Report

Gaurav Bhole - 2021113015

8.1: Theory:

1. Concept of Soft Prompts: Soft prompts address limitations of discrete text prompts through learnable continuous embeddings that exist in the model's latent space rather than being constrained to actual tokens. Key advantages include:
    - Flexibility in Representation: Soft prompts can learn optimal task-specific representations that may not correspond to any actual tokens in the vocabulary, allowing them to capture more nuanced patterns and relationships than discrete text prompts.
    - Continuous Optimization: Unlike discrete prompts that are limited to existing tokens, soft prompts can be continuously optimized via gradient descent, enabling fine-grained adaptation to the task.
    - Expressive Power: They can encode complex task instructions and conditioning that might be difficult or impossible to express through natural language tokens.
    - Parameter Efficiency: Soft prompts add very few trainable parameters (typically just the prompt embeddings) while allowing effective task-specific adaptation.
    - Task Compositionality: Multiple soft prompts can be combined or interpolated to handle multiple tasks or create new task variations.

2. Scaling and Efficiency in Prompt Tuning: The relationship between model scale and prompt tuning efficiency has important implications:
    - Performance Scaling: As language models get larger, the effectiveness of prompt tuning improves disproportionately. The performance gap between prompt tuning and full fine-tuning narrows significantly with scale.
    - Parameter Efficiency: The number of trainable parameters in prompt tuning remains constant regardless of model size, making it increasingly efficient for larger models. This contrasts with full fine-tuning where parameters grow linearly with model size.
    - Memory Benefits: Prompt tuning requires storing only the prompt parameters, while full fine-tuning needs separate copies of model weights for each task. This becomes crucial for very large models.
    - Future Implications: As models continue to grow in size, prompt tuning may become the default adaptation method due to its combination of strong performance and resource efficiency.

3. Understanding LoRA: Key principles of Low-Rank Adaptation include:
    - Matrix Factorization: LoRA decomposes weight updates into products of lower-rank matrices (W + BA where B and A are low-rank), significantly reducing the number of trainable parameters.
    - Rank-Based Compression: The rank hyperparameter controls the capacity/compression tradeoff, allowing flexible scaling of adaptation complexity.
    - Parameter Freezing: Original model weights remain frozen while only the low-rank adaptation matrices are trained.
    - Training Efficiency: By training fewer parameters, LoRA reduces memory requirements and training time while maintaining performance comparable to full fine-tuning.

4. Theoretical Implications of LoRA: The success of LoRA has several important theoretical implications:
    - Low-Rank Structure: The effectiveness of low-rank updates suggests that many model adaptations naturally lie in a low-dimensional subspace, challenging the need for full parameter fine-tuning.
    - Implicit Regularization: The low-rank constraint acts as a form of regularization, potentially improving generalization by restricting the space of possible adaptations.
    - Model Capacity: LoRA demonstrates that model capacity can be effectively expanded through targeted, low-dimensional updates rather than full parameter tuning.
    - Optimization Landscape: The success of low-rank updates suggests that the optimization landscape for model adaptation may be more structured than previously thought.
    - Modularity: The ability to achieve strong performance with modular, low-rank updates suggests possibilities for more efficient and composable model adaptation strategies.

8.3: Analysis Report: Comparison of Fine-tuning Methods for GPT-2 on Summarization Task

Training Configuration

Common hyperparameters across all methods:

- Base Model: GPT-2 Small
- Number of Epochs: 20
- Batch Size: 8
- Initial Learning Rate: 1e-4
- Optimizer: AdamW
- Loss Function: CrossEntropyLoss
- Learning Rate Schedule: Linear with warmup (100 steps)
- Dataset: 10% of CNN/Daily Mail dataset
- Maximum sequence lengths: 512 (input), 128 (summary)

Method-specific configurations:

1. Prompt Tuning:
    - Soft prompt length: 20 tokens
    - Only prompt embeddings trainable
    - Random initialization with range ±0.5
2. LoRA:
    - Rank: 8
    - Alpha: 32
    - Applied to attention layers
    - Initialization: Zero for B matrices, Normal for A matrices
3. Traditional Fine-tuning:
    - Only last transformer block and LM head trainable
    - All other parameters frozen

Performance Analysis

1. Training and Validation Loss

All three methods showed similar loss patterns:

- Initial rapid decrease in training loss
- Stabilization after ~5 epochs
- Slight divergence between training and validation loss

Best Validation Losses:

1. Prompt Tuning: 0.9863 (epoch 5)
2. Traditional Fine-tuning: 0.9998 (epoch 4)
3. LoRA: 1.0006 (epoch 7)

Prompt Tuning achieved the lowest validation loss, suggesting better generalization despite having fewer trainable parameters.

2. ROUGE Scores

All methods achieved similar ROUGE scores:

ROUGE-1: ~0.056 (5.6%)
ROUGE-2: ~0.031 (3.1%)
ROUGE-L: ~0.049 (4.9%)

The similarity in ROUGE scores across methods suggests that all approaches were equally effective at learning the summarization task, despite their different mechanisms.

3. Resource Usage

GPU Memory Usage:

- LoRA: 94.24%
- Traditional Fine-tuning: 94.24%
- Prompt Tuning: 89.43%

CPU Usage (relatively consistent across methods):

- LoRA: 7.62%
- Traditional Fine-tuning: 7.74%
- Prompt Tuning: 7.52%

Prompt Tuning showed slightly lower GPU memory usage while maintaining comparable performance.

4. Learning Dynamics

Training Loss Patterns:

- Prompt Tuning: Most aggressive initial decrease (1.29 → 0.83)
- LoRA: Moderate decrease (1.05 → 0.95)
- Traditional Fine-tuning: Similar to LoRA (1.06 → 0.95)

All methods showed stable learning with minimal oscillation in validation loss.

5. Parameter Efficiency

Total Parameters vs. Trainable Parameters:

- Base Model: ~124M parameters
- Prompt Tuning: Added minimal parameters (soft prompts only)
- LoRA: Added low-rank matrices
- Traditional Fine-tuning: No additional parameters

Key Findings

1. **Effectiveness**: All three methods achieved comparable performance metrics, suggesting they are all viable approaches for the summarization task.
2. **Efficiency**:
    - Prompt Tuning showed the best parameter efficiency while achieving slightly better validation loss
    - LoRA provided a good balance between performance and parameter count
    - Traditional Fine-tuning was competitive despite only updating the last layers
3. **Stability**: All methods demonstrated stable training with minimal overfitting, as evidenced by the relatively small gap between training and validation losses.
4. **Resource Usage**: Prompt Tuning had a slight edge in GPU memory efficiency, while CPU usage was comparable across methods.

Recommendations

1. For resource-constrained environments: Choose Prompt Tuning due to its lower memory footprint and competitive performance.
2. For best balance of performance and complexity: Use LoRA, as it provides comparable results with a clean implementation approach.
3. For simplicity and reliability: Traditional Fine-tuning remains a solid choice, showing robust performance with minimal complexity.

Limitations

1. Relatively low ROUGE scores across all methods suggest room for improvement in the summarization quality.
2. The training was conducted on only 10% of the dataset, which might not represent full potential performance.