Summary of Fine-Tuning Mistral-7B

By: Yugal Singh

Model and Dataset Selection:

For this project, I have used the **unsloth/mistral-7b-q4**. It's a 4-bit quantized version of the powerful Mistral-7B model, it was purely an experimental choice, but about It I knew that it can offers a good balance between high performance and manageable memory requirements, allowing me to train effectively even with limited GPU resources. The "Unsloth" library helped me to make the whole process significantly faster.

In order to teach the model how to follow instructions better, I used the well-known **tatsu-lab/alpaca** dataset. The Alpaca dataset is essentially a collection of instruction-and-response pairs, perfect for this kind of fine-tuning. To keep the training process quick and manageable for this initial run, I decided not to use the full dataset. Instead, I have just used a smaller, representative **subset of 1,500 examples**.

Training Configuration & Hyperparameters:

As specified in the assignment I have used the **LoRA (Low-Rank Adaptation)** technique for the fine-tuning of the **unsloth/mistral-7b-bnb-4bit**. It's a method which, allows us to train the model much more efficiently it only updates a small number of additional weights, rather than the training the entire model.

Here are the key hyperparameters we used for the training run:

Hyperparameter	**Value**	**Purpose**
LoRA Rank (r)	16	Size of the trainable LoRA matrices.
LoRA Alpha (lora_alpha)	16	A scaling factor for the LoRA weights.
Epochs	2	The number of times we trained on the dataset.
Batch Size	8	Number of samples processed per device.
Gradient Accumulation	4	Simulates a larger batch size (effective 32).
Learning Rate	2e-4	Controls how much the model's weights update.
Optimizer	adamw_8bit	An efficient, memory-saving optimizer.

LR Scheduler	cosine	Adjusts the learning rate during training.

Performance: Before vs. After:

The results I got was a per my expectation, I measured the model's performance on the **arc_easy** benchmark, a common-sense reasoning task, I also tried to benchmark it on others like **hellaswag**, and **piqa**. After fine-tuning the model, I was able to achieve some improvement in accuracy.

Metric	Baseline (Before)	Fine-Tuned (After)	Improvement
Accuracy	79.80%	**81.19%**	**+1.39%**
Normalized Acc.	78.49%	**80.47%**	**+1.98%**

These results are achieved from a relatively short training cycle on a small data sample.

Ideas for Further Improvement:

1. **Training on a Larger Dataset:** Currently I have used only 1,500 samples from the Alpaca dataset and I guess next step can be; to train the model on the full dataset or we can try to fine-tune the model with much larger portion of it. This would expose the model to a wider variety of instructions and it should result in more robust and generalized performance improvements.
2. **Experimenting with the LoRA Configurations:** I have used a LoRA rank of 16, which is a most widely used However, we could experiment with a larger rank (like 32 or 64). As the higher rank can allows the model to learn more complex patterns, which could result in achieving better performance, but doing so requires more memory and results in slower training. but it would be worth exploring this trade-off.