CS7.501: Advanced NLP — Assignment 3

Instructor: Manish Shrivastava

Deadline: October 28, 2024 — 23:59

1 General Instructions

1. You should implement the assignment in Python.
2. Ensure that the submitted assignment is your original work. Please do not copy any part from any source, including your friends, seniors, or the internet. If any such attempt is caught, serious actions, including an F grade in the course, are possible.
3. A single .zip file needs to be uploaded to the Courses Portal.
4. Your grade will depend on the correctness of answers and output. Due consideration will also be given to the clarity and details of your answers and the legibility and structure of your code.
5. Please start early to meet the deadline. Late submissions won't be evaluated.

2 Objective

In this assignment, you are required to implement and compare three fine-tuning methods on the summarization task using the GPT-2 small model: **Prompt Tuning**, **Traditional Fine-Tuning (Last Layers Only)**, and **LoRA (Low-Rank Adaptation)**. For Prompt Tuning, refer to https://aclanthology.org/2021.emnlp-main.243.pdf, and https://arxiv.org/pdf/2106.09685 for LoRA.

For LoRA you are **allowed** to use PEFT libraries for fine-tuning.

The specific objectives are as follows:

- Implement Prompt Tuning for the GPT-2 small model using soft prompts.
- Implement LoRA by adding low-rank matrices to the GPT-2 small model for parameter-efficient adaptation.

- Implement Traditional Fine-Tuning where only the last classifier layer(s) of GPT-2 are fine-tuned.
- Compare the performance of these three models on the summarization task.

You should evaluate the following metrics:

- **Evaluation loss**: Compare the evaluation loss across all three models.
- **ROUGE score**: Compare the ROUGE scores across all three models.
- **Bonus (Optional)**: Compare the time taken for training, the number of added parameters, and GPU compute for each model.

By completing this assignment, you will gain hands-on experience in implementing and evaluating three fine-tuning techniques for large language models.

3 Setup

- Use the transformers library from Hugging Face to load the GPT-2 small model and tokenizer.
- Ensure you have the necessary datasets for the summarization task.

4 Architecture

4.1 Soft Prompt Embedding Layer (Prompt Tuning)

Create an embedding layer for the soft prompts. This layer will have its own parameters separate from the GPT-2 model. The size of this embedding layer should be `[num_prompts, embedding_size]`, where `num_prompts` is the number of tokens in your soft prompt and `embedding_size` is 768 for the GPT-2 small model.

4.2 LoRA Architecture

In the LoRA approach, low-rank matrices are added to specific weight matrices in the GPT-2 model. These matrices allow fine-tuning without modifying the original model parameters. The GPT-2 model remains frozen except for the added low-rank layers.

The size of the low-rank matrices should be `[input_dim, rank]` and `[rank, output_dim]`, where `rank` is a hyperparameter determining the compression factor, typically much smaller than the input/output dimensions. These matrices are updated during training, while the rest of the model remains frozen.

However, as mentioned above, you can standard libraries for this.

4.3 Traditional Fine-Tuning (Last Layer Only)

In Traditional Fine-Tuning, instead of fine-tuning the entire GPT-2 model, only the last linear layer (the language modeling head) is updated. The rest of the model remains frozen. This reduces computational requirements while still allowing the model to adapt to the new task.

The following lines of code demonstrate how to freeze all layers except the last LM head layer:

```
# Freeze all layers
for param in model.parameters():
    param.requires_grad = False

# Unfreeze the language modeling head (LM head)
for param in model.lm_head.parameters():
    param.requires_grad = True
```

This approach allows fine-tuning with minimal additional parameters and lower computational overhead compared to full fine-tuning.

4.4 Fine-tuning Architecture and Hyperparameters

For the fine-tuning process, consider the following architecture and hyperparameters:

- **Batch Size**: Can be anything. If facing memory constraints, consider using gradient accumulation as a strategy to effectively increase the batch size without requiring additional memory.
- **Epochs**: Train for up to 10 epochs for the summarization task, but consider early stopping based on validation loss to prevent overfitting.
- **Gradient Clipping**: Clip gradients with a norm of 1.0 to prevent exploding gradients.

5 Fine-tuning on the Summarization Task

- **Dataset**: Use the CNN/Daily Mail dataset. https://www.kaggle.com/datasets/gowrishankarp/newspaper-text-summarization-cnn-dailymail/code
- **Soft Prompt**: Initialize with tokens like [SUMMARIZE].
- **Note**: Use 10% of the dataset, which is approximately 30,000 samples in total; split as 21,000 for training, 6,000 for validation, and 3,000 for testing.

6 Training

During training, apply the following methods:

- **Prompt Tuning**: Prepend the soft prompt tokens to your input sequences and backpropagate only through the soft prompt embeddings, keeping the GPT-2 parameters frozen.
- **LoRA Fine-Tuning**: Implement LoRA by adding low-rank matrices to adapt the GPT-2 parameters efficiently, as described in the reference code. Keep the original model parameters frozen except for the LoRA layers.
- **Traditional Fine-Tuning (Last Layers Only)**: Fine-tune only the last few layers (such as the pre-classifier and classifier layers) of the GPT-2 model while freezing the rest of the model.

7 Evaluation

Evaluate the performance of your fine-tuned models on the summarization task using the evaluation loss. Optionally, compare the time, number of added parameters, and GPU compute for each model.

8 Questions & Grading

[Total marks: 100, Bonus marks: 25]

8.1 Theory [20]

1. **Concept of Soft Prompts**: How does the introduction of "soft prompts" address the limitations of discrete text prompts in large language models? Why might soft prompts be considered a more flexible and efficient approach for task-specific conditioning?
2. **Scaling and Efficiency in Prompt Tuning**: How does the efficiency of prompt tuning relate to the scale of the language model? Discuss the implications of this relationship for future developments in large-scale language models and their adaptability to specific tasks.
3. **Understanding LoRA**: What are the key principles behind Low-Rank Adaptation (LoRA) in fine-tuning large language models? How does LoRA improve upon traditional fine-tuning methods regarding efficiency and performance?
4. **Theoretical Implications of LoRA**: Discuss the theoretical implications of introducing low-rank adaptations to the parameter space of large language models. How does this affect the expressiveness and generalization capabilities of the model compared to standard fine-tuning?

8.2 Implementation & Training [70]

1. **Architecture [30 marks]**: Follow the architecture as described above. Implement the Soft Embedding layer correctly and modify the GPT-2 model to incorporate soft prompts, LoRA, and traditional fine-tuning (last layers only).

2. **Finetuning [35 marks]**: Fine-tune on the summarization task using the three methods: Prompt Tuning, LoRA, and Traditional Fine-Tuning (last layers only).

3. **Evaluation and Results [15 marks]**: Report relevant metrics, particularly the evaluation loss for each model. Optionally, include the bonus metrics (time, parameters, GPU compute) and put all results in your report.

8.3 Analysis [5]

Write up sufficient analysis on the performance of the three models in a properly compiled report. This includes the hyperparameters you selected in training (like loss functions, number of epochs, learning rate, optimizer, etc.) and metrics. Report all the metrics and store the results of the model separately.

8.4 Presentation [5]

- Implementation efficiency & quality
- Report quality
- Inclusion of a README along with the code
- Crisp & clear explanation of the code during evaluations

8.5 Bonus: Comparison of Metrics [15]

Optionally, compare the time taken for training, the number of added parameters, and the GPU compute for the three fine-tuning methods (Prompt Tuning, LoRA, and Traditional Fine-Tuning). Report your findings in the analysis.

9 Submission Format

Zip the following into one file and submit it on the Moodle course portal:

- Source code (Should include .py files only.)
- Finetuned models (if file size permits, otherwise provide a link)
- Report answering all the questions in PDF format

- README file (should contain instructions on how to execute the code, restore the pre-trained model, and any other information necessary for clarity)

If the model size crosses the file size limits, upload them to your OneDrive folder and share the read-only accessible links in the README file.

10 FAQs

1. Can I reduce size of dataset or number of epochs due to computational constraints?
   Yes, you can. However, make sure to mention this in your report, with appropriate justification for your choices.

2. Do I need to use lemmatization as a part of my data preprocessing?
   No, lemmatization is not necessary for this assignment.

3. Can I use tools like spaCy, torchtext, etc. for data cleaning and preparation?
   Yes, you are encouraged to use such tools for ease in the data preparation process.

4. Is it okay to include the test data in the pretraining process? No, going by the general ethics in machine learning, data used for evaluation should be something completely unknown to the model.

5. How should the analysis be like?
   We have added the primary points you need to address in Section 8.3. You are free to develop the analysis section based on these pointers.

6. Can I submit my code in Jupyter Notebooks?
   Only the Python script will be considered for grading.

11 Resources

11.1 Papers

- Language Models are Unsupervised Multitask Learners: https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

- The Power of Scale for Parameter-Efficient Prompt Tuning: https://aclanthology.org/2021.emnlp-main.243.pdf

- LoRA: Low-Rank Adaptation of Large Language Models: https://arxiv.org/pdf/2106.09685

11.2 Explanatory Supplements

- LoRA From Scratch – Implement Low-Rank Adaptation for LLMs in PyTorch:
  https://lightning.ai/lightning-ai/studios/code-lora-from-scratch
- Brief Review: The Power of Scale for Parameter-Efficient Prompt Tuning:
  https://sh-tsang.medium.com/brief-review-the-power-of-scale-for-parameter-efficient-prompt-tuning-4d914365025
- Soft Prompts:
  https://learnprompting.org/docs/trainable/soft_prompting
- Prompt Tuning, Hard Prompts & Soft Prompts:
  https://cobusgreyling.medium.com/prompt-tuning-hard-prompts-soft-prompts-49740de6c64c