Lab 4: The One Where You Learn to Specialize

Introduction

In this laboratory, you will explore parameter-efficient fine-tuning methods, specifically Low-Rank Adaptation (LoRA) and Prompt Tuning, by applying these techniques to an already pretrained model. This laboratory practice is designed to give you practical experience with PEFT techniques, allowing you to efficiently fine-tune a pre-trained model with minimal computational resources. You will compare the performance of full fine-tuning against these parameter-efficient methods to see how they impact both qualitative and quantitative results.

Throughout this practice, you will:

- Apply LoRA to reduce the number of trainable parameters in specific layers of the model.
- Implement prompt tuning, where virtual tokens are added to the input without modifying the model's internal weights.
- Fine-tune the model using the `DialogSum` dataset for text summarization.
- Compare the effectiveness of full fine-tuning, LoRA, and prompt tuning in terms of ROUGE scores and qualitative results.

By the end of this lab, you will have a deeper understanding of parameter-efficient fine-tuning techniques and how they compare to full fine-tuning, as well as how they can be applied to improve the efficiency and effectiveness of NLP models.

Learning Outcomes

Upon completion of this laboratory exercise, you will be able to:

- Implement LoRA and Prompt-Tuning techniques to a pretrained model.
- Understand how different Parameter-Efficient Fine-Tuning (PEFT) techniques affect model performance.
- Understand how different PEFT techniques affect model retraining computational requirements.

Deliverables

1. `finetuning.py` - This file must implement the LoRA and Prompt-Tuning related modules.
2. `finetuning.ipynb` - This file must use a `flan-t5` model from Google to test and try the different PEFT methods. Play with it!

These files should be uploaded to Github maintaining the format handled in the laboratory practice.

Exercises

A word of advice

Due to version compatibility issues, for this lab, a new conda environment was created with Python 3.11.10 and the following package versions:

- `accelerate==1.0.1`
- `datasets==2.19.1`
- `evaluate==0.4.3`
- `huggingface==0.0.1`
- `huggingface_hub==0.24.6`
- `loralib==0.1.2`
- `numpy==1.26.4`
- `pandas==2.2.2`
- `peft==0.13.2`
- `rouge_score==0.1.2`
- `transformers==4.44.1`

If you find that your current environment misses some libraries or has mismatched versions, install the provided `requirements.txt` file with the command:

```
pip install -r requirements.txt
```

Mac users may also need to update the `protobuf` library with:

```
conda install -c conda-forge protobuf
```

For faster execution, consider running the Jupyter notebook in Google Colab.

Exercise 0: T5 or not T5

In this exercise, you will need to make sure that you are able to download a model from Google's `T5` models family. You have already done this step in the previous practice, but due to the packages upgrading process you might have broken something.

Therefore, check that you pass the `test_download_and_load_model()` function from `test_utils.py` file. In case you do not pass it, modify the `download_and_load_model()` function from `utils.py` file until you are able to download the model.

Exercise 1: ♪♪ LoRA, LoRA, LoRAAAA! ♪♪

In this exercise, you will complete the implementation of the `LoRA` (Low-Rank Adaptation) module and integrate it into a pre-trained model. This will allow you to modify the attention layers in a more parameter-efficient way. We have divided the task into two sub-exercises to guide you.

Exercise 1.1: Tuning your `LoRA` nk voice

In this part, you will complete the implementation of the `LoRA` class. The following steps must be followed:

- Initialize the low-rank matrices `A` and `B` for the LoRA module.
- Properly scale the LoRA output by the factor `alpha/r` .
- Ensure that the original layer's weights are optionally frozen during training.

Make sure to refer to the provided docstring for further clarification on the arguments and structure of the module. Once completed, your `LoRA` module should add a low-rank update to the original layer during the forward pass.

Exercise 1.2: Add Your Voice to the Chorus

Next, you will modify the `inject_lora_into_model()` function. This function injects LoRA modules into specific linear layers of the attention mechanism in the pre-trained model. You need to:

- Loop through the layers of the model and identify attention layers, such as the query, key, value, and output layers (`Q` , `K` , `V` , `O`).
- Replace these layers with the corresponding LoRA modules.

Your goal is to modify the model architecture to include the LoRA components while maintaining its overall functionality.

Exercise 2: Prompting me softly ♪♪

In this exercise, you will implement the Soft-Prompting technique and integrate it into a pre-trained model. Soft-prompting allows you to prepend trainable virtual tokens (soft prompts) to the original input embeddings, which enables task-specific fine-tuning without modifying the model's internal weights.

You will complete the code for the `SoftPromptEmbedding` module and use it to inject the soft prompts into a pre-trained model.

Exercise 3: `PEFT` ! This is so easy!

In this exercise, you will explore the Low-Rank Adaptation (LoRA) and Prompt Tuning techniques using the HuggingFace PEFT (Parameter-Efficient Fine-Tuning) library. These techniques allow you to efficiently fine-tune large language models with only a small fraction of the parameters, making them ideal for resource-constrained environments. You will experiment with both techniques to fine-tune the FLAN-T5 model on the DialogSum dataset. Make sure to **answer the questions at the end of the notebook!**

Follow the steps below to complete the tasks and answer the questions at the end of the notebook.

Exercise 3.1: Preprocessing and Tokenization

You will first preprocess the DialogSum dataset to create instruction-based prompts for training the model. Make sure to:

- Add an instruction like `Summarize the following conversation` to each prompt.
- Tokenize the dataset using the HuggingFace tokenizer.

Exercise 3.2: Full Fine-Tuning

In this part, you will perform full fine-tuning on the pre-trained FLAN-T5 model. This requires training all model parameters on the task of summarizing dialogues. After fine-tuning, evaluate the model's performance both qualitatively (comparing model outputs) and quantitatively (using the ROUGE metric).

Exercise 3.3: LoRA Fine-Tuning

Now, you will inject LoRA into the attention layers of the FLAN-T5 model and fine-tune only the LoRA parameters. Compare the performance of the LoRA-adapted model to the fully fine-tuned model using the ROUGE metric.

Exercise 2.4: Prompt Tuning

Finally, you will implement prompt tuning, an additive fine-tuning technique where trainable virtual tokens are added to the input. Fine-tune the model with prompt tuning and evaluate its performance against both LoRA and full fine-tuning.

Test your work!

Specific test functions have been developed to test your work. These tests are designed to automatically check the correctness of your implementations.