Assignment 3

Siya Puttagunta

2021101062

Theory Questions

1.

The introduction of soft prompts addresses limitations in discrete text prompts by allowing the prompt to be represented as continuous embeddings rather than fixed text tokens. Discrete prompts, which use specific words or phrases to guide the model are limited by natural language vocabulary and structure, which may not fully capture subtle task-specific nuances or efficiently explore the model's latent space.

Soft prompts are learnable vectors inserted at the input layer, providing more flexibility and efficiency for task-specific conditioning because of the following reasons:

1. **Direct Optimization**: Soft prompts can be directly optimized during training for a given task, bypassing the need to find the "right words" that work well as prompts in natural language.
2. **Reduced Parameter Update**: Soft prompts allow the model to adapt to new tasks without adjusting the entire model's parameters, making training faster and more resource-efficient.

2.

The efficiency of prompt tuning becomes increasingly advantageous as language models scale up. In large-scale language models, traditional fine-tuning methods require updating and storing vast numbers of parameters, which demands significant computational resources and memory. Prompt tuning, however, adjusts only a small subset of parameters (the prompts) while leaving the main model frozen, resulting in:

1. **Significant Reduction in Memory and Compute**: As the model size grows, storing task-specific prompts (rather than an entire model copy) reduces both memory requirements and compute cost, making adaptation to multiple tasks much more feasible.
2. **Efficient Multi-Task Adaptability**: Large models can retain general-purpose knowledge and adapt to numerous tasks with small, optimized prompts, allowing users to switch between tasks without reloading entire model weights. This approach scales well with the growing number of potential applications for LLMs.

3.

Low-Rank Adaptation (LoRA) is a fine-tuning approach designed to address the high memory and computational costs of adapting large language models (LLMs) to specific tasks. LoRA improves efficiency by updating only a small set of task-specific, low-rank matrices instead of modifying the full set of model parameters.

Key Principles:

- **Low-Rank Decomposition**: LoRA introduces low-rank matrices to approximate the parameter updates, leveraging the observation that many updates in fine-tuning can be represented in a reduced, low-rank form. This reduces the number of parameters needed to capture task-specific adaptations.
- **Freezing Main Model Parameters**: The core parameters of the pretrained model remain frozen. Instead, LoRA injects additional, trainable low-rank matrices into certain layers, typically within the model's attention or feedforward layers.
- **Efficient Gradient Update**: During training, gradients only adjust the low-rank matrices, minimizing memory consumption while still allowing the model to learn task-specific nuances.

Efficiency Improvements over Traditional Fine-Tuning:

- **Reduced Memory and Compute Needs**: LoRA updates only the small, low-rank matrices rather than the full parameter set, drastically lowering the memory footprint and enabling efficient storage and transfer of task-specific modifications.
- **Modular Adaptability**: By isolating task-specific changes in separate low-rank matrices, LoRA allows models to store multiple task adaptations without needing an entirely separate fine-tuned model for each task. This makes multi-task adaptation more resource-efficient.

4.

Introducing low-rank adaptations (LoRA) to the parameter space of large language models has significant theoretical implications. By adding low-rank matrices to the existing weight matrices, LoRA effectively reduces the number of trainable parameters, leading to a more efficient training process. This approach maintains model expressiveness while enabling rapid adaptation to specific tasks without full fine-tuning. The low-rank structure allows the model to capture task-specific nuances, potentially enhancing generalization capabilities across diverse domains. Compared to standard fine-tuning, LoRA can prevent overfitting due to fewer parameters, as it leverages the pre-trained model's existing knowledge. Additionally, it

facilitates easier transfer learning by preserving the model's original weights, allowing for quick adjustments in various contexts.

Prompt Tuning

Hyperparameters

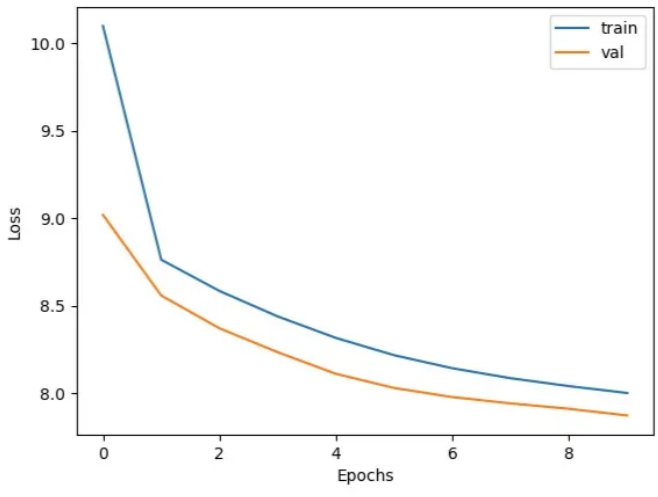
- Prompt = '[SUMMARIZE]'
- num_prompts = 6
- Batch Size = 8
- Learning Rate = 5e-5
- Epochs = 10
- Optimizer = AdamW
- Max Length = 512

Training

```
Number of trainable parameters: 4608
```

```
Epoch 1/10, Training Loss: 10.1112
Epoch 1/10, Validation Loss: 8.9918
Epoch 2/10, Training Loss: 8.7441
Epoch 2/10, Validation Loss: 8.5307
Epoch 3/10, Training Loss: 8.5543
Epoch 3/10, Validation Loss: 8.3446
Epoch 4/10, Training Loss: 8.4194
Epoch 4/10, Validation Loss: 8.2195
Epoch 5/10, Training Loss: 8.2972
Epoch 5/10, Validation Loss: 8.0964
Epoch 6/10, Training Loss: 8.2007
Epoch 6/10, Validation Loss: 8.0282
Epoch 7/10, Training Loss: 8.1293
Epoch 7/10, Validation Loss: 7.9757
Epoch 8/10, Training Loss: 8.0780
Epoch 8/10, Validation Loss: 7.9396
Epoch 9/10, Training Loss: 8.0365
Epoch 9/10, Validation Loss: 7.9079
Epoch 10/10, Training Loss: 7.9987
Epoch 10/10, Validation Loss: 7.8789
Max GPU Memory Used during training: 10393.22 MB
Total time:  3972.211979866028 s
```

Loss Curve



Testing

```
Device: cuda
Test Loss: 7.900955614725748
Rouge Scores:  {'rouge-1': {'r': 0.39417122678150895, 'p': 0.1384894650122559, 'f': 0.2002380277740924}, 'rouge-2': {'r': 0.08433153609012248, 'p': 0.0166792848284411
55, 'f': 0.02726412915862562 8}, 'rouge-l': {'r': 0.3671593053167417, 'p': 0.12871050918195204, 'f': 0.18621338664756187}}
```

Traditional Fine-Tuning

Hyperparameters

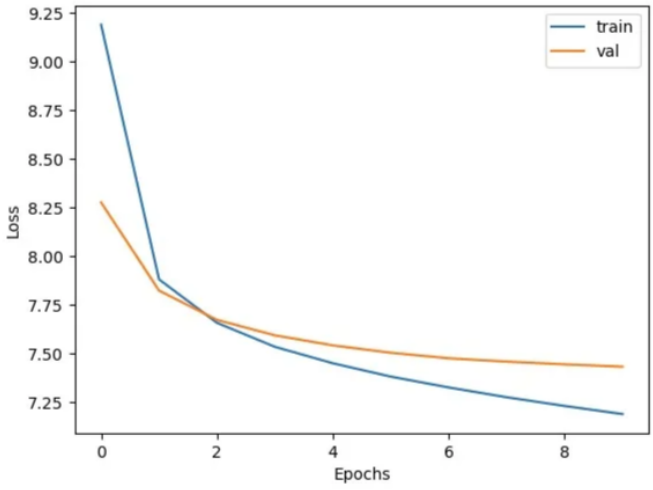
- Batch Size = 8
- Learning Rate = 5e-5
- Epochs = 10
- Optimizer = AdamW
- Max Length = 512

Training

```
Number of trainable parameters: 45685248
```

```
Epoch 1/10, Training Loss: 9.1907
Epoch 1/10, Validation Loss: 8.2780
Epoch 2/10, Training Loss: 7.8827
Epoch 2/10, Validation Loss: 7.8252
Epoch 3/10, Training Loss: 7.6596
Epoch 3/10, Validation Loss: 7.6745
Epoch 4/10, Training Loss: 7.5363
Epoch 4/10, Validation Loss: 7.5952
Epoch 5/10, Training Loss: 7.4518
Epoch 5/10, Validation Loss: 7.5436
Epoch 6/10, Training Loss: 7.3832
Epoch 6/10, Validation Loss: 7.5057
Epoch 7/10, Training Loss: 7.3275
Epoch 7/10, Validation Loss: 7.4773
Epoch 8/10, Training Loss: 7.2772
Epoch 8/10, Validation Loss: 7.4601
Epoch 9/10, Training Loss: 7.2325
Epoch 9/10, Validation Loss: 7.4463
Epoch 10/10, Training Loss: 7.1908
Epoch 10/10, Validation Loss: 7.4341
Max GPU Memory Used during training: 10630.52 MB
Total time:  4918.6916518211365 s
```

Loss Curve



Testing

```
Device: cuda
Test Loss: 7.492344150543213
Rouge Scores:  {'rouge-1': {'r': 0.08052196949587004, 'p': 0.2105488155033811, 'f': 0.11147372467159375}, 'rouge-2': {'r': 0.003786017837603327, 'p': 0.00670916093889
4373, 'f': 0.004550220396994205}, 'rouge-l': {'r': 0.07868466171111857, 'p': 0.2060554570634373, 'f': 0.10896503423381892}}
```

LoRA

Hyperparameters

- lora_r = 128

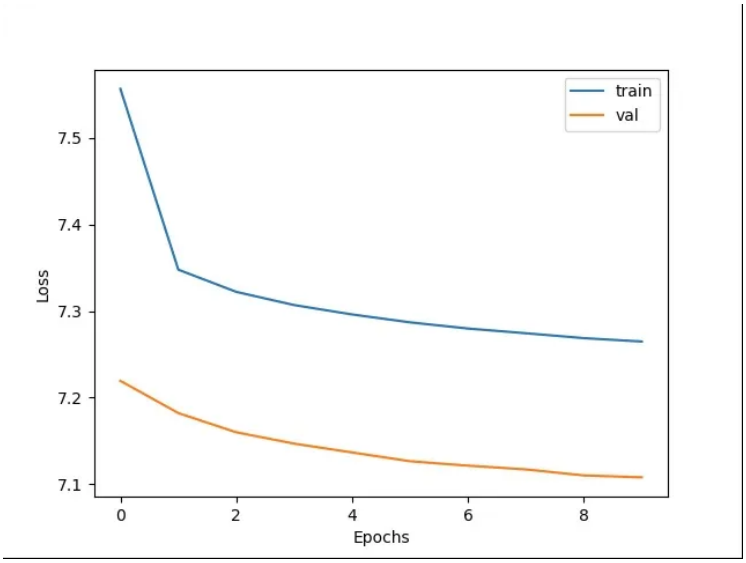
- lora_alpha = 32
- lora_dropout = 0.1
- Batch Size = 8
- Learning Rate = 5e-5
- Epochs = 10
- Optimizer = AdamW
- Max Length = 512

Training

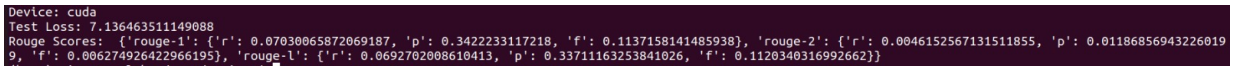
```
trainable params: 4,718,592 || all params: 129,158,400 || trainable%: 3.6533
```

```
Epoch 1/10, Training Loss: 7.5566
Epoch 1/10, Validation Loss: 7.2193
Epoch 2/10, Training Loss: 7.3478
Epoch 2/10, Validation Loss: 7.1821
Epoch 3/10, Training Loss: 7.3221
Epoch 3/10, Validation Loss: 7.1599
Epoch 4/10, Training Loss: 7.3069
Epoch 4/10, Validation Loss: 7.1468
Epoch 5/10, Training Loss: 7.2960
Epoch 5/10, Validation Loss: 7.1366
Epoch 6/10, Training Loss: 7.2869
Epoch 6/10, Validation Loss: 7.1265
Epoch 7/10, Training Loss: 7.2798
Epoch 7/10, Validation Loss: 7.1214
Epoch 8/10, Training Loss: 7.2743
Epoch 8/10, Validation Loss: 7.1170
Epoch 9/10, Training Loss: 7.2687
Epoch 9/10, Validation Loss: 7.1101
Epoch 10/10, Training Loss: 7.2648
Epoch 10/10, Validation Loss: 7.1079
Max GPU Memory Used during training: 10487.85 MB
Total time:  4789.1146030426025 s
```

Loss Curve



Testing



Analysis

Fine-tuning Method	Test Loss	Rouge-L F1 Score	Trainable Parameters	Training Time	GPU Memory Used
Prompt Tuning	7.9	0.18	4608	1.1 hrs	10.39 GB
Traditional Fine-Tuning	7.49	0.10	45685248	1.36 hrs	10.63 GB
LoRA	7.13	0.11	4718592	1.33 hrs	10.48 GB

- **Test Loss:** Prompt Tuning > Traditional Fine-Tuning > LoRA
- **Rouge-L F1 Score:** Prompt Tuning > LoRA > Traditional Fine-Tuning
- **Trainable Parameters:** Prompt Tuning < LoRA < Traditional Fine-Tuning (Prompt Tuning involves less no. of parameters which suggests its adaptibility without extensive re-training)
- **Training Time:** Prompt Tuning < LoRA < Traditional Fine-Tuning (follows the same order as no. of trainable parameters)
- **GPU Memory Used:** Prompt Tuning < LoRA < Traditional Fine-Tuning (follows the same order as no. of trainable parameters)