V_{k,n}$, where k is a hyperparameter of the LoRA configuration. The idea is that rather than updating a weight matrix of size m, n where $m \times n$ is huge, we update the two matrices of size m, k and k, n, where $m \times k + k \times n << m \times n$.

We were given the task of fine-tuning the Meta-Llama 3.1-8B model using LoRA to evaluate the correctness of given answers to mathematical questions. Along with the answers, an accompanying explanation is also provided to justify the answer. This is how the rest of the report is structured: - In section 2 we discuss the training dataset contents and the pre-processing performed on the dataset. Section 3 details the model used along with the parameters of the LoRA implementation provided by the peft module of the "FastLanguageModel" library of unsloth. In Section 4 we discuss the hyperparameter search experiment for optimum fine-tuning of the model, post which Section 5 discusses the results obtained followed by a discussion and conclusion in Sections 6 and 7 respectively.

We provide the GitHub repository link to our best three performing models' code (.ipynb notebook) and model weights [5].

2 Dataset

The dataset for this task is the Math Question Answer Verification Competition dataset, available on Hugging Face. It consists of questions from various math topics, structured into several columns as follows: -

question: The math question posed to the student.

answer: The ideal or correct answer to the question.

solution: A detailed reasoning or solution that explains the answer, which participants can use to enhance their model's understanding of the answer's correctness.

is_correct: The target variable indicating whether the answer provided is correct (True) or incorrect (False).

The "is_correct" column in the test dataset contains all TRUE values as a placeholder.

3 Model

Model Information: The Meta Llama 3.1 collection of multilingual LLMs is a collection of pretrained and instruction-tuned generative models in 8B, 70B, and 405B sizes (text in/text out). The Llama 3.1 instruction-tuned text-only models (8B, 70B, 405B) are optimized for multilingual dialogue use cases.
Model Architecture: Llama 3.1 is an autoregressive LM that uses an optimized transformer architecture. The tuned versions use supervised fine-tuning and reinforcement learning with human feedback (RLHF) to align with human preferences for helpfulness and safety.
Intended Use Case: Llama 3.1 is intended for commercial and research use in multiple languages. Instruction-tuned text-only models are intended for assistant-like chat, whereas pretrained models can be adapted for a variety of NL generation tasks. The Llama 3.1 model collection also supports the ability to leverage the outputs of its models to improve other models including synthetic data generation and distillation.
Particularly, we utilize the pretrained Meta-LLaMA-3.1-8B model provided by Unsloth on their Hugging face repository. We import the model weights as well as the tokenizer using the "from_pretrained" method of the "FastLanguageModel" library from unsloth. Also, we load the weights with 4-bit quantization to save memory and faster compute.
Thereafter, we utilize the "get_peft_model" method of the "FastLanguageModel" library, to setup the LoRA configuration/parameters for fine-tuning which are discussed in the following section. We choose to apply LoRA fine-tuning to the following layers of the model: - `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj` and `down_proj`.

4 Hyperparameter Experimentation for Supervised Fine Tuning

In this section we discuss the set of hyperparameters we considered and their search space to find the optimum fine-tuning configuration for the given task. **Note that there are other parameters as well but we note the ones which we explored for fine-tuning.**

4.1 Hyperparameter Tuning

First, we list the hyper-parameters considered for the LoRA configuration in the fine-tuning process:

- **r:** Rank of the low-rank decomposition for factorizing weight matrices
  Impact: -

  Higher: Retains more information, increases computational load.

  Lower: Fewer parameters, more efficient training, potential performance drop if too small.

- **lora_alpha:** Scaling factor for the low-rank matrices' contribution.
  Impact: -

  Higher: Increases influence, speeds up convergence, risks instability or overfitting.

  Lower: Subtler effect, may require more training steps

- **lora_dropout:** Probability of zeroing out elements in low-rank matrices for regularization.
  Impact: -

  Higher: More regularization, prevents overfitting, may slow training and degrade performance.

  Lower: Less regularization, may speed up training, risks overfitting.

- **bias:** Bias applied to the LoRA layers

- **use_rslora:** Enables Rank-Stabilized LoRA (RSLora).
  Impact: -

  True: Uses Rank-Stabilized LoRA, setting the adapter scaling factor to lora_alpha/math.sqrt(r), which has been proven to work better.

  False: Uses the original default scaling factor lora_alpha/r

We list the hyper-parameters considered for the supervised fine-tuning training process: -

- **per_device_train_batch_size:** The batch size per GPU for training.

- **gradient_accumulation_steps:** Number of updates steps to accumulate the gradients for, before performing a backward/update pass
  Impact: - A higher value gives stability to the gradient update process.

- **warmup_steps:** Number of steps used for a linear warmup from 0 to the learning rate.
  Impact: - A higher value gives stability to the training process.

- **max_steps:** The total number of training steps to perform. Overrides the "num_train_epochs" parameter (Total number of training epochs to perform).
  Impact: - Generally a higher value leads to better fitting to the train data but it saturates and might lead to overfitting if too high a value.

- **learning_rate:** The initial learning rate for AdamW optimizer

- **weight_decay:** The weight decay to apply (if not zero) to all layers except all bias and LayerNorm weights in AdamW optimizer.
  Impact: - Form of regularization used to prevent overfitting by penalizing large weights.

- **lr_scheduler_type:** The learning rate scheduler type to use.

- **max_grad_norm:** Maximum gradient norm (for gradient clipping).
  Impact: - Used to clip gradients greater than a threshold to avoid exploding gradients.

4.2 Prompt Design

One important aspect considered while fine-tuning the model was the "instruct-prompt" which was appended to the training and test examples as part of pre-processing before the training and inference process respectively. This prompt works as an instruction to the model during the training and the inference procedure and in a way steers the model in the right direction leading to generate the required output.
We experimented with multiple prompts and found that being instructive and concise as well as giving a step-by-step approach to complete the task leads to better-accuracy-yielding prompts. One such prompt that we used in our best-performing model is as follows: -

You are a highly experienced mathematician and logical thinker. Your task is to carefully verify if the provided answer to the math question is correct. You will analyze the question, the given answer, and the explanation provided. Your response should strictly be based on factual correctness, logical accuracy, and mathematical validation.
Math Question:{}
Provided Answer: {}
Solution Explanation: {}
Verification Process:
1. Confirm if all calculations are correct.
2. Ensure that the answer logically follows from the solution explanation.
3. Double-check for any errors or inconsistencies in the reasoning.
Final Decision: Take a moment to consider: Is the provided answer logically sound and mathematically accurate? Respond with 'True' or 'False' accordingly.
Final Output (True/False):

4.3 Training Process

For supervised fine-tuning, we utilized the SFTTrainer method from the trl library.[3]

4.4 Hyperparameter search space

Following are few of the hyper-parameter options that we used to fine-tune the model: -

M*	r	LA*	PDBS*	GAS*
v1	16	16	2	4
v2	64	16	2	8
v3	32	32	2	16
v4	64	32	2	16
v5	64	32	2	16
v6	128	256	4	2
v7	128	256	4	4
v8	128	128	4	2

Table 1: M - Model version, LA - lora_alpha, PDBS - per_device_batch_size, GAS - gradient_accumulation_steps

We set "lora_dropout" to 0 since we did not seem to overfit the training data given the fact that we trained over a fraction of the dataset (as is evident by the number of "max_steps" parameter, given the computational constraints. We also set the "bias" to

M*	WS*	MS*	LR*	WD*	LRS*
v1	5	100	2e-4	1e-2	linear
v2	20	200	2e-5	1e-2	linear
v3	20	200	2e-4	1e-2	linear
v4	100	200	1e-4	1e-2	CWR*
v5	100	300	1e-4	1e-3	CWR*
v6	30	500	2e-5	1e-3	CWR*
v7	40	800	1e-5	1e-3	CWR*
v8	120	800	1e-5	1e-3	cosine

Table 2: M - Model version, WS - warmup_steps, MS - max_steps, LR - learning_rate, WD - weight_decay, LRS - linear_rate_scheduler, CWR - cosine_with_restarts

"None" since it is the optimized choice [1]. Moreover, since setting "use_rslora" is known to perform better, we set this to true for all our fine-tuning experiments with the hyperparameter search space.

5 Results

The fine-tuned model `128_256_N` achieved the highest accuracy of **0.78703**, significantly outperforming the baseline accuracy of **0.60031**. Table 3 provides a comparison of performance metrics across different configurations.

Model	Accuracy	Loss
Baseline	0.60031	0.7296
v6	0.78703	0.413
v7	0.78484	0.425
v8	0.78425	0.438

Table 3: Model performance summary.

6 Discussion

The following insights emerged from our analysis of the models:

- **LoRA's Efficiency:** Leveraging LoRA for fine-tuning reduced computational overhead while retaining high accuracy. The decomposition of weight updates into low-rank matrices allowed for faster training and lower memory usage, demonstrating that fine-tuning massive models can be computationally feasible without full parameter updates.

- **Role of Hyperparameters:** The study revealed that a combination of LoRA parameters, such as `r`, `lora_alpha`, `lora_dropout`, and fine-tuning parameters like learning rate, batch size, and gradient accumulation steps, played a critical role in balancing efficiency and performance. This highlights the need for a comprehensive search of hyperparameter configurations to achieve stable loss and enhanced accuracy.
  Specifically, we conclude that increasing 'r', the rank of the low-rank decomposition for LoRA, as expected, leads to better performance in general. This is because we retain more information with higher-rank matrices.
  Also, we initially thought that using a higher gradient_accumulation_steps would impart stability to the training process and thus lead to better performance but that wasn't the observation.
  We observed that in general having 100-120 warm-up steps is a good choice. One important observation we made is that after about 150-180 train steps, the train loss doesn't improve and saturates no matter the choice of hyper-parameters.
  Regarding the learning rate, we observed that using a value of 1e-5 and that a "cosine" scheduler" generally leads to stable decrease in the training loss as compared to a "linear" or "cosine with restarts" scheduler.
  Moreover we observed that a lesser value of the "weight_decay" parameter, used for regularization, as indicated in table 2 in section 4, was a reasonable choice as we weren't training for many training steps or just a fraction of the training dataset.

- **Prompt Design as a Game-Changer:** A structured, instructive prompt significantly improved the model's reasoning. The step-by-step approach, combined with concise instructions, helped the model better evaluate mathematical reasoning and solution correctness.

- **Challenges and Limitations:**
  - *Resource Constraints:* Limited access to computational resources restricted the exploration of larger batch sizes, additional epochs, and more diverse datasets.
  - *Generalizability:* While the model performed well on the given dataset, its robustness to unseen mathematical problems or other domains remains untested.

7 Conclusion

The study underscores that fine-tuning, when paired with tailored prompts and optimized configurations, can unlock the potential of large models for specific, computationally expensive tasks. Future work should explore generalizability and scalability to strengthen these findings.

Ethics Statement

This work adheres to AI research ethics, emphasizing transparency, fairness, and reproducibility. The code, weights, and methodology are shared for collaborative improvement.

Acknowledgments

We thank Professor Gustavo Sandoval and the teaching assistants for organizing the Math Question Answer Verification Competition[6] and providing invaluable resources and guidance throughout this project.

References

1. Starter Notebook for Supervised fine-tuning of Llama 3.1-8B
2. Documentation for LoRA parameters
3. Documentation for the SFT Trainer
4. Documention for Training Arguments of the SFT Trainer
5. GitHub repository for code and model weights
6. Amardeep Kumar (AD). Nyu-dl-fall-24-competition. https://kaggle.com/competitions/nyu-dl-fall-24-competition, 2024. Kaggle.

A Appendix

A.1 Additional Prompts

Following are some of the other prompts that we tried with the models mentioned in the section 4: -

- *Goal: - To verify the correctness of the given answers to math questions using the explanation provided.*
  Rules: -
  1) You cannot afford to incorrectly verify these answers.
  2) DO NOT BE BIASED BY THE SOLUTION/EXPLANATION GIVEN.
  Steps: -
  1) Read the question and the explanation of the given answer with all due attention.
  2) Then, provide step by step reasoning for your verification of the correctness of the given answers.
  3) Then again, go through your reasoning and check for any mistakes.
  4) If you are sure that the given answer is correct, then your final output should be 'True', otherwise if the given answer is incorrect or labelled as 'N' , then your final output should be 'False'.
  ### Question: {}
  ### Answer: {}
  ### Explanation: {}
  ### Output: {}

- *You are a great mathematician and are tasked with the MOST important task of verifying if a given answer to a math question is correct. Analyze the explanation given for the answer and provide your reasoning to verify the correctness of the answer. The future of models depends on this, thus be accurate. Your final response should be 'True' if the answer is correct, otherwise 'False'.*
  ### Question: {}
  ### Answer: {}
  ### Explanation: {}
  ### Output: {}

- *My job is at stake. I need to verify the correctness given answers to math questions using the explanation provided. You cannot afford to incorrectly verify these answers.*
  Read the question and the explanation of the given answer with all due attention. Then, provide step by step reasoning for your verification of the correctness of the given answers. Then again, go through your reasoning and check for any mistakes. If you are sure that the given answer is correct, then your final output should be 'True', otherwise if the given answer is incorrect, then your final output should be 'False'.
  ### Question: {}
  ### Answer: {}
  ### Explanation: {}
  ### Output: {}