CIS 6200 Advanced Topics in Deep Learning, Spring 2024

Homework 4
Due: Monday, February 19, 11:59pm
Submit to Gradescope

Task 1: Form Project Groups!

Form a team of 2-4 people (you are encouraged to use the teammate browsing resources available on Ed). Submit your team information through Google Form.

Each team only needs to submit **ONCE**.

Task 2: Fine-Tuning

Experiment and evaluate various fine-tuning methods with HuggingFace PEFT on image classification task

Learning Objectives

After completing this assignment, you will:

- be able to use different fine-tuning methods and recognize their relative costs and benefits

Deliverables

Task 2 is an **individual** assignment for both the written and coding portions.

1. **A PDF with your name**

   Copy and edit this google doc to enter your names and answer the questions below.

2. **hw4.ipynb file with the functional code**

   Complete the coding assignment in the Jupyter Notebook and upload the file to Gradescope.

Homework Submission Instructions

Written Homeworks

All written homework must be submitted as a PDF to Gradescope. **Handwritten assignments (scanned or otherwise) will not be accepted.**

Coding Homeworks

All coding assignments will be done in Jupyter Notebooks. We will provide a .ipynb template for each assignment as well as function stubs for you to implement. You are free to use your own installation of Jupyter for the assignments, or Google Colab, which provides a Jupyter

Environment connected to Google Drive along with a hosted runtime containing a CPU and GPU.

Coding: Experiment with LoRA and last-layer fine-tuning with PEFT

1. With the provided notebook, load a pre-trained Vision-Transformer Model, and preprocess the dataset to fit the fine-tuning task. **Report**:
    - What metrics are we using for the evaluation? Can you think of any other options (no need to implement)?
    - What is the baseline accuracy on our task?
2. Implement **LoRA fine-tuning with PEFT frameworks** and **Report**:
    - What hyperparameters have you used for LoRA configuration? What are their meanings respectively?
    - How many parameters are there in the LoRA model?
    - How long does the fine-tuning take? What is the resulting accuracy?
    - **[Note]** Recommended LoRA rank = 8 (**optional**: feel free to try other values!)
3. Instead of LoRA, **directly fine-tune the last layer** (with all previous layers fixed). **Report**:
    - How many parameters are involved in the fine-tuning?
    - How long does the fine-tuning take? What is the resulting accuracy?

Discussion question:

4. Calculate theoretically the **number of parameters** for the fine-tuning task in 2 and 3. Does it match with your implementation?
    - **[Note]** Information about model architecture is available at HuggingFace
5. Compare the two fine-tuning methods on their accuracy, efficiency, and reliability. What are the advantages and disadvantages of each?

Answers:-

Coding:

1.a) From the evaluate library, we use the accuracy metric in this coding assignment. It computes the proportion of correct predictions among the total number of cases processed. Other options include exact match and mean IOU.

b) The baseline accuracy of this task is around 0.01, that is, very close to zero.

2.a) The hyperparameters are-r,lora_alpha,lora_dropout,bias,target_modules,modules_to_save.

“R” is the parameter that specifies the rank of the adaptation matrices in LoRA. A lower rank means that the adaptation will be more parameter-efficient but might capture less complexity. A higher rank allows for a more complex adaptation at the cost of more parameters.

“lora_alpha” is the parameter that controls the scale of the adaptation applied by LoRA. It is a scaling factor for the low-rank matrices that are used to adapt the pre-trained model's weights. A higher “lora_alpha” means a stronger adaptation effect.

“lora_droput” is the parameter that specifies the dropout rate to be applied to the LoRA adaptation parameters.

“bias” is the parameter that indicates whether and how bias terms should be adapted or used in the LoRA adaptation.

“target_modules” is the parameter that specifies which parts or modules of the pre-trained model should be adapted using LoRA.

“modules_to_save” is the parameter that specifies which modules of the adapted model should be saved.

b) Total Parameters:86248906
Percentage of trainable parameters=0.4319834503176191%
Number of trainable parameters:372581

c) Fine-tuning took a bit more than 5 minutes, to be precise, 330 seconds and the best accuracy was 0.964.

3.a)Total Parameters:85876325
Percentage of trainable parameters=8.344023803999532%
Number of trainable parameters:7165541

b) The fine-tuning took around three and a half minutes and the best accuracy was 0.95.

Discussion questions:-

4. LoRA: There are 12 transformer layers. We are targeting the query and key values and the rank of the adaptation matrix is 8.

So, for every layer, we will have 2*(2*d*r) for the key and query values.
Since we have 12 layers, it would become 12*4*d*r=12*4*768*8.
This would become 294,912.

When we include the classifier head, (din*dout)+dout, we would get (768*101)+101=77669.

We would finally obtain 372,581 which matches with the one I found in the notebook.

Last layer finetuning: We fine-tune the last layer of the transformer and the classifier head.

Transformer: There are three types of values-query,key and value, so we will have 3*((768*768)+768) and for the bias term we will have 768*768+768.

There are two FFNs, one to increase it from 768 to 3072 and another to decrease it back to 768 from 3072.

So, that would be ((768*3072)+3072)+((768*3072)+768).

If we add them all up, I get 7,165,541 which matches with the code.

5. Accuracy: LoRA only has a slight edge in this case but would have a significant edge in other cases, which might justify its use in scenarios where maximum performance is critical. Efficiency: Last layer fine-tuning is faster and simpler, making it suitable for quicker iterations or when computational resources are limited.

Reliability: Both methods can be reliable, but their effectiveness can vary depending on the specific task and data. LoRA might offer better reliability across diverse tasks due to its more nuanced adaptation, but last layer fine-tuning can be very effective for tasks closely related to the original training data.

LoRA has a higher accuracy and can be used to adapt the whole model in an efficient manner and even though it may take more time to train than the last layer fine tuning, it is pretty good considering that multiple layers are being adapted. Disadvantages are listed in the advantages of last layer fine tuning, also it introduces new hyperparameters to tune which can be considered a disadvantage.

Last layer fine tuning is simple and straightforward and takes less time to train. It can be rapidly adapted to new tasks if the pre-trained features are robust and transferable across similar domains. Disadvantages: It has a lower flexibility and may risk overfitting as well. The accuracy would also be lower than LoRA.