Advanced NLP

ASSIGNMENT 3

October 29, 2024

Ashna Dua
2021101072
ashna.dua@students.iiit.ac.in

Contents

1 Theory Questions

1.1 Question 1

Concept of Soft Prompts: How does the introduction of soft prompt 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 concept of **soft prompts** in large language models (LLMs) is an important recent development aimed at improving how we guide or prompt these models to perform specific tasks. Traditionally, we've used **discrete text prompts** actual words or phrases to instruct the model on what to do. For example, to get an LLM to write a poem, we might start with a prompt like, *Write a poem about nature.* However, while this works, it has some real limitations, especially when it comes to tailoring the model to more complex or very specific tasks.

When we use a discrete text prompt, we're basically relying on a natural language phrase to condition the model. But this approach has a few key issues:

- **Lack of Precision:** Natural language isn't always precise. If we want the model to do something specific, finding the exact words that consistently trigger the right behavior can be tricky. Small changes in wording may lead to unpredictable changes in output, as the LLM interprets phrases in slightly different ways.
- **Difficulty with Optimization:** Since prompts are made of discrete words, there isn't an easy way to tune or optimize them for the best performance on a particular task. If a prompt doesn't work well, we have to experiment with various phrases manually.
- **Token Limitations:** Discrete prompts are built from tokens (the model's vocabulary), so they can't capture information outside this vocabulary. For very specific instructions or technical nuances, discrete text may fall short of fully conveying the task's requirements.

These limitations mean that while discrete prompts are simple and intuitive, they're not always effective for more advanced or task-specific conditioning.

Soft prompts were introduced as a way to get around these issues. Instead of using actual words, a soft prompt uses continuous embeddings, a set of numerical values in the model's embedding space. These embeddings are designed to *represent* a prompt without having to be tied to specific words or tokens in the model's vocabulary.

- **Continuous Embedding Space:** Soft prompts are created as vectors in the continuous embedding space of the LLM. This embedding space is a high-dimensional space where each point represents a concept or idea. By designing soft prompts as vectors in this space, they can direct the model toward certain behaviors or types of output in a more controlled way, without relying on natural language expressions.
- **Parameter Optimization:** Soft prompts are *learnable parameters.* This means we can fine-tune these embeddings during training or through backpropagation to optimize their influence on the model. Unlike discrete prompts, which we have to adjust manually, soft prompts can be tuned automatically based on how well they perform on a specific task.
- **Higher Flexibility and Consistency:** Since soft prompts aren't bound by language tokens, they provide a more consistent way of guiding the model's behavior. Once optimized, a soft

prompt can consistently produce the desired response, even for complex tasks, because it directly manipulates the model's internal representations.

Soft prompts are considered more flexible and efficient for task-specific conditioning for a few key reasons:

- **Task-Specific Adaptability:** Soft prompts can be trained for a specific task, meaning they're better suited for cases where we want the model to act in a very precise way. We can adapt them to tasks like classification, summarization, or sentiment analysis by optimizing the embeddings specifically for that purpose.
- **Reduced Computational Cost:** With soft prompts, we don't have to fine-tune the entire model on a new task, which would require significant resources. Instead, we just optimize the soft prompts, saving time and computational power. This makes soft prompts an efficient solution for task adaptation.
- **Improved Control Over Output:** Since soft prompts are numeric embeddings rather than words, they allow us to exercise fine control over the LLM's behavior. We can gradually adjust the embeddings to nudge the model in a specific direction, providing more controlled and nuanced guidance than we could achieve with language-based prompts.

1.2 Question 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.

As language models continue to grow in scale, with billions or even trillions of parameters, **prompt tuning** has emerged as an efficient method for adapting these models to specific tasks without the need for extensive retraining. The relationship between prompt tuning efficiency and the scale of the language model is essential for understanding how large models can be deployed and adapted for various applications. This relationship has significant implications for the future of large-scale language models, particularly in terms of resource allocation, task adaptability, and general usability.

Prompt tuning involves optimizing a small set of parameters (the prompt) rather than the entire model. The efficiency of this process increases as models scale up for several reasons:

- **Limited Parameter Adjustment:** In prompt tuning, only a small number of prompt parameters are fine-tuned, while the core model parameters remain fixed. As the language model scales, prompt tuning continues to only involve these few prompt parameters, meaning that the computational requirements for task adaptation remain low regardless of model size.
- **Leverage of Large Representations:** Larger models have more nuanced and comprehensive language representations. This richness enables prompt tuning to be more effective, as small adjustments to the prompt can leverage the extensive knowledge encoded in the model's layers without requiring model-wide updates. Essentially, as the model scales, prompt tuning becomes more effective at controlling behavior with minimal computational effort.

- **Reduction in Full Fine-Tuning Necessity:** Larger models typically require more resources to fine-tune across all layers. With prompt tuning, however, only the prompt parameters are modified, reducing the need to allocate massive resources for retraining. Therefore, prompt tuning's efficiency scales positively with model size, as the need for full fine-tuning becomes less pressing.

The efficiency of prompt tuning relative to model scale has several important implications for the future of large language models and their adaptability:

1. Resource Savings and Accessibility

With the growth of model size, full model fine-tuning becomes highly resource-intensive. Prompt tuning offers a cost-effective and less resource-demanding solution:

- This means that organizations with limited computational resources can still adapt massive models for specific applications using prompt tuning.
- As a result, large-scale language models become more accessible for various industries, from healthcare to finance, which may need model adaptation but lack the infrastructure for full retraining.

2. Rapid Adaptation for Diverse Applications

Prompt tuning allows large models to be adapted quickly to different tasks by simply adjusting the prompt parameters:

- This capability supports the deployment of general-purpose language models across diverse domains without requiring model-specific modifications.
- It enables a more modular approach to LLM deployment, where a single base model can be rapidly tailored to specific tasks via different prompts, making large models more versatile.

3. Enhanced Task-Specific Performance

Prompt tuning becomes more precise with larger models because small prompt changes can harness the model's extensive pre-trained knowledge:

- For specialized or niche tasks, prompt tuning can selectively activate relevant aspects of a model's knowledge, achieving high performance without the need for additional data or extended training periods.
- This enhances the model's ability to generalize effectively across tasks, since prompt tuning allows for specific, efficient customization.

4. Future Innovation in Model Architecture and Prompting Techniques

The positive scaling relationship between prompt tuning efficiency and model size opens new avenues for research in model architecture and prompt engineering:

- There is likely to be a focus on designing model architectures that are even more compatible with prompt tuning, potentially incorporating specialized layers or embeddings that respond more precisely to prompts.
- Prompt tuning may also drive the development of more advanced prompt engineering techniques, such as hybrid prompts combining discrete and soft prompts or dynamic prompts that adapt based on task complexity.

1.3 Question 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?

As language models grow in scale, fine-tuning them on specific tasks becomes increasingly computationally expensive and resource-intensive. **Low-Rank Adaptation** (LoRA) is a technique designed to make fine-tuning large language models more efficient by reducing the number of trainable parameters. This approach helps maintain performance while lowering memory and computation requirements, which are major concerns in traditional fine-tuning.

The core idea of Low-Rank Adaptation (LoRA) is to reduce the complexity of fine-tuning by introducing low-rank matrices in place of directly modifying the full parameter matrices of the model. Here are the main principles behind LoRA:

- **Parameter Efficiency Through Low-Rank Matrices:** LoRA assumes that updates required for task-specific adaptations lie in a lower-dimensional subspace. Instead of updating the full weight matrices of the model, LoRA introduces two low-rank matrices A and B with rank r. These matrices approximate the necessary task-specific adjustments with fewer parameters.

- **Decomposition of Parameter Updates:** In LoRA, the parameter update for a weight matrix W is decomposed as follows:

  $$W' = W + \Delta W = W + BA$$

  where W' is the adapted matrix, and ΔW is the low-rank update matrix computed as BA, with B and A having dimensions corresponding to the low-rank configuration. This decomposition allows the updates to be learned without modifying W directly.

- **Trainability of Low-Rank Matrices Only:** Instead of training the entire parameter matrix W, LoRA trains only the matrices B and A, which significantly reduces the number of parameters that need to be learned. This greatly reduces memory usage and computational cost.

The low-rank structure provides an efficient way to represent the directions of variation needed for fine-tuning, which simplifies adaptation and reduces overfitting risk by limiting the number of adaptable parameters.

LoRA's approach to fine-tuning offers several advantages over traditional methods, particularly in terms of efficiency and adaptability:

1. Reduced Computational Cost

Traditional fine-tuning requires updating and storing gradients for all parameters in a large model. However:

- LoRA focuses only on learning the low-rank matrices B and A, reducing the number of trainable parameters. For example, if W has dimensions $d \times d$, LoRA's matrices B and A can have dimensions $d \times r$ and $r \times d$, respectively, where $r \ll d$.

- This reduction in parameters results in faster training times and lower memory usage, making LoRA a more computationally feasible choice for fine-tuning large models.

2. Memory Efficiency
LoRA's low-rank adaptation uses fewer parameters and thus requires less storage during training:

- Traditional fine-tuning requires storing gradients for all parameters. By training only A and B, LoRA reduces the total gradient memory overhead.
- This memory efficiency allows LoRA to fine-tune larger models on GPUs with limited memory, making it a practical choice for resource-constrained environments.

3. Reduced Overfitting Risk
By constraining the number of trainable parameters, LoRA may also reduce overfitting, especially in scenarios with limited task-specific data:

- LoRA's low-rank approach effectively imposes a regularization effect, preventing the model from making unnecessary adjustments and focusing only on critical task-related modifications.
- This helps the model generalize better and improves performance on tasks where data is scarce.

4. Compatibility with Pre-Trained Models
LoRA modifies the model by adding low-rank matrices rather than changing the model's core weights, which has the following benefits:

- The original pre-trained weights remain unchanged, preserving the general knowledge already embedded in the model. This allows LoRA to leverage the existing pre-trained knowledge more effectively.
- Fine-tuned low-rank matrices can be added to different parts of the model as needed, allowing flexible task adaptation without the need to retrain or significantly alter the original model.

Given its efficiency and memory advantages, LoRA is highly suited for large-scale language models, enabling them to be adapted for specific tasks at a fraction of the resource cost. Here's why LoRA is important for future developments in LLM fine-tuning:

- **Scalability:** LoRA's efficiency allows for scalable fine-tuning on extremely large models, as it can be applied without excessive memory demands, even as models continue to grow.
- **Broad Accessibility:** By reducing the computational and memory requirements of fine-tuning, LoRA makes adapting large language models accessible to organizations or researchers without extensive computational resources.
- **Enhanced Task-Specific Adaptation:** With LoRA, large pre-trained models can be quickly adapted to specialized tasks or domains without retraining the entire model, supporting faster deployment across varied applications.

1.4 Question 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 generaliza- tion capabilities of the model compared to standard fine-tuning?

Low-Rank Adaptation (LoRA) introduces low-rank matrices as a method for fine-tuning large language models, which significantly reduces the number of trainable parameters. This adaptation can have profound theoretical implications, particularly in terms of the modelâs **expressiveness** (its ability to capture complex patterns in data) and **generalization** (its capacity to perform well on unseen data).

In traditional fine-tuning, all parameters within the large language model are adjusted to accommodate the specific task. However, this can be inefficient and susceptible to overfitting, as it updates even those parameters not strictly necessary for the new task. LoRA addresses this by *restricting* the parameter space using low-rank matrices.

- **Dimensionality Reduction of Parameter Updates:** By introducing low-rank matrices, LoRA projects high-dimensional updates into a lower-dimensional space, which reduces the complexity of the parameter adaptation.
- **Decomposition of Adaptations:** Let W represent the weight matrix in the pre-trained model. LoRA introduces matrices A and B such that the adaptation can be expressed as:

$$W' = W + \Delta W = W + BA$$

  where B and A are low-rank matrices with rank $r \ll d$, where d is the dimensionality of W. This decomposition forces the task-specific information into a lower-dimensional subspace, which could have implications for both expressiveness and generalization.

Expressiveness, in the context of large language models, refers to the modelâs ability to capture complex relationships and represent rich, diverse linguistic structures.

1. Potential Reduction in Expressiveness Since LoRA constrains the parameter space by forcing updates into a low-rank subspace, there is a theoretical limitation on the expressiveness of the model.

- Low-rank matrices may not capture highly intricate task-specific patterns, especially if the task requires complex reconfigurations of the original pre-trained features.
- For tasks that require subtle adjustments across many dimensions, this constraint could lead to a loss in expressiveness compared to full-rank fine-tuning, as the model may lack the flexibility to represent detailed variations.

2. Focused Adaptations Improve Task-Specific Expressiveness However, in many cases, the constraints imposed by LoRA are beneficial rather than limiting:

- By restricting the update space, LoRA may *filter out noisy or irrelevant variations*, which can allow the model to focus more on the most salient task-specific features.
- Low-rank adaptations encourage the model to capture only the *essential patterns* necessary for the task, which can effectively enhance expressiveness in more structured or repetitive tasks by avoiding overfitting to highly specific or irrelevant features.

Thus, while LoRA may theoretically reduce the model's overall expressiveness, it can improve task-specific expressiveness by focusing on the most meaningful adaptations. This can be particularly advantageous for structured tasks that align well with the low-rank adaptations.

The generalization capability of a language model determines its performance on data it has not seen during training. LoRAâs low-rank adaptation has significant implications for generalization:

1. Improved Generalization Through Regularization The low-rank restriction in LoRA serves as a form of regularization, which can enhance generalization by preventing the model from overfitting:

- By limiting updates to a low-rank subspace, LoRA reduces the risk of overfitting, especially in cases where task-specific data is limited.
- This regularization effect can lead to better performance on unseen data, as it discourages the model from "memorizing" the fine details of the training data, focusing instead on more general patterns.

2. Retention of Pre-Trained Knowledge LoRA also preserves the pre-trained weights in their entirety, with low-rank matrices added as separate components. This design choice enhances generalization in the following ways:

- Since the pre-trained parameters W remain unchanged, the general linguistic knowledge embedded in the original model is preserved, providing a strong foundation for generalization.
- LoRA adapts the model for the specific task without disrupting the foundational linguistic features, allowing the model to apply general linguistic principles learned during pre-training to novel examples within the task.

The ability to retain and selectively augment pre-trained knowledge makes LoRA particularly effective for generalization across tasks that share linguistic characteristics with the pre-training corpus.

When we compare LoRA with traditional fine-tuning, there are clear trade-offs between expressiveness and generalization.

- **Expressiveness:** Standard fine-tuning has an advantage in expressiveness, as it allows the model to adapt all parameters to the new task. This can be crucial for tasks that require highly specific patterns or intricate transformations. However, it also increases the risk of overfitting, especially on smaller datasets.
- **Generalization:** LoRA often outperforms traditional fine-tuning in terms of generalization, thanks to its regularizing effect. By constraining the parameter updates, LoRA prevents the model from fitting to task-specific data noise, leading to better generalization on unseen data.

Overall, LoRA provides a balance between expressiveness and generalization that is often more efficient and effective than traditional fine-tuning, particularly in scenarios where computational resources are limited or when task-specific data is sparse.

2 Prompt Tuning Analysis

The task involves using a soft prompt token, `[SUMMARIZE]`, to condition the model specifically for summarization tasks, with the objective of improving model performance without full retraining.

2.1 Environment Setup

The environment was prepared by importing the necessary libraries, including `torch` and `transformers`, specifically using `GPT2LMHeadModel` and `GPT2Tokenizer`. A custom vocabulary token, `[SUMMARIZE]`, was incorporated as the soft prompt token to facilitate task-specific conditioning.

2.2 Hyperparameter Selection

The following hyperparameters were employed in the model training:

- **Batch Size:** 1
- **Epochs:** 1
- **Max Length:** 1024 (input sequence length limit)
- **Learning Rate:** 1×10^{-4}
- **Gradient Accumulation Steps:** 4
- **Gradient Clip Norm:** 1.0
- **Early Stopping Patience:** 1

2.3 Soft Prompt Integration

The token `[SUMMARIZE]` was added as a custom vocabulary item, and a mapping, `soft_prompt_word2idx`, was created to link this custom token to its corresponding index, allowing the model to interpret `[SUMMARIZE]` as part of the input.

2.4 Results Evaluation

2.4.1 Training and Validation Metrics

Metric	**Value**
Train Loss (Epoch 1)	9.4259
Validation Loss (Epoch 1)	8.8849

Table 1: Training and Validation Losses

Metric	**Value**
Validation ROUGE-1	0.0912
Validation ROUGE-2	0.0229
Validation ROUGE-L	0.0579
Test ROUGE-1	0.0891
Test ROUGE-2	0.0221
Test ROUGE-L	0.0566

Table 2: ROUGE Scores for Validation and Testing

Phase	**GPU Compute Time (ms)**
Training Epoch 1	5819270.50
Validation Epoch 1	195395.25

Table 3: GPU Compute Times

3 LORA Finetuning Analysis

The task involves applying Low-Rank Adaptation (LoRA) to fine-tune a pre-trained GPT-2 model for summarization tasks. The objective is to enhance model performance while significantly reducing the number of trainable parameters compared to traditional fine-tuning methods.

3.1 Environment Setup

The environment was prepared by importing the necessary libraries, including `torch`, `transformers`, and `peft`. The `GPT2LMHeadModel` and `GPT2Tokenizer` were utilized from the `transformers` library. Additionally, the LoRA configuration was established to facilitate efficient adaptation of the model.

3.2 Hyperparameter Selection

The following hyperparameters were employed in the model training:

- **Batch Size:** 1
- **Epochs:** 1
- **Max Length:** 1024 (input sequence length limit)
- **Learning Rate:** 1×10^{-3}
- **LoRA Rank (r):** 4
- **LoRA Alpha:** 32
- **LoRA Dropout:** 0.1
- **Gradient Accumulation Steps:** 4
- **Gradient Clip Norm:** 1.0
- **Early Stopping Patience:** 1

3.3 LoRA Configuration

LoRA configuration was defined to specify the target modules and dropout rates. This allows the model to maintain a lightweight structure while adapting to the summarization task. The configuration included parameters such as rank, alpha, and dropout, which influence the effectiveness of the adaptation process.

3.4 Results Evaluation

3.4.1 Training and Validation Metrics

Metric	**Value**
Train Loss (Epoch 1)	7.3975
Validation Loss (Epoch 1)	7.2039

Table 4: Training and Validation Losses

Metric	**Value**
Validation ROUGE-1	0.0979
Validation ROUGE-2	0.0041
Validation ROUGE-L	0.0808
Test ROUGE-1	0.0982
Test ROUGE-2	0.0048
Test ROUGE-L	0.0805

Table 5: ROUGE Scores for Validation and Testing

Phase	**GPU Compute Time (ms)**
Training Epoch 1	4403551.50
Validation Epoch 1	159648.27

Table 6: GPU Compute Times

4 Traditional Finetuning Analysis

The task involves applying traditional fine-tuning to a pre-trained GPT-2 model for summarization tasks. The objective is to enhance the model's performance by training only the last layer while preserving the knowledge encoded in the earlier layers.

4.1 Environment Setup

The environment was prepared by importing the necessary libraries, including `torch` and `transformers`. The `GPT2LMHeadModel` and `GPT2Tokenizer` were utilized from the `transformers` library. The model was initialized with pre-trained weights, and only the last layer was set to be trainable.

4.2 Hyperparameter Selection

The following hyperparameters were employed in the model training:

- **Batch Size:** 1
- **Epochs:** 1
- **Max Length:** 1024 (input sequence length limit)
- **Learning Rate:** 1×10^{-3}
- **Gradient Accumulation Steps:** 4
- **Gradient Clip Norm:** 1.0
- **Early Stopping Patience:** 1

5 Fine-Tuning Procedure

During the fine-tuning process, only the last layer of the model was unfrozen. The remaining layers were kept fixed to leverage the pre-trained weights. This approach allowed the model to adapt specifically to the summarization task with minimal training overhead.

6 Results Evaluation

6.1 Training and Validation Metrics

Metric	**Value**
Train Loss (Epoch 1)	8.2446
Validation Loss (Epoch 1)	7.6454

Table 7: Training and Validation Losses

Metric	**Value**
Validation ROUGE-1	0.0791
Validation ROUGE-2	0.0040
Validation ROUGE-L	0.0575
Test ROUGE-1	0.0796
Test ROUGE-2	0.0041
Test ROUGE-L	0.0579

Table 8: ROUGE Scores for Validation and Testing

7 GPU Compute

Phase	GPU Compute Time (ms)
Training Epoch 1	5369129.50
Validation Epoch 1	183794.66

Table 9: GPU Compute Times

8 Number of Added Parameters

The number of added parameters in the three models can be observed below:

Type	Number of Parameters)
Prompt	768
LoRA	147,456
Traditional	38597376

Table 10: Number of Parameters

9 Time taken to train model

The number of added parameters in the three models can be observed below:

Type	Time taken to Train (sec))
Prompt	5818
LoRA	5003
Traditional	5368

Table 11: Number of Parameters

10 Observations

- **Performance Comparison:** Among the three methods, LoRA fine-tuning achieved the lowest training and validation loss (7.3975 and 7.2039, respectively), indicating better model performance for summarization tasks compared to soft prompt tuning (train loss: 9.4259, validation loss: 8.8849) and traditional fine-tuning (train loss: 8.2446, validation loss: 7.6454). The ROUGE scores further support this, with LoRA yielding the highest values across validation and test sets.
- **Efficiency in Parameter Usage:** The number of parameters added was significantly lower for the LoRA and prompt tuning methods (147,456 and 768, respectively) compared to

traditional fine-tuning (38,597,376). This indicates that LoRA and prompt tuning effectively enhance performance with fewer additional parameters, demonstrating the potential for efficient adaptations in large language models.

- **Compute Time Analysis:** The compute times indicate that prompt tuning required the most GPU time (5819 seconds), followed closely by traditional fine-tuning (5369 seconds), while LoRA fine-tuning was the fastest (4403 seconds). This suggests that LoRA fine-tuning not only enhances performance but also reduces computational overhead, making it a practical choice for adapting models for specific tasks.