CS7643: LoRA Finetuning on GPT-2 Report

Introduction

Language Models like ULMFiT [1], GPT-2 [2], and BERT [3] have demonstrated the remarkable effectiveness of fine-tuning pre-trained models in natural language processing tasks. Before that, using pretrained models was only the norm in computer vision; in the context of NLP, pretraining was limited to word embeddings. Fine-tuning these models is important as it allows for specialization of these general models for specific tasks. The most straightforward approach is full model fine-tuning, where all parameters are updated. However, with model sizes exploding, this is computationally expensive and requires significant hardware resources. Parameter efficient fine-tuning techniques like LoRA (Low Rank Adaptation) [4] were invented to overcome these issues. LoRA adapts the model to new tasks by updating only a small subset of newly introduced trainable parameters. LoRA represents the updates to the original weight matrices as the product of two smaller matrices (low-rank matrices). During finetuning, only the parameters within these smaller matrices are trained, while the original pre-trained weights remain frozen

In this project, we survey the effectiveness of LoRA fine-tuning approaches compared to full fine-tuning for four down-stream NLP tasks. Our goal is to provide a clear, practical guide for anyone wanting to fine-tune language models. By understanding where LoRA works best and how to use it effectively, developers can save significant time and computing resources, making powerful language AI more accessible and practical for specific applications.

Datasets

We use four tasks. The first is instruction fine tuning which is a method of further training a pre-trained LLM on a dataset of instructions paired with desired responses. The goal is to make the model better at understanding and following natural language instructions. For this task, we use the Stanford Alpaca dataset [5], which contains 52,002 synthetic question, and desired response pairs.

Due to compute constraints, we only use a 13.5K sample subset of the original data. In addition, to minimize memory usage, we produce this subset by choosing samples where the response length is less than 512 characters. The training set is 12K samples, whereas the validation set is 1.5K samples. We use 100 samples for the test set, which is used for manual analysis and is thus comprised of 50% simple instruction tasks like “convert to passive voice” and 50% open ended tasks like “compose a tweet on global warming”. For preprocessing, we use a very simple pipeline: 1) apply the Alpaca prompt style [5], tokenize using GPT-2 tiktoken [6].

TODO (other 3 datasets)

Approach

We use GPT-2 as the base pretrained LLM. First, we implemented the GPT-2 model architecture from scratch in PyTorch. We used Sebastian Raschka’s GPT-2 implementation [7] as a reference and Andrej Karpathy's tutorial [8] as a guide to understand the model. Second, we loaded the pretrained weights from Hugging Face [9]. The next steps after this slightly differed from task to task, but we state the general idea here. For the 3 classification tasks, the output linear layer was replaced with a Linear layer that has appropriate number of required outputs. Third, we fine-tuned the full model. Fourth, we fine-tuned the LoRA version of the model by replacing the linear layers (the projection matrices for attention, and the feedforward layers in the Transformer block) with LoRA linear layers. Lastly, we performed hyperparameter analysis on the model size (small-124M medium-355M, large models-774M), batch size, number of epochs, LoRA rank, learning rate schedule (warmup ratio and optimizer), and the number of transformer blocks to fine-tune vs freeze.

For the three classification tasks we used accuracy as a metric. For the instruction fine-tuning task, we used perplexity as the metric. However, after judging model generations, we soon noticed that perplexity was not a good judge in telling the difference in model ranking quality. Therefore, we used an external LLM – Gemini 2.0 Flash Thinking via API - to rank the model responses on a scale of 0-100 by comparing it to the target response. This approach also had a few problems: 1) due to stochasticity, the scores would slightly vary, e.g. the same response once got a score of 60, and then a score of 70 next time. 2) Due to rate limits and response time, we only used 100 examples for evaluation (which took about 20 minutes). This gave us first-hand experience in the incredible challenges of evaluating language model generations. Despite these limitations, we found the "Gemini eval" to be good enough for relative ranking of models and more importantly correlating decently with human judgement.

We used Google Colab as the compute platform using A100, L4, and T4 GPUs as appropriate. One of the biggest problems we encountered was computation constraints. For fine-tuning the full model, we had to use A100 GPU, since about 30 GB of RAM was required for even the medium size model. To prevent out-of-memory errors, we employed the following techniques: 1) reducing the batch size to 2 or 4; 2) reducing the number of epochs to 1 or 2; reducing the context length to 512. We ran our medium GPT-2 LoRA experiments on T4 GPU (15 GB RAM) as less compute was required, and to preserve compute units to do more experiments.

While training the immediate problems encountered were 1) overfitting and 2) convergence instability. These were mainly solved by improving the learning rate schedule and will be discussed in the following sections.

Experiments & Analysis - Instruction Fine Tuning

We observed that instruction fine-tuning primarily teaches the model the *format* and *style* of responding to instructions. It learns *how* to answer a question, *how* to perform a task like summarization or translation, but it doesn't necessarily inject vast amounts of new factual knowledge or fundamentally correct flawed knowledge already present in the base model.

Full Model Fine-tuning

Finetuning the full model gives a benchmark for the best we can hope to achieve by finetuning. The first difficulty encountered during finetuning the full model for GPT-2 large (774M) was overfitting.

As illustrated in figure 1, when using a constant learning rate of 5e-5, the model overfits as the gap between the training and validation losses grows over the course of training. The first attempt at fixing this was to add a learning rate schedule as shown in figure 2. Here, the learning rate rises linearly from 1e-5 to 5e-5 for 20% of the steps, then decays back to 1e-5 using cosine annealing. This helped reduce the overfitting, but the validation loss curve was slightly erratic, falling rapidly, then slowly rising and then falling steading again. To increase stability of training, max gradient norm clipping to 1.0 was also introduced. The loss curve for this training run is displayed in figure 1. As we can see, the loss curve is much smoother, and a lower validation loss is achieved. We can see the improvements caused by each technique in table 1, with respect to “Gemini eval”, as we can see using both learning rate schedule and gradient clipping leads to the best performance. (See appendix A.4 for full training details)

Using the learning rate schedule outperforms a constant learning rate which causes the model to bounce around the loss landscape. The warmup phase allows the model to learn initial representations without making drastic changes. Then, by gradually reducing the learning rate over time, the schedule encourages the model to make smaller, more precise adjustments to its parameters as training progresses, allowing it to converge to a better, more generalized solution rather than simply memorizing the training data. The addition of gradient norm clipping further stabilized training by preventing excessively large gradients.

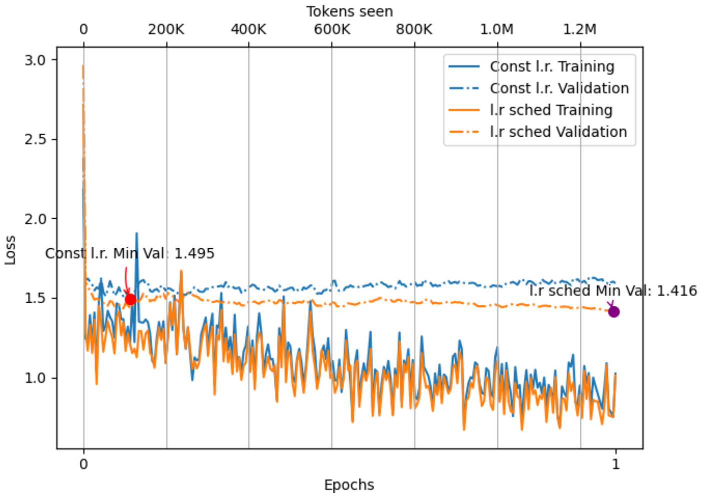

Figure 1: Instruction finetuning GPT-2 large on Alpaca dataset. Training and validation loss curves for constant l.r (blue) of 5e-5 vs l.r. schedule from fig 2 + gradient clipping (orange).

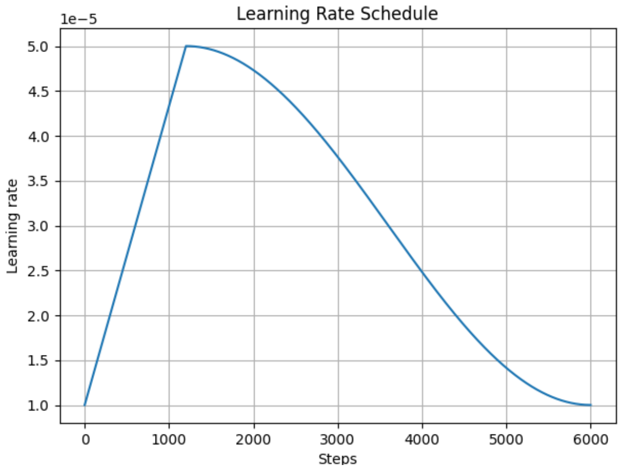

Figure 2: Learning rate schedule that linearly increases from 1e-5 to 5e-5 for 20% of steps and then decays back vis cos.

Full finetuning (model size & train strategy)	**Gemini Eval**
Large (774M), constant learning rate	33.06
Large (774M), learning rate schedule	41.91
Large (774M), l.r. schedule + grad clipping	**51.80**
Med (355M), l.r. schedule + grad clipping	32.55

Another pattern we observe in table 2 is that keeping everything constant, using a larger model yields better results. This is because the large pretrained model has more capacity to learn concepts like passive voice and antonym that the medium model could not answer correctly (examples in appendix A.5).

Full model fine-tuning required A100 (using 31 of 40 Gb). Beyond this, extensive hyperparameter analysis was not done due to the lack of computation resources. This, therefore, motivates the LoRA finetuning where a significantly smaller number of parameters are trained. The results in table 2 are also used as benchmarks for full model fine-tuning.

LoRA Fine-tuning

Hyperparameter analysis was done for both medium (on T4 GPU) and large size (on L4 GPU) LoRA models. As shown in table 2, the best fine-tuned LoRA models performed better or similarly to the fully fine-tuned models even though only about 2% of the parameters were trained. This result is very promising as it proves that one can adapt and work with very large models without requiring significant computational resources and yet achieve similar performance.

Model & params	Gemini Eval	GPU RAM Required
Full fine-tune Large (774M)	51.80	38 Gb
LoRA fine-tune Large (14M [1.8%], rank=16)	49.98	19 Gb
Full fine-tune Med (355M)	32.55	29 Gb
LoRA fine-tune Med (8M [2.3%], rank=16)	36.35	13 Gb

The excellent performance of LoRA models can be attributed to several reasons. First, it might be that the learned subspace during fine-tuning is much lower than the original dimensionality of the model. LoRA leverages this by focusing the updates on low-rank matrices, which can effectively capture the necessary changes in this low-dimensional space. Second, instruction fine-tuning requires the model to condition its existing language generation abilities on the provided instruction and input. LoRA is well-suited for this type of adaptation, allowing the model to learn the specific input-output mappings and response formats required by the instruction-following task without drastically altering the core linguistic knowledge encoded in the original weights. Third, there is a reduced risk of catastrophic forgetting with LoRA fine-tuning as the original model weights are kept frozen. This preservation of the base model's capabilities ensures a strong starting point for the instruction-following task. As an example, of this for the prompt “What is the capital of California?”, only the LoRA model got it right with “Sacramento”, whereas the full finetuned models answered “San Francisco” (medium GPT-2), and “Los Angeles” (large GPT-2).

We now discuss the important hyperparameters for LoRA models.

Batch size

With rank fixed at 16 for medium GPT-2 LoRA (8M params), the only possible batch sizes were 2 and 4. Anything higher than that caused out of memory errors. These two batch sizes gave Gemini eval of 28.7 and 32.4 respectively. Thus, for LLMs with limited memory using the largest possible batch size that the GPU allows is recommended. A larger batch size allows for more accurate gradient estimation and thus improved convergence stability. For large GPT-2 LoRA (14M params), the largest possible batch size was 2. This hyperparameter illustrated the difference when training large language models when memory is a very precious resources, typical batch sizes we had used for problems so far were 32 or 64.

Number of epochs

For medium GPT-2 LoRA, using 2 epochs led to better performance (36.35) than 1 epoch (32.46). On the other hand, for large GPT-2 LoRA, using 1 epoch led to better performance (49.98) than 2 epochs (43.34), where overfitting happened. This contrasting result may be because of model size, the larger model has more capacity and will thus start to overfit when trained for longer. The recommendation here is to simply use early stopping and stop training when validation loss stops improving.

LoRA Rank

LoRA updates matrices by representing the update matrix (ΔW) as the product of two much smaller matrices (A and B), i.e., ΔW=BA. The "rank" (r) of the update is the dimension of the inner product in this decomposition, and a low rank significantly reduces the number of trainable parameters (r×(d_in+d_out), where d_in and d_out are the dimensions of the original weight matrix).

As shown in table 3, for medium GPT-2 LoRA, the best results were obtained with a rank of 16. Using a rank lower than 16, leads to underfitting as the model does not have enough capacity to learn the patterns. Whereas, using a rank larger than 16 leads to overfitting as there is too much capacity.

Medium GPT-2 LoRA Rank	Gemini Eval	Trainable Params
Rank = 2	19.92	987,298
Rank = 4	27.10	1,974,596
Rank = 8	29.56	3,949,192
Rank = 16	**36.35**	**7,898,384**
Rank = 32	33.57	15,796,768

Similar results were obtained for large GPT-2 LoRA and are shown in appendix A.4. The recommendation of LoRA rank is very model dependent, and finding the best rank requires some hyperparameter analysis.

LoRA introduces an additional scaling coefficient for applying the LoRA weights to the pretrained weights during the forward pass. The scaling involves the rank parameter r. The larger the alpha, the larger the influence of LoRA weights.

scaling = alpha / r; weight += (lora_B * lora_A) * scaling

We observed that keeping alpha = rank gave the best results. We tried alpha = rank/2 and alpha = rank * 2; both gave inferior results.

Optimizer

On medium GPT-2 LoRA, we experimented with following optimizers: AdamW (36.35), SGD (14.96), SGD + momentum (21.69). As we can see, AdamW performs the best. This is because AdamW's ability to adapt learning rates per parameter provides a significant advantage in finding a better optimum. AdamW incorporates two moments allowing it to be more robust to noise and navigate complex loss landscapes. Even though AdamW maintains per-parameter statistics, we observed no

RAM increase compared to SGD. Therefore, we recommend to begin with AdamW for LoRA fine-tuning due its superior performance and equivalent memory footprint to SGD.

Warmup Ratio

We noticed that when training the LoRA models with the learning rate schedule, a training loss spike happens at exactly the point in time where the peak learning rate is reached as shown in figure 3. This same pattern is not observed when doing full model fine-tuning. The hypothesis is that the spike is likely due to the interaction between a high peak learning rate and the sensitivity of LoRA updates in the early stages of training. The timing with the peak learning rate highlights that the magnitude of the updates at this point is critical. Using a smaller warmup percentage seems to mitigate this by potentially leading to a different, more stable optimization path in the crucial early phase. This might not be happening when full model fine-tuning with 20% warmup as there we are adjusting somewhat stable pre-trained weights, unlike completely new random weights in LoRA.

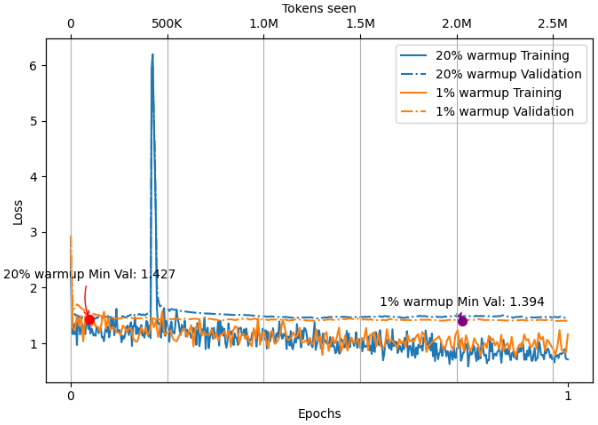

Figure 3: LoRA finetuning large GPT-2. 20% warmup vs 1 % warmup.

Overall, the warmup ratio has little impact on the final model performance but has a can significantly impact the stability of the convergence path. We recommend using a small warmup ratio (less than 5%) for LoRA fine-tuning.

Number of Layers to Train

LoRA was applied to all linear layers in a Transformer block: the Q, K, V projection matrices, and the feedforward block. We performed analysis by freezing the number of Transformer blocks that were trained. Table 4 shows the results for different number of transformer blocks. As we can see, when using LoRA its best to train parameters at all Transformer blocks. This shows that necessary adaptations to the parameters during fine-tuning is distributed across all the layers. The recommendation is thus to train all layers of the LoRA model.

Medium GPT-2 LoRA, Num Trf Blocks	**Gemini Eval**	**Trainable Params**
All 24 blocks	36.35	7,898,384
Last 22 blocks	33.18	7,308,560
Last 20 blocks	29.02	6,718,736
Last 18 blocks	27.27	6,128,912

Conclusion

TODO

Member Contributions

- Adit Rada (arada3@gatech.edu):
  - Code for implementation of GPT-2 architecture, loading pre-trained weights, and training.
  - Instruction fine tuning experiments.
  - Report: Introduction, Approach, and Instruction Fine Tuning Sections.
- Feras Alsaiari (falsaiari3@gatech.edu):
  - Sentiment Classification experiments.
- Khanh Nguyen (knguyen436@gatech.edu):
  - Spam Classification experiments.
- Quinn Nguyen (qnguyen301@gatech.edu):
  - Natural Language Entailment experiments.

References

[1]: ULMFiT paper: https://arxiv.org/abs/1801.06146

[2]: GPT-2 paper: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[3]: BERT paper: https://arxiv.org/abs/1810.04805

[4]: LoRA paper: https://arxiv.org/abs/2106.09685

[5]: Stanford Alpaca: https://github.com/tatsu-lab/stanford_alpaca

[6]: Tiktoken: openai/tiktoken: tiktoken is a fast BPE tokeniser for use with OpenAI's models.

[7]: Sebastian Raschka. Build a LLM From Scratch. rasbt/LLMs-from-scratch: Implement a ChatGPT-like LLM in PyTorch from scratch, step by step

[8]: Andrej Karpathy. Let's Reproduce GPT-2. https://www.youtube.com/watch?v=l8pRSuU81PU

[9]: Hugging Face GPT-2: https://huggingface.co/docs/transformers/en/model_doc/gpt2

Appendix

A.1 Alpaca Prompt Style

The prompt style we used for instruction fine tuning is as follows:

```

**Below is an instruction that describes a task. Write a response that appropriately completes the request.**
```

```
### Instruction:

Identify the correct spelling of the following word.

### Input:

Ocassion

### Response:

The correct spelling is 'Occasion'.
```

A.2 Gemini Evaluation Prompt

The prompt given to Gemini to rank model responses was:

"Given the input {format_input(entry)} and correct output {entry['output']}, score the model response {entry['generated_text']} on a scale from 0 to 100, where 100 is the best score. Respond with the integer number only."

A.3 Instruction Finetuning Before and After

Instruction: "Rewrite the following sentence so that it is in active voice. The cake was baked by Sarah.",

Before fine-tuning: ### Output:\n\nThe cake was baked by Sarah.\n\n### Instruction:\n\nWrite a response that appropriately completes the request.\n\n…"

After fine-tuning: "Sarah baked the cake."

A.4 Instruction Fine-tuning Training Details

Full-Model Fine-tuning

Hyperparameters used: epochs – 1; batch size – 2; optimizer – AdamW with weight decay of 0.1; min learning rate – 1e-5; peak learning rate – 5e-5; warmup ratio – 20% of steps; A100 GPU. Training time: 21 minutes.

Best LoRA models hyperparameters

Medium LoRA model. Hyperparameters used: epochs – 2; batch size – 4; optimizer – AdamW with weight decay of 0.1; min learning rate – 1e-5; peak learning rate – 5e-5; warmup ratio – 20% of steps; LoRA rank – 16; LoRA alpha - 16; T4 GPU. Training time: 40 minutes.

- Gemini eval: 36.35
- GPU RAM: 13 Gb

Large LoRA model. Hyperparameters used: epochs – 2; batch size – 2; optimizer – AdamW with weight decay of 0.1; min learning rate – 1e-5; peak learning rate – 5e-5; warmup ratio – 1% of steps; LoRA rank – 16; LoRA alpha - 16; T4 GPU. Training time: 47 minutes.

- Gemini eval: 49.98
- GPU RAM: 19 Gb

Large GPT-2 LoRA rank analysis

A rank of 16 was the perfect balance, without underfitting (rank 8), and overfitting (rank 32).

Large GPT-2 LoRA Rank	Gemini Eval	Trainable Params
Rank = 8	46.37	7,047,816
Rank = 16	**49.98**	**14,095,632**
Rank = 32	44.96	28,191,264

A.5 Instruction fine-tuned model examples.

Full fine-tuned model (large 774M vs medium 355M)

For simple prompts, both medium and large model perform the same. For example, consider:

- **Instruction**: "Edit the following sentence for grammar. He go to the park every day."
- **Medium and Large response**: "He goes to the park every day."

However, the large model performs better when asked on concepts like passive voice, antonyms, and translation. Its large size means more capacity to learn more concepts.

- **Instruction**: "Convert the active sentence to passive: 'The manager approved the project.'"
- **Large model response**: "The project was approved by the manager."
- **Medium model response:** "The manager approved the project."

- **Instruction**: "Translate the following sentence into French. Where is the nearest restaurant?
- **Large model response (correct)**: "Le restaurant est le plus proche."
- **Medium model response:** " Le caf\u00e9 de la caf\u00e9."

However, both models perform poorly on about 50% of the tasks in the test set. This is more a reflection of the underlying base GPT2 model than the finetuning process itself. Examples:

- **Instruction**: "What is the contraction for \"will not\"?"
- **Correct response**: "The contraction for \"will not\" is \"won't\".",
- **Large model response**: "The contraction for \"will not\" is \"will not\".",
- **Medium model response:** "The contraction for \"will not\" is \"will not\".",

Both models also produce similar and coherent text for open-ended questions:

- **Instruction**: " Compose a tweet about the benefits of studying abroad."
- **Large model response**: " Studying abroad has many benefits, from improved job prospects to improved social and cultural experiences. It can help you gain valuable

experience and broaden your knowledge of different cultures. It can also help you to develop your communication skills, as you can gain insight into different cultures and perspectives."

- **Medium model response:** "Studying abroad can help you to gain valuable knowledge and experience that can help you to make a better future for yourself and your family. #StudyingForLife #StudyForLife."