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 adapt the model to new tasks by updating only a small subset of parameters or introducing a small number of new, trainable parameters.

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 as 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.

Instruction Fine Tuning

Sdf

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

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'.
```

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."