ADVANCED NLP - ASSIGNMENT 3

Prisha,2021101075

THEORY QUESTIONS

QUESTION : 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?

The introduction of “soft prompts” offers a transformative approach to task-specific conditioning in large language models (LLMs), addressing significant limitations found in discrete text prompts. Discrete prompts are traditional, human-readable text phrases used to elicit desired responses from LLMs. However, discrete prompts have notable limitations:

- **Limited Flexibility and Specificity:** Discrete prompts rely on natural language constructs, which may be limited in expressing nuanced instructions, leading to variable model responses that might not align with task requirements.

- **Manual Tuning and Inefficiency:** Crafting discrete prompts often requires time-consuming, iterative adjustments by users, as slight changes in wording can dramatically affect the output. This manual tuning process lacks scalability, especially in complex, task-specific scenarios.

- **Parameter Constraints:** Discrete prompts do not leverage the model's latent feature space directly, limiting their ability to harness the full representational capacity of the LLM.

Soft Prompts as a Solution

Soft prompts address these limitations by moving beyond discrete tokens to a continuous embedding space, which allows for **direct and more granular control over the model's behavior**. Unlike discrete prompts, soft prompts are not human-readable and consist of learned embeddings optimized within the model's high-dimensional feature space. These embeddings are generated through training, conditioning the model toward task-specific objectives with increased precision and efficiency.

Advantages of Soft Prompts

1.**Enhanced Flexibility:** Since soft prompts operate in a continuous vector space, they allow more precise task conditioning than the rigid structure of discrete prompts. This flexibility enables LLMs to adapt better to diverse tasks, such as sentiment analysis, translation, and summarization, without needing a completely new prompt structure.

2.**Parameter Efficiency:** Soft prompts leverage a smaller set of additional task-specific parameters, keeping the base LLM architecture unchanged. This efficiency is especially valuable when deploying LLMs in resource-constrained environments, as it enables fine-tuning with minimal computational overhead.

3.**Reduction of Human Intervention:** By automatically learning optimal task conditioning during training, soft prompts eliminate the trial-and-error prompt engineering needed in discrete methods, streamlining the process for developers and end-users.

4.**Improved Task Adaptability:** Soft prompts can be adapted through backpropagation for specific tasks, allowing the model to quickly align with nuanced requirements and enabling more consistent output quality across similar prompts.

QUESTION: 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 task.

The efficiency of prompt tuning is closely tied to the scale of the language model. As language models grow larger and more complex, traditional fine-tuning approaches—where all or most model parameters are adjusted for specific tasks—become increasingly resource-intensive and difficult to scale. Prompt tuning, however, introduces a far more efficient method for adapting large-scale models to new tasks, offering significant benefits in terms of computational resources, storage, and flexibility

Efficiency Gains with Prompt Tuning in Large Models

1.**Reduction in Trainable Parameters:** In prompt tuning, only a small set of task-specific prompt embeddings is optimized, leaving the core model parameters frozen. This approach results in considerable reductions in the number of parameters that need training, which is particularly advantageous as models scale to billions of parameters. For instance, a 100-billion-parameter model might require only a few thousand additional parameters in a prompt tuning setup, as opposed to millions in a traditional fine-tuning setting.

2.**Memory and Storage Efficiency:** Because fewer parameters are modified, the storage and memory footprint for each new task-specific prompt is significantly smaller compared to full or partial model fine-tuning. This is especially valuable for deploying large models in production environments, as the same base model

can be adapted for multiple tasks using relatively lightweight prompt tuning configurations .

3.**Computational Savings:** The reduction in trainable parameters leads to faster training and lower computational costs. This is beneficial for large-scale models, where training efficiency can mean substantial time and cost savings. Efficient prompt tuning allows faster experimentation and deployment of task-specific versions of large models, making them more accessible and practical for industry applications.

Implications for Future Developments

The relationship between model scale and prompt tuning efficiency has several significant implications for the future of large-scale language models and their adaptability:

1.**Enhanced Adaptability Across Diverse Tasks:** As models become more general-purpose and foundational, prompt tuning enables users to efficiently tailor these models to niche tasks or specific domains without re-training the entire model.

2.**Cost-Effective Scaling of Language Models:** Prompt tuning allows researchers and developers to extend the utility of large models without incurring the high costs associated with extensive fine-tuning

3.**Increased Accessibility for Specialized Applications:** Efficient prompt tuning makes it feasible for small- to mid-sized organizations to leverage large-scale language models for specific applications without needing specialized infrastructure for full fine-tuning.

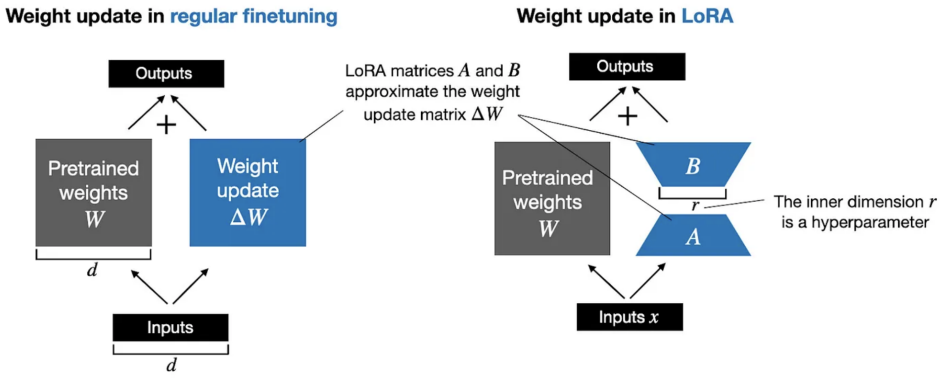
QUESTION : What are the key principles behind Low-Rank Adaptation (LoRA) in fine-tuning large language models? How does LoRA improve upon traditional fine-tuning methods regarding efficiency and performance?

Low-Rank Adaptation (LoRA) is an efficient fine-tuning method that addresses the limitations of traditional fine-tuning by reducing the number of trainable parameters. LoRA works by adding low-rank matrices to specific weights in the model, thereby minimizing the number of parameters that need updating during training. Working with low-rank approximations simplifies the mathematical operations involved in many machine learning and data processing tasks.

Here are the key principles and benefits of LoRA:

1.**Decomposition of Weight Updates:** LoRA modifies the model's architecture by introducing low-rank matrices, which significantly reduces the number of parameters that need to be adjusted during training. As shown in the image below :

`h = W0x + ΔW x = W0x + BAx` where h is the hidden layer, W0 is the pre-trained frozen weights of the pre-trained model of shape (d x k), `ΔW` are the LoRA tracked weights, and `B and A` are the new matrices created of the dimension `(d x r) and (r x k)`. The B and A matrices are created in such a way that after matrix multiplication, we get a matrix of shape (d x k).



2.**Parameter Efficiency:** Since only the low-rank matrices are learned, LoRA requires significantly fewer parameters than traditional fine-tuning. This is highly efficient for large models, as it means only a fraction of the model's parameters are updated, lowering memory and storage needs.Low-rank matrices significantly reduce the number of parameters. This reduction makes computations faster and less memory-intensive.

3.**Performance Consistency:** LoRA maintains performance comparable to full fine-tuning by keeping the main weights of the model frozen. This stability allows LoRA to achieve similar task-specific performance without the risk of overfitting associated with fine-tuning all parameters.

4.**Reduced Computational Cost:** With fewer parameters to train, LoRA lowers computational costs, allowing faster convergence and reducing training time. This is especially advantageous for large models, where computational resources are often a bottleneck.

In essence, LoRA leverages low-rank matrix updates to create a more parameter-efficient fine-tuning process, achieving strong performance with lower computational overhead than traditional fine-tuning methods.

QUESTION : 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?

Introducing Low-Rank Adaptations (LoRA) to the parameter space of large language models has meaningful theoretical implications for the model's expressiveness and generalization capabilities.

1.**Reduced Parameter Complexity with Preserved Expressiveness:** By approximating weight updates with low-rank matrices, LoRA constrains the model to a lower-dimensional subspace, thereby reducing the effective parameter complexity. Despite this constraint, LoRA retains much of the model's expressiveness because the main (frozen) parameters still provide a strong foundation of learned representations, and the low-rank updates can capture essential task-specific variations.

2.**Improved Generalization:** Constraining updates to a low-rank subspace regularizes the model, reducing the likelihood of overfitting. This regularization encourages the model to learn simpler, generalizable features for new tasks, rather than overfitting to noise or specifics of the fine-tuning dataset, which is a common risk in full fine-tuning.

3.**Controlled Adaptability with Efficient Capacity Use:** LoRA enables the model to adapt to new tasks without fully modifying its large parameter space. The low-rank matrices can still encode important task-specific information, but the restriction to a smaller adaptation space inherently controls the degree of task-specific tuning, balancing adaptability and stability in the original model parameters.

4.**Trade-off in Fine-grained Representations:** While LoRA provides efficient task adaptation, the low-rank approximation may limit the model's capacity to learn highly complex, nuanced adaptations compared to standard fine-tuning. This can

be beneficial for generalization on similar tasks but may affect performance on tasks requiring significant deviations from the base model's learned features.

In summary, low-rank adaptations provide a controlled yet expressive way to fine-tune large language models, enhancing generalization and efficiency while maintaining strong task performance, although with potential limitations in tasks that require extensive fine-grained parameter adjustments.

FINE-TUNING USING TRADITIONAL METHOD OF UNFREEZING LAST LAYER

Implementation Details :

In traditional fine-tuning, instead of fine-tuning the entire GPT-2 model, only the last classifier layers are updated, while the rest of the model remains frozen. This reduces computational requirements while still allowing the model to adapt to the new task. Hence, we kept the parameters of the LM head unfrozen in the GPT-2 model; only the rest were frozen.

Notable Values :

Total Number of parameters:124587264

Number of trainable parameters: 38597376
GPU memory allocated: 1857.42 MB

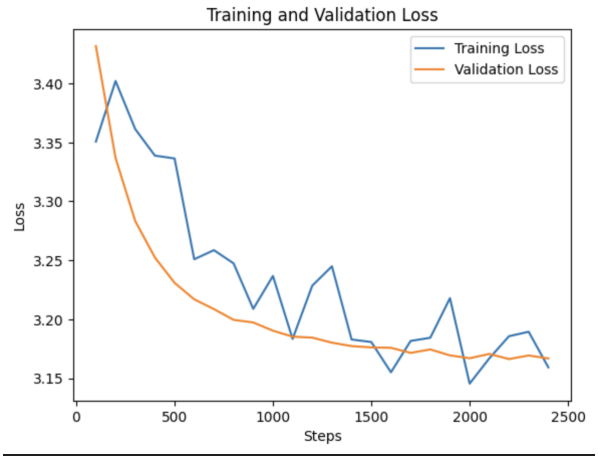
Average ROUGE-1 Score: 0.291331312387222
Average ROUGE-2 Score: 0.1292747932091129
Average ROUGE-L Score: 0.20238509188873338

Training time (seconds): 1699.6672704219818

Test results: {'eval_loss': 3.1388018131256104, 'eval_runtime': 27.2823, 'eval_samples_per_second': 42.115, 'eval_steps_per_second': 5.278, 'epoch': 0.33435497353023125}

[2400/21534

Step	Training Loss	Validation Loss
100	3.350700	3.431694
200	3.402100	3.336799
300	3.361300	3.283325
400	3.338800	3.252369
500	3.336400	3.230885
600	3.250900	3.216898
700	3.258600	3.208629
800	3.247400	3.199492
900	3.208800	3.197267
1000	3.236700	3.190312
1100	3.183300	3.185227
1200	3.228500	3.184411
1300	3.244900	3.180120
1400	3.182800	3.177222
1500	3.180700	3.176097
1600	3.155000	3.175769
1700	3.181600	3.171443
1800	3.184300	3.174385
1900	3.217900	3.169318
2000	3.145400	3.166926
2100	3.166800	3.170565
2200	3.185600	3.166215
2300	3.189400	3.169252
2400	3.159200	3.166747



FINE-TUNING USING LORA

Implementation Details :In the LoRA approach, low-rank matrices are added to specific weight matrices in the GPT-2 model. These matrices allow for 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, 4] and [4, output dim], where the rank is a hyperparameter determining the compression factor we took 4, which is typically much smaller than the input and output dimensions. These matrices are updated during training, while the rest of the model remains frozen.

- Configuration used :

```
lora_config = LoraConfig(
    r=4,
    lora_alpha=32,
    target_modules=["c_attn"],
    lora_dropout=0.1,
    bias="none",
```

```
    task_type="CAUSAL_LM"
)
```

Dataset Used : 10% of the size of the original dataset.

Notable Values :

Total parameters: 124587264

Time taken 1652.5358126163483

Number of trainable parameters: 294912
Estimated FLOPs: 7077888
GPU memory allocated: 507.19 MB

Evaluation Loss: 3.0397279262542725
ROUGE Scores: {'rouge1': 0.28124062631288177, 'rouge2': 0.11407364784042123, 'rougeL': 0.18764651661704718}

[5001/21534

Step	Training Loss	Validation Loss
500	3.107000	3.043425
1000	3.064100	3.042688
1500	3.072900	3.041061
2000	3.060000	3.039882
2500	3.048300	3.039144
3000	3.055000	3.040625
3500	3.072700	3.035505
4000	3.027600	3.038156
4500	3.045300	3.037278
5000	3.031600	3.039728

FINE TUNING USING SOFT PROMPT :

Implementation Details : I created an embedding layer sized [num prompts, embedding_size] for the soft prompts. In this case, num_prompts refers to the number of tokens, and the embedding_size is set to 768. This layer has its own set of parameters, distinct from those of the GPT-2 model. I initialized it with the

[SUMMARISE] token. The soft prompts were added at the beginning of the input sequence, so backpropagation was applied only to the soft prompt embeddings, keeping the GPT-2 parameters frozen.

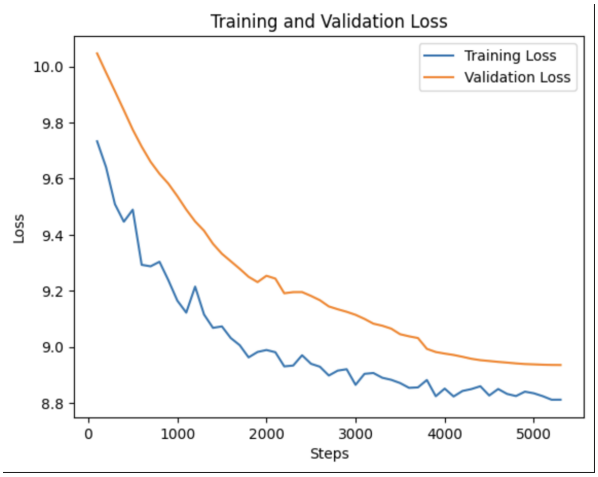
Dataset Used : 10% of the size of the original dataset.

Notable Values :

Evaluation results: {'eval_loss': 8.934844017028809, 'eval_runtime': 6.0305, 'eval_samples_per_second': 33.165, 'eval_steps_per_second': 4.146, 'epoch': 3.0}
Number of trainable parameters: 2304
Estimated FLOPs: 27648
GPU memory allocated: 1975.33 MB

[5349/5349 30:32, Epoch 3/3]

Step	Training Loss	Validation Loss
100	9.733300	10.047009
200	9.641100	9.978683
300	9.508800	9.912032
400	9.446300	9.844051
500	9.489100	9.774864
600	9.292400	9.714208
700	9.287000	9.660138
800	9.303300	9.617360
900	9.237100	9.581841
1000	9.164700	9.537551
1100	9.121800	9.489642
1200	9.214600	9.447174
1300	9.115000	9.413887
1400	9.067400	9.367474
1500	9.072600	9.331943
1600	9.031000	9.305124
1700	9.005600	9.278263
1800	8.962000	9.249636
1900	8.981400	9.230726
2000	8.988400	9.253292
2100	8.980500	9.243222
2200	8.929900	9.190871
2300	8.932900	9.194952
2400	8.969700	9.195110
2500	8.939200	9.181402
2600	8.928700	9.166047
2700	8.897300	9.143859
2800	8.914800	9.133821
2900	8.919800	9.124765
3000	8.864100	9.113955
3100	8.903300	9.099448
3200	8.906300	9.082402
3300	8.893900	9.074991
3400	8.888200	9.064612
3500	8.870400	9.044688
3600	8.853400	9.037215
3700	8.854900	9.031110
3800	8.881300	8.992579
3900	8.823400	8.981241
4000	8.850700	8.975641
4100	8.825000	8.970699
4200	8.822200	8.964128
4300	8.849000	8.956992
4400	8.859100	8.952076
4500	8.826000	8.949098
4600	8.849200	8.945980
4700	8.831600	8.943409
4800	8.824000	8.940526
4900	8.839700	8.938280
5000	8.833900	8.937083
5100	8.823600	8.935840
5200	8.810900	8.935086
5300	8.811000	8.934844



OVERALL RESULTS :

Based on the results of fine-tuning, we find that after analyzing the ROUGE scores and evaluating both the quality and accuracy of the outputs, **LoRA (Low-Rank Adaptation) achieved the highest performance, surpassing other methods by approximately 3x.** Conversely, soft prompting emerged as the least effective approach. **This is primarily due to LoRA's ability to isolate and fine-tune a smaller, low-rank matrix that efficiently captures task-specific nuances,**

whereas soft prompting lacks this targeted optimization, impacting its precision.

When considering the number of parameters, tuning only the final layer (i.e., the complete LM head) in GPT-2 involved the most parameters, while LoRA used the least. LoRA's low-rank trainable matrix results in a compact model structure, requiring significantly fewer parameters.

Memory utilization analysis reveals that LoRA also consumed the minimum GPU RAM during training, while the complete LM head tuning consumed the most, as detailed in the preceding data.

Training time analysis further shows that LoRA achieved the fastest training time, whereas soft prompting took the longest due to its inefficiency in focusing on task-specific layers.

In conclusion, the comparative evaluation demonstrates that **LoRA** consistently outperforms other methods in fine-tuning for language model tasks.