HOMEWORK 3
APPLYING AND ADAPTING LLMS *

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

OUT: Oct. 08, 2024
DUE: Oct. 23, 2024
TAs: Ketan, Afreen, Shreya

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. 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
LaTeX Template Alignment	0
In-Context Learning	14
Parameter Efficient Fine-Tuning	10
Direct Preference Optimization	15
Programming: LoRA for GPT-2	25
Code Upload	0
Collaboration Questions	2
Total:	66

*Compiled on Wednesday 9th October, 2024 at 10:13

1 LaTeX Template Alignment (0 points)

1.1. (0 points) **Select one:** Did you use LaTeX for the entire written portion of this homework?

- ◯ Yes
- ◯ No

1.2. (0 points) **Select one:** I have ensured that my final submission is aligned with the original template given to me in the handout file and that I haven't deleted or resized any items or made any other modifications which will result in a misaligned template. I understand that incorrectly responding yes to this question will result in a penalty equivalent to 2% of the points on this assignment.
Note: Failing to answer this question will not exempt you from the 2% misalignment penalty.

- ◯ Yes

2 In-Context Learning (14 points)

2.1. (2 points) Explain the relationship between in-context learning and chain-of-thought prompting.

2.2. (3 points) Write a prompt 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?

2.3. (3 points) Modify your answer from the previous question to use chain-of-thought prompting.

2.4. (3 points) Modify your answer from the previous question to use zero-shot chain-of-thought prompting.

2.5. (2 points) Describe an advantage and a disadvantage of zero-shot chain-of-thought prompting as compared to chain-of-thought prompting.

2.6. (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.

3 Parameter Efficient Fine-Tuning (10 points)

3.1. Suppose you are building a simple feed-forward neural network consisting of L layers. Each layer is fully connected with sigmoid activations, $\sigma(\cdot)$, applied elementwise. The D_0 input features are $\mathbf{z}_0$. The lth hidden layer $\mathbf{z}_l$ has D_l hidden units. The output layer $\mathbf{z}_L$ has only one unit $D_L = 1$. The model architecture is defined as follows for each, $\mathbf{z}_l \in \mathbb{R}^{D_l}$:

$$\mathbf{z}_l = \sigma(\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l), \quad \forall l \in \{1, \ldots, L\}$$

where $\mathbf{W}_l$ and $\mathbf{b}_l$ are the parameters of the model.

In the questions below, assume that $L = 10$ and $D_l = 2^{L-l}$ for all $l \in \{1, \ldots, L\}$.

3.1.a. (1 point) **Numerical answer:** How many hidden units are in this model? Your answer must be an integer.

3.1.b. (1 point) **Numerical answer:** How many parameters are in this model? Your answer must be an integer.

3.1.c. (1 point) **Numerical answer:** Now suppose we instead do parameter efficient fine-tuning in a style similar to BitFit (Ben-Zaken et al., 2021). Specifically, we leave the architecture unchanged and we fine-tune only the intercept terms, $\mathbf{b}_l, \forall l \in \{1, \ldots, L\}$, keeping all other parameters fixed. What percentage of the total parameters are fine-tuned in this setting? Your answer must be a percentage. (Report percent with two decimal places.)

3.1.d. (2 points) **Numerical answer:** Suppose we instead inject a bottleneck adapter module (of the variety introduced by Houlsby et al. (2019)) after each layer:

$$\mathbf{z}_l = \sigma(\mathbf{W}_l \mathbf{a}_{l-1} + \mathbf{b}_l), \quad \forall l \in \{1, \ldots, L\}$$
$$\mathbf{a}_l = \text{adapter}(\mathbf{z}_l, \mathbf{V}_l), \quad \forall l \in \{1, \ldots, L\}$$

where $\mathbf{a}_0 = \mathbf{z}_0$ and $\mathbf{a}_l \in \mathbb{R}^{D_l}$, $\mathbf{V}_l = [\mathbf{V}_{l,\text{down}}, \mathbf{v}_{l,\text{down}}, \mathbf{V}_{l,\text{up}}, \mathbf{v}_{l,\text{up}}]$ is a collection of all the adapter parameters for the lth adapter layer, and:

$$\text{adapter}(\mathbf{a}, \mathbf{V}) = \mathbf{a} + (\mathbf{V}_{\text{up}} \tilde{\sigma}(\mathbf{V}_{\text{down}} \mathbf{a} + \mathbf{v}_{\text{down}}) + \mathbf{v}_{\text{up}})$$

We follow the original paper and define $\tilde{\sigma}(\cdot) = \text{GELU}(\cdot)$ as the nonlinearity—GELU has no parameters. Assume the rank of the adapter is 4, i.e. its hidden layer after down-projection has 4 units. We tune only the adapter parameters. What percentage of the total parameters are fine-tuned in this setting? Your answer must be a percentage. (Report percent with two decimal places.)

3.2. In Prefix Tuning (Li & Liang, 20), we augment each attention head with a prefix, whose parameters are fine tuned while the rest of the model remains fixed. If $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are the standard query, key, and value matrices in attention, the prefix tuning does the following attention computation:

$$\tilde{\mathbf{K}} = [\mathbf{P}_K; \mathbf{K}]$$
$$\tilde{\mathbf{V}} = [\mathbf{P}_V; \mathbf{V}]$$
$$\tilde{\mathbf{A}} = \text{softmax}(\mathbf{Q}\tilde{\mathbf{K}}^T/\sqrt{d})$$
$$\tilde{\mathbf{X}}' = \tilde{\mathbf{A}}\tilde{\mathbf{V}}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n\times d}$ and $\tilde{\mathbf{K}}, \tilde{\mathbf{V}} \in \mathbb{R}^{m\times d}$ with $m = n + p$ where p is the prefix length. $\mathbf{P}_K$ and $\mathbf{P}_V$ are the learned prefix parameters (typically encoded via a bottleneck network).

3.2.a. (2 points) **Derivation:** Write down how $\tilde{A}_{i,j}$ the (i, j)th entry of the attention weights is computed in prefix tuning. Your answer must be in terms of the $\exp(\cdot)$ function. (You should not refer to softmax$(\cdot)$).

3.2.b. (3 points) **Proof:** Conceptually, the key/value vectors $\tilde{\mathbf{K}}_i$ and $\tilde{\mathbf{V}}_i$ can be divided into *prefix* tokens, $i \in \{1, \ldots, p\}$, and *content* tokens, $i \in \{p+1, \ldots, p+n\}$.

Show that Prefix Tuning does not change the *relative* attention weight between content tokens. That is, show that for any $i, j, k \in \{p+1, \ldots, p+n\}$ we have:

$$\frac{\tilde{A}_{i,j}}{\tilde{A}_{i,k}} = \frac{A_{i,(j-p)}}{A_{i,(k-p)}}$$

where A_{ij} are the attention weights of regular attention without prefix tuning.

4 Direct Preference Optimization (15 points)

4.1. In this question, you will step through the derivation of the direct preference optimization (DPO) objective function. Recall that in DPO, we assume there exists a latent reward function $r^*(x, y)$ that returns a real-value score which represents how good some response y is given a prompt x. We further assume that this latent reward governs the probability that a human ranker prefers response y_w to another response y_l according to the formula

$$p\left(y_w \succ y_l \mid x\right) = \frac{\exp r^*\left(x, y_w\right)}{\exp r^*\left(x, y_w\right) + \exp r^*\left(x, y_l\right)} \tag{1}$$

where $y_w \succ y_l$ indicates that y_w is preferred over y_l.

Given a (parameterized) LLM that defines a conditional distribution over responses given a prompt, $\pi_\phi(y \mid x)$, the goal of DPO is to fine-tune the parameters, ϕ, such that the expected reward subject to a KL-divergence penalty is maximized:

$$\pi_\phi^*(y \mid x) = \underset{\pi_\phi(y|x)}{\text{argmax}}\, \mathbb{E}_{\pi_\phi(y|x)}\left[r^*(x, y) - \beta \log \frac{\pi_\phi(y \mid x)}{\pi_{\text{ref}}(y \mid x)}\right] \tag{2}$$

for some reference LLM π_{ref}.

4.1.a. (5 points) Show that the optimal LLM can be expressed as

$$\pi^*_\phi(y \mid x) = \frac{1}{Z}\pi_{\text{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r^*(x,y)\right) \tag{3}$$

for some normalizing constant Z that does not depend on y. **Hint:** first, manipulate the objective function in (2) to be of the form

$$\pi^*_\phi(y \mid x) = \underset{\pi_\phi(y|x)}{\text{argmin}}\, KL\left(\pi_\phi(y \mid x)||p(y \mid x)\right) + C \tag{4}$$

for some distribution $p(y|x)$ and constant term(s), C, that don't depend on y. From there, argue what distribution $\pi_\phi(y \mid x)$ optimizes the objective in that form.

4.1.b. (3 points) Using the result in (3), show that

$$p\left(y_w \succ y_l \mid x\right) = \sigma\left(\beta \log \frac{\pi_\phi^*(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\phi^*(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right) \tag{5}$$

by solving (3) for $r^*(x, y)$ and plugging it in to (1)

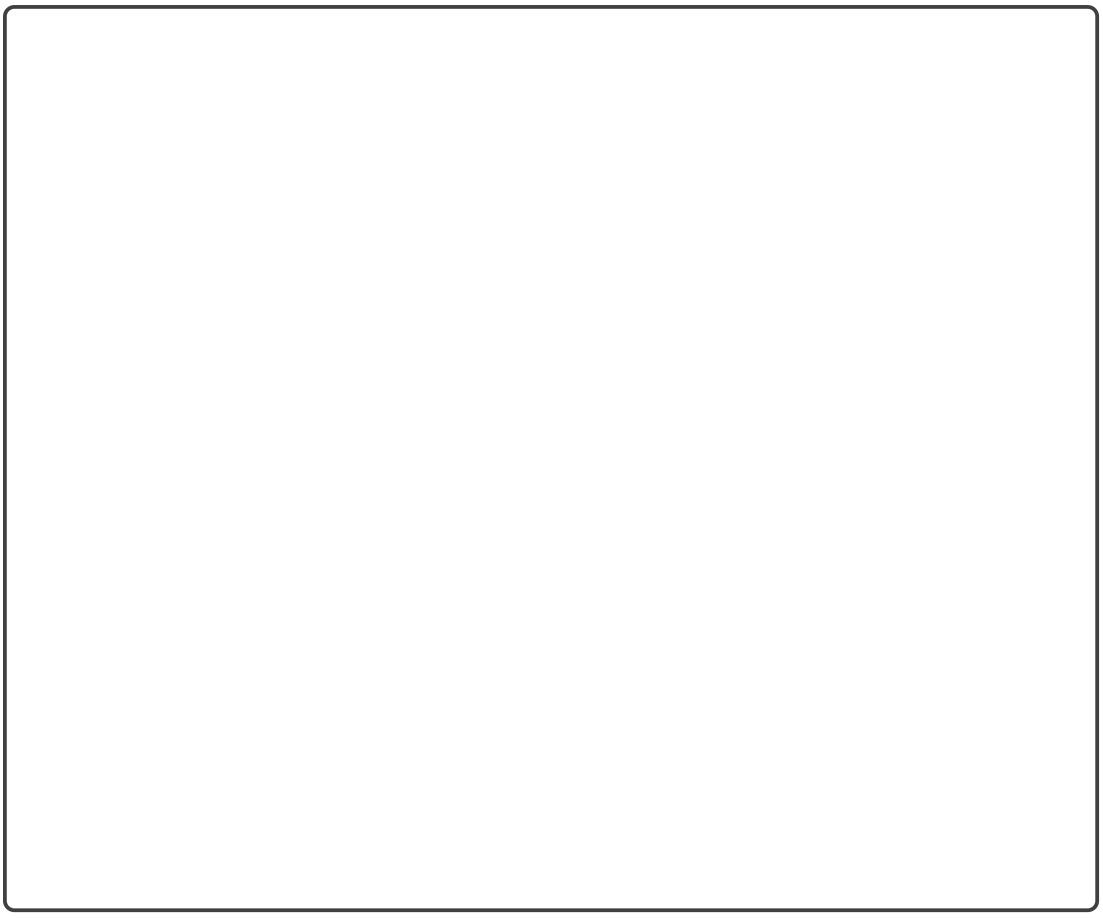
4.1.c. (3 points) The probability in (5) can be maximized by minimizing the objective function

$$\ell_{DPO}(\phi) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\delta(\phi)\right)\right]. \tag{6}$$

where $\delta(\phi) = \beta\log\frac{\pi_\phi(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta\log\frac{\pi_\phi(y_l|x)}{\pi_{\text{ref}}(y_l|x)}$.

Show that the gradient of ℓ_{DPO}, $\nabla_\phi\ell_{DPO}(\phi)$, is equal to

$$\nabla_\phi\ell_{DPO}(\phi) = \mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\sigma\left(-\delta(\phi)\right)\left(-\beta\frac{\nabla_\phi\pi_\phi(y_w \mid x)}{\pi_\phi(y_w \mid x)} + \beta\frac{\nabla_\phi\pi_\phi(y_l \mid x)}{\pi_\phi(y_l \mid x)}\right)\right] \tag{7}$$



4.1.d. The gradient in (7) can be used to optimize the parameters of π_ϕ directly i.e., without having to train an intermediate reward model! The two terms inside the expectation of (7) can be interpreted as having different effects on the actual updates made to π_ϕ.

4.1.d.i. (2 points) What *direction* does the second term, $-\beta\frac{\nabla_\phi \pi_\phi(y_w|x)}{\pi_\phi(y_w|x)} + \beta\frac{\nabla_\phi \pi_\phi(y_l|x)}{\pi_\phi(y_l|x)}$, encourage the parameters to move in? Recall that gradient descent moves in the *opposite* direction of the gradient. **Hint:** think about the effect of updating ϕ in this direction on the likelihoods $\pi_\phi(y_w \mid x)$ and $\pi_\phi(y_l \mid x)$.

4.1.d.ii. (2 points) Under what conditions is the *magnitude* of the first term, $\sigma(-\delta(\phi))$ large i.e., ≥ 0.5? Frame your answer in terms of the likelihoods $\pi_\phi(y_w \mid x)$ and $\pi_\phi(y_l \mid x)$.

5 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
    requirements.txt
    run_in_colab.ipynb
    wandb_api.json
```

Here is what you will find in each file:

1. `lora.py`: Implement LoRA in this. Some starter code is provided to guide you. Only implement LoRA in a linear layer. **UPLOAD to Gradescope**
2. `model.py`: The vanilla working transformer implementation from HW1 (i.e. without GQA and ROPE). Use your implemented LoRA in the attention layers. **UPLOAD to Gradescope**
3. `dataloader.py`: A custom dataloader implemented for the rotten tomatoes dataset. After running other experiments, customize the prompt. **UPLOAD to Gradescope**
4. `train.py`: The script for training GPT. This file is long but your only requirement is to make your model lora-friendly. Note: This is only done if we are using a pretrained model to begin with. **UPLOAD to Gradescope**
5. `generate.py`: The script for generating text with your trained (or raw) GPT model. Since we are using a classification dataset, convert text outputs from the LLM to integer labels. **UPLOAD to Gradescope**
6. `requirements.txt`: A list of packages that need to be installed for this homework.
7. `run_in_colab.ipynb`: Provides command lines to run your model in Google Colab.
8. `wandb_api.json`: Paste your WandB API key here. You don't have to upload this file.

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"`
device	`--device="cuda"`
rank	`--rank=128`
alpha	`--alpha=256`
lr	`--lr=2e-5`
dropout	`--dropout=0.05`
lora_dropout	`--lora_dropout=0.05`
max_iters	`--max_iters=80`
wandb_project	`--wandb_project="HW3_lora_finetune_handout"`

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=0.6`
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

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, purchasing additional GPU resources, switching to Kaggle, or switching to a cloud provider (such as GCP or AWS).

```
python train.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:

$$h = W_0x + \Delta Wx = W_0x + BAx$$

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 ΔWx 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:

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

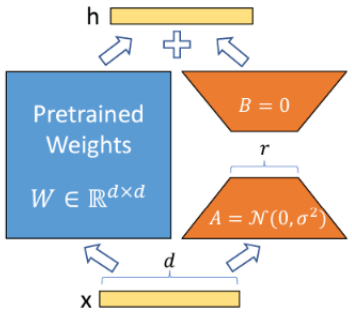


Figure 1: Low-Rank Adaptation (LoRA) applied to a Transformer model.

Instruction FineTuning

As 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. This happens in the `get_sentiment_prompt()` 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.

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

- All the methods in `CustomDataLoader` and `dataloader.py` are complete. However, you will return to this file at the end of the empirical section and modify the prompt in `get_sentiment_prompt(text, label)`.

- Take note of what `_add_instruction_finetuning(self, rec)` is doing. `rec` is a dataset record with "text" and "label" fields. The function modifues the record by adding an `"instr_tuned_text"` field. This field integrates instructional cues into the original text to guide model training. It also converts labels to a more intuitive format (e.g., from numeric to textual labels 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 `predict_labels` which iterates through a dataset, constructs the prompt for each example, and converts the response of the model from a text string to an integer label (1 for positive and 0 for negative).
- **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.
- Make sure that you account for garbage generations when converting from text labels generated by the model to integers. 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.

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` flag 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.

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 $r = 128$, $\alpha = 512$, lora_dropout $=$ `0.05`, dropout$=$ `0.0`, and learning_rate $=$ `2.5e-4`). Use the default prompt unless mentioned (5.11)

5.1. (2 points) Does training with LoRA add inference latency (i.e. are more parameters being learned that would add to inference time)? Explain.

5.2. (2 points) What percentage of parameters are fine-tuned with when you set $r = 128$ and $\alpha = 512$?

Inference and Evaluation with LoRA

5.3. (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"`) [Expected runtime on Colab T4: 2 minutes]

5.4. (4 points) What is the test accuracy with LoRA fine-tuning across $r \in \{16, 128, 196\}$? What is the test accuracy of full fine-tuning (dropout 0.05) i.e. without LoRA? In this question, we maintain a constant scaling factor of 4, i.e. $\alpha = 4r$. *Note: Be sure to report the test accuracy from the "Best Val Checkpoint" and* not *the "Last Iter Checkpoint".*

[Expected runtime on Colab T4: 25-30 minutes per experiment]

method	r	alpha	test accuracy (Best Val Checkpoint)
LoRA	16	64	
LoRA	128	512	
LoRA	196	784	
Full Fine Tuning (dropout=0.05)	0	–	

5.5. (4 points) Plot wandb validation loss curves for LoRA experiments with

$$(r, \alpha) \in \{(16, 64), (128, 512), (196, 784)\} \text{ and full fine-tuning } (r = 0).$$

5.6. (3 points) How does your fine-tuning with LoRA model's performance compare to full fine-tuning (without LoRA)? Also, how does the value of r affect performance? Briefly discuss (include comments on convergence analysis of validation loss).

5.7. (2 points) 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. Do you think it is useful to increase LoRA rank beyond this point $(r = 196)$ or should we tune it with different hyperparameters?

5.8. (2 points) Report the test accuracy and plot wandb validation loss curves for LoRA with $(r, \alpha) \in \{(16, 64), (16, 256)\}$ and full fine-tuning ($r = 0$).

[Expected runtime on Colab T4: 25-30 minutes for the new setting]

method	r	alpha	accuracy
LoRA	16	64	
LoRA	16	256	
Full Fine Tuning (dropout=0.05)	0	–	

5.9. (1 point) In your results from the previous question, how did increasing α while keeping the rank r unchanged affect the performance of the model? Why might this be the case?

5.10. (1 point) Changing both the learning rate and α may be redundant. Why?

5.11. (1 point) Try out a different prompt template during training and generation by modifying `get_sentiment_prompt()` in `dataloader.py`. Report the text of your new instruction template here. You can do so simply by copy/pasting in the python code. Comment on what motivated your change to the prompt. (Note that you do *not* need to find one that performs better.)

Code Snippet

Comments

5.12. (2 points) For the default setting $r = 128, \alpha = 512$, report the test accuracy with the original prompt template and with your new prompt template. (Note that you do *not* need to find one that performs better.)

[Expected runtime on Colab T4: 25-30 minutes for the new setting]

6 Code Upload (0 points)

6.1. (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.

7 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.

7.1. (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.

7.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.