HOMEWORK 3
APPLYING AND ADAPTING LLMS [*]

10-423/10-623 GENERATIVE AI
http://423.mlcourse.org

OUT: Feb. 20, 2024
DUE: Feb. 29, 2024
TAs: Meghana, Samuel, Ritu

Instructions

- **Collaboration Policy**: Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
  - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template. Submissions can be handwritten, but must be clearly legible; otherwise, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
  - **Programming:** You will submit your code for programming questions to Gradescope. There is no autograder. We will examine your code by hand and may award marks for its submission.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Question	Points
In-Context Learning	14
Programming: LoRA for GPT-2	25
Code Upload	0
Collaboration Questions	2
Total:	41

[*]Compiled on Monday 4th March, 2024 at 03:59

1 In-Context Learning (14 points)

1.1. (2 points) Explain the relationship between in-context learning and chain-of-thought prompting.

Chain of thought prompting tries to improve model performance by adding intermediary ”thought” process steps. In-context learning is a technique where examples of a task and solution are provided to the model in a prompt and then also asking the model to solve a new version of the task.

1.2. (3 points) Write a prompt to that might help facilitate in-context learning for the following question:

The cost of electricity per kilowatt-hour increases by 5 cents. Last month, a family used 150 kilowatt-hours at the old rate. This month, they used 100 kilowatt-hours at the new rate. Altogether, their electricity bills for these two months amount to \$45. How much was the old rate per kilowatt-hour?

Question: The cost of water per gallon increases by 10 cents. Last month, Lora used 50 gallons of water at the old rate. This month, they used 200 gallons of water at the new rate. Altogether, their water bills for these two months amount to \$70. How much was the old rate per gallon?
Answer: \$0.20 per gallon of water

Question: The cost of electricity per kilowatt-hour increases by 5 cents. Last month, a family used 150 kilowatt-hours at the old rate. This month, they used 100 kilowatt-hours at the new rate. Altogether, their electricity bills for these two months amount to \$45. How much was the old rate per kilowatt-hour?
Answer:

1.3. (3 points) Modify your answer from the previous question to use chain-of-thought prompting.

Solve the following question step by step: The cost of water per gallon increases by 10 cents. Last month, Lora used 50 gallons of water at the old rate. This month, they used 200 gallons of water at the new rate. Altogether, their water bills for these two months amount to \$70. How much was the old rate per gallon?
Answer: Lets solve the problem step by step. Let the old price of water be represented by ”x”. Then the total water bill across two months should be $200(x + 0.10) + 50x = \$70.00$. Isolating x results in the equation $250x = \$70.00 - \$20.00 = \$50.00$. Then solving for x we get $x = \$0.20$. Hence the old rate is \$0.20 per gallon of water.

Solve the following question step by step: The cost of electricity per kilowatt-hour increases by 5 cents. Last month, a family used 150 kilowatt-hours at the old rate. This month, they used 100 kilowatt-hours at the new rate. Altogether, their electricity bills for these two months amount to \$45. How much was the old rate per kilowatt-hour?
Answer: Lets solve the problem step by step.

1.4. (3 points) Modify your answer from the previous question to use zero-shot chain-of-thought prompting.

> Solve the following question step by step: The cost of electricity per kilowatt-hour increases by 5 cents. Last month, a family used 150 kilowatt-hours at the old rate. This month, they used 100 kilowatt-hours at the new rate. Altogether, their electricity bills for these two months amount to $45. How much was the old rate per kilowatt-hour?
> Answer: Lets solve the problem step by step.

1.5. (2 points) Describe an advantage and a disadvantage of zero-shot chain-of-thought prompting as compared to chain-of-thought prompting.

> The advantage of zero-shot chain-of-thought prompting is that the prompt can be much shorter. The disadvantage is that the model won't have an example of what "chain of thought" is, so it may not reason out the steps as accurately resulting in worse performance.

1.6. (1 point) Meta learning refers to the process of learning how to learn. One use case of meta learning is determining adaptation rules that, given small amounts of data for new tasks, facilitate good performance. Describe a similarity and a difference between in-context learning and meta learning.

> The difference between meta learning and in-context learning is that meta learning modifies the parameters while in-context learning does not modify the parameters. The similarity between meta learning and in-context learning is that both provide "examples" of how to perform a particular task in the prompt.

2 Programming: LoRA for GPT-2 (25 points)

Introduction

For large pre-trained models, full fine-tuning, which retrains all model parameters, becomes less feasible, due to the increased training time and memory requirements. In this section, you will explore, and build from scratch, a parameter efficient fine-tuning (PEFT) method, **Lo**w **R**ank **A**daptation (LoRA), and apply it to a pre-trained GPT2 model.

Dataset

The dataset for this homework is the Rotten Tomatoes Dataset from HuggingFace. It is a balanced movie review dataset containing positive and negative labels denoting sentiment. This dataset will download automatically when you run train.py

Starter Code

The main structure of the files is organized as follows:

```
hw3/
   lora.py
   model.py
   dataloader.py
   train.py
   generate.py
   configs/
      finetune_params_config.py
   configurator.py
   requirements.txt
```

Here is what you will find in each file:

1. `lora.py`: Implement LoRA in this. Some starter code is provided to help guide you. We only implement LoRA in a linear layer. (20 lines of code)
2. `model.py`: The vanilla working transformer implementation from HW1 (i.e. without GQA and ROPE). Use your implemented LoRA in the attention layers (2 lines of code).
3. `dataloader.py`: A custom dataloader implemented for the rotten tomatoes dataset. You only have to implement the _add_instruction_finetuning method in this dataloader. (5-10 lines of code)
4. `train.py`: The script for training GPT. This file is long but your only requirement is to make your model lora-friendly (2 line of code, marked with a TODO). Note: This is only done if we are using a pretrained model to begin with.
5. `generate.py`: The script for generating text with your trained (or raw) GPT model. Since we are using a classification dataset, you are expected to implement the get_accuracy function. (10 lines of code)
6. `configs/finetune_params_config.py`: You can use default parameters from this config or change them here

7. `configurator.py`: Utility script for loading parameters from the command line. Overrides parameters in finetune_params_config.py
8. `requirements.txt`: A list of packages that need to be installed for this homework.

Flags

All the parameters printed in the config can be modified by passing flags to `train.py`. Table 1 and Table 2 and contains a list of flags you may find useful while implementing HW3. You can change other parameters as well in a similar manner.

Configuration Parameter	Example Flag Usage
init_from	`--init_from=gpt2-medium`
out_dir	`--out_dir=gpt_lora_default`
rank	`--rank=8`
alpha	`--alpha=32`
lr	`--lr=2e-4`
max_iters	`--max_iters=20`
wandb_run_name	`--wandb_run_name=gpt-lora-r-8-alpha-32`

Table 1: Useful flags for `train.py`

Configuration Parameter	Example Flag Usage
init_from	`--init_from="resume"`
out_dir	`--out_dir="gpt_lora_default"`
device	`--device="cuda"`
max_new_tokens	`--max_new_tokens=5`
temperature	`--temperature=1.0`
top_k	`--top_k=200`

Table 2: Useful flags for `generate.py`

There are more parameters available to modify(see train.py), but we don't expect that you will need to modify more than the ones mentioned above.

Command Line

We recommend conducting this homework on Colab. Colab provides a free T4 GPU for code execution, albeit with a time limitation that may result in slower training. In the event of GPU depletion on Colab, options include waiting for GPU recovery, switching Google accounts, or purchasing additional GPU resources.

```
python train.py
     config/finetune_params_config.py \
    --init_from=gpt2-medium \
    --out_dir="gpt-lora-default"
```

```
python generate.py
    --init_from=resume \
    --out_dir="your_saved_lora_model" \
```

Low-Rank Adaptation (LoRA) of LLMs

In this problem, you will implement Low-Rank Adaptation (LoRA), following the approach outlined in (Hu et al., 2021). Before you continue, we strongly recommend you to go through the paper and understand how LoRA works.
Models can continue to learn efficiently even when their parameters are projected onto a smaller subspace. Essentially, this means that the vast majority of the model's capabilities can be retained and modified through adjustments in a significantly reduced parameter space. This allows for us to inject trainable low-rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks.
For a pretrained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA constrains its update through a low-rank decomposition, expressed as follows:

$$W_0 + \Delta W = W_0 + BA,$$

where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, and the rank $r \ll \min(d, k)$. During the adaptation process, W_0 remains unchanged—frozen—to ensure the stability of the pre-trained knowledge, while A and B are updated, serving as the trainable parameters. Note that that we achieve this by setting `requires_grad = False` for all parameters except the matrices A and B.

We then apply both W_0 and the adjustment $\Delta W = BA$ to the same input x, with their outputs being summed coordinate-wise, resulting in the modified forward pass:

$$h = W_0x + \Delta W x = W_0x + BAx$$

As depicted in Figure 1, our initial conditions for training involve setting A with a random Gaussian distribution and B to zero, making $\Delta W = BA$ start from zero. To integrate these updates effectively, remember to scale $\Delta W x$ by α/r, with α acting as a constant relative to r. This approach simplifies the optimization process, akin to adjusting the learning rate in Adam, and eliminates the need for hyperparameter retuning as r varies. You can start with setting α to the initial value of r you explore, and experiment with different scaling factors(α/r) by adjusting α and r accordingly. Thus, the scaled LoRA forward pass you should implement is:

$$h = W_0x + \frac{\alpha}{r}\Delta W x = W_0x + \frac{\alpha}{r}BAx$$

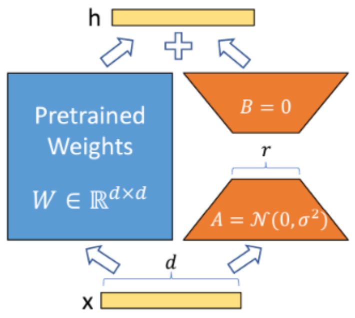

h
Pretrained Weights
$W \in \mathbb{R}^{d\times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Figure 1: Low-Rank Adaptation (LoRA) applied to a Transformer model.

Instruction FineTuning

As you must have seen in Homework 1 with the Shakespeare dataset, when given a text, GPT2 (or any LLM for that matter) generates more text to complete the given text. But for a task like text classification, how do you get the model to generate the labels you want for the given context?
Enter Instruction Fine Tuning. Instruction tuning is a specialized form of fine-tuning in which a model is trained using instruction-output pairs. It helps bridge the gap between the next-word prediction objective of LLMs and the our objective of having LLMs adhere to human instructions.
For the purpose of this homework, you can do this by prepending the text in each sample in the Rotten Tomatoes dataset with an instruction prompt template and then appending it with the actual label. This modified text is what you will train the model with. You will do this in the `_add_instruction_finetuning(self, rec)` function in `dataloader.py`. You can experiment with different instruction templates and ways to respresent the labels.

Implementation

Note: In the original paper, LoRA has been implemented only in the attention layers, specifically for the query and value matrices. In this homework you will implement LoRA on the query, key and value matrices. Note that you should also implement LoRA for the output projection in the attention layer.
The LoRA Linear Layer:

- In `lora.py`, implement these modifications in the `LoRALinear` class. This includes:
  - **__init__**: Initialize inherited nn.Linear class, LoRA parameters, and matrices A and B if the LoRA rank is greater than 0.
  - **reset_parameters**: Reinitialize weights of the inherited linear layer and LoRA matrices A and B. A is typically initialized with `kaiming_uniform_` and B is initialized with zeroes according to the paper.
  - **forward**: Implement the forward pass of the layer, including the application of LoRA modifications and dropout if applicable.
  - **train**: Override to ensure LoRA matrices are demerged and set to training mode.
  - **eval**: Override to ensure that LoRA matrices are merged with the actual model weight metrices and set to evaluation mode.
- **mark_only_lora_as_trainable**: A utility function to set only LoRA matrices as trainable parameters for a model.

LoRA for Transformer LMs:

- Apply your above implemented LoRALinear layer to the attention layers within your transformer model. This is marked with TODOs in `model.py`.

Instruction Fine-Tuning Method

- You are only required to implement `_add_instruction_finetuning` method in `CustomDataLoader`.
  - **Method:** `_add_instruction_finetuning(self, rec)`
    - **Parameters:**

        · `rec` (dict): A dataset record with ”text” and ”label” fields.

      * **Functionality:** Modifies the record by adding an `"instr_tuned_text"` field. This field integrates instructional cues into the original text to guide model training. Optionally convert labels to more intuitive formats (e.g., from numeric to textual labels such as positive/negative)

Training:

- Now that you have made your GPT model lora friendly, modify `train.py` to enable training with the LoRA-enhanced model. Ensure the model is made LoRA-friendly as indicated by the relevant TODO.

Accuracy Evaluation Method

- In `generate.py` you must implement the method `get_accuracy` designed to evaluate the model's performance on the test dataset (which will be passed in).
  - **Method:** `get_accuracy(self)`
    - **Functionality:** Iterates through a dataset to calculate the model's accuracy by comparing generated text against expected labels.
    - **Details:** Small models like GPT2 may not easily generate EOS token (especially for small r). Acknowledging these limitations, one simple hack in our case(where training labels are categorical) is to simply check if these labels exists in the first few characters of the generated text.

Hints

1. While implementing your code, you may find it help to adjust the model, e.g. 'gpt' is the smallest, but you will need at least 'gpt-medium' to see decent results from fine-tuning with LORA.
2. When trying different variations (across r, alpha, etc) it is recommended you use the `--out_dir` parameter so you can save the different models you create.
3. If you are facing CUDA BLOCKING errors, run with CPU device instead of CUDA on Colab to isolate errors better. Switch to CUDA for the actual training though.
4. Make sure that you account for garbage generations in your accuracy calculation. For example, for a given sample, if the model predicts something other than the specified labels(for eg, positive/negative) you should not omit it when calculating accuracy.

LoRA Implementation and Training

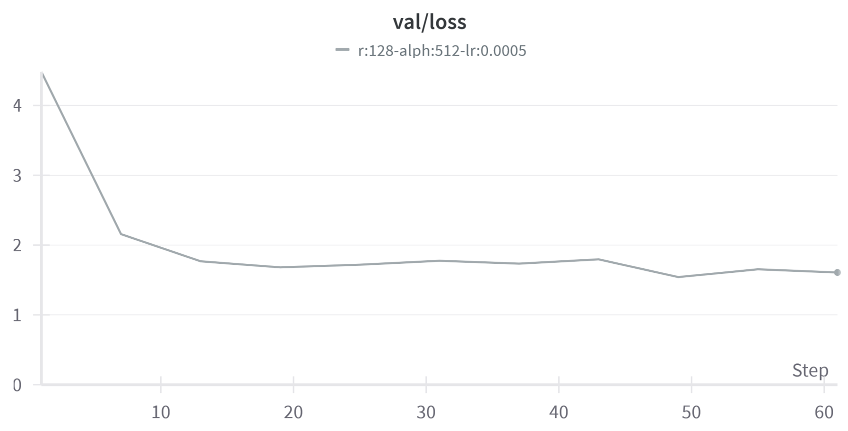
Note: For all the empirical questions report results using gpt2-medium. If not specified, return results using default parameters (i.e with $r = 128$, $\alpha = 512$ and learning rate = `5e-4`)

2.1. (2 points) Does training with LoRA add inference latency (i.e. are more parameters being learned that would add to inference time)? Explain

Training with LoRA does not add inference latency because the LoRA parameters are merged with the linear layer weights when performing inference. This is a single time process because $W_0x + BAx = (W_0 + BA)x$.

2.2. (4 points) Plot your validation loss curve for LoRA for the default configuration.

[Expected runtime on Colab T4: 1-2 minutes]



2.3. (2 points) What percentage of parameters are fine-tuned with when you set $r = 128$ and $\alpha = 512$?

Number of parameters with $r = 128$ and $\alpha = 512$: 372.65M. Number of parameters with $r = 0$ is 353.77M. Thus the number of parameters fine tuned is 372.65M-353.77M = 18.88M which is $18.88/372.65 = 0.0506 = 5.06\%$ of the paramters when $r = 128$ and $\alpha = 512$.

2.4. (2 points) What string did you use as `INSTRUCTION_TEMPLATE` for instruction fine-tuning?

"Guess if the review tone is positive or negative. Here is the review:{text}. Is the review's tone positive or negative? The review tone is {label}."

Inference and Evaluation with LoRA

2.5. (1 point) What is the accuracy of your model without any fine-tuning? (Hint: you can run this directly using python generate.py –init-from=”gpt2-medium”)

The accuracy of the model without any fine-tuning was 28%.

2.6. (4 points) What is the test accuracy of your model with LORA fine-tuning across $r \in \{16, 32, 128\}$? What is the test accuracy of fine-tuning without LoRA? (Hint: Make sure to also modify alpha to ensure a constant scaling factor of 4).[1]

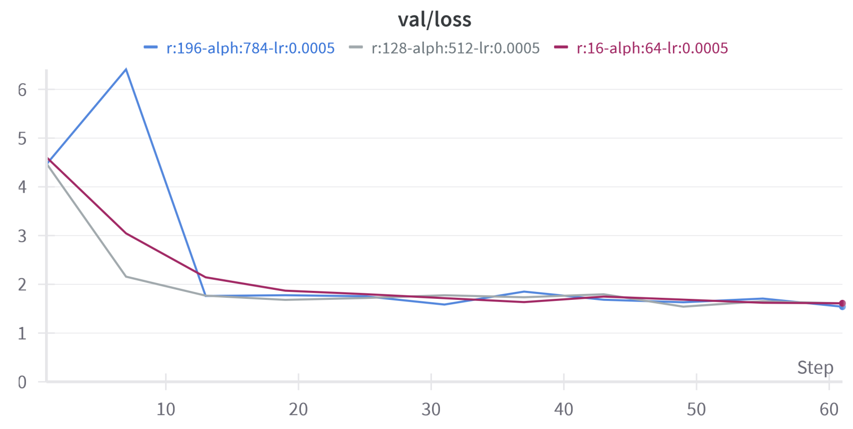
method	r	alpha	accuracy
LoRA	16	64	56%
LoRA	128	512	75%
LoRA	196	784	57%
Fine Tuning	–	–	79%

2.7. (3 points) How does your LoRA model's performance compare to fine-tuning without LoRA? Also, how does the value of r affect performance? Briefly discuss.

The LoRA model performs worse than full fine-tuning without LoRA. This is the expected result because full fine-tuning process can modify all of the parameters, hence it should be able to learn the task better than LoRA.
Increasing r appears to increase the ability for LoRA fine-tuning to adjust to the prompt. This makes sense since the rank of the LoRA matrices is r. So increasing r allows for the fine-tuning to capture more dimensions of the data. However, it appears that having too large a value of r can cause the model performance to decrease as the accuracy at $r = 196$ is lower than $r = 128$.

[1]The LoRA paper indicates that we should be able to reach (and perhaps surpass) the accuracy of full fine-tuning. You are encouraged to play around with hyperparameters (e.g. learning rate) to try to accomplish this, but this is not required.

2.8. (4 points) Plot wandb validation loss curves for LoRA with $r \in \{16, 128, 196\}$.



2.9. (1 point) Is there anything unexpected about the shape of the validation loss when $r = 196$? If yes, explain what is unexpected. If no, describe why it appears typical.

The validation loss seems to be normal for the most part. The expected result is that loss decreases faster for larger values of r which is what was observed. This was observed with $r = 16$ and $r = 128$.
The one surprising thing is that loss behavior for $r = 196$ spikes up towards the start of the training. This is somewhat strange since the other loss curves (for $r = 16$ and $r = 128$) do not have any spikey behavior. One reason this may have happened is that when r is too large, the information gained during the fine-tuning processes may dominate the pretrained parameters causing a decrease in performance.

2.10. (2 points) Changing both the learning rate and α may be redundant. Why?

The learning rate scales the size of a step the learner takes when updating gradients. The α parameter scales the influence of the LoRA term. Since only the LoRA weights are updated during fine-tuning, α and learning rate scale the gradient in similar ways during back propagation. Thus changing both the learning rate and α may be redundant as a similar effect could be achieved by only adjusting one of the hyperparameters.

3 Code Upload (0 points)

3.1. (0 points) Did you upload your code to the appropriate programming slot on Gradescope?
Hint: The correct answer is 'yes'.

[✓ Yes]

○ No

For this homework, you should upload all the code files that contain your new and/or changed code. Files of type `.py` and `.ipynb` are both fine.

4 Collaboration Questions (2 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

4.1. (1 point) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

I collaborated with Michael Leone and Gabriel Fonseca. They helped with prompt engineering for the instruction fine tuning.

4.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.

No