SUMMARIZE PAPER

Nguyen Khanh Huyen

15th May 2024

1 LORA - Low-Rank Adaptation ofLarge Language Models

1.1 Introduction

This paper presents an approach to adapting large-scale pre-training models like GPT-3 with 175B parameters to specific tasks or domains while minimizing computational and memory costs typically associated with such adaptations.

1.2 Baseline

The paper discusses several baseline methods including Fine-Tuning (FT), Bias-only or BitFit, Prefix Embedding Tuning, Prefix Layer Tuning, and Adapter Tuning.

- Fine-Tune(FT): All parameters of a pre-trained model are updated
- Bias-only or BitFit: fine-tuning only the bias parameters of a pre-trained mode
- Prefix Embedding Tunning: optimizing a set of continuous vectors (prefixes) that are prepended to the input sequence, effectively steering the model's behavior without modifying the pre-existing weights
- Prefix Layer Tuning: Similar to Prefix Embedding, but instead of adding vectors to the input, it adapts a small number of layers at the beginning of the model.
- Adapter Tunning: This method inserts trainable layers (adapters) between existing layers of a pre-trained model, allowing for task-specific adjustments without altering the original weights.

1.3 Other Solutions

The problem of making model adaptation more efficient in terms of parameters and computation is well-established. Two main strategies for efficient adaptations in language modeling are adding adapter layers and optimizing input layer activations.

- Adapter Layers:
  - Adapter layers introduce inference latency
  - Original and recent designs have limitations in reducing compute requirements
  - Despite having few parameters, adapter layers increase latency due to sequential processing
  - Latency issues worsen in large-scale scenarios requiring hardware parallelism and GPU operations.
- Direct Optimization of Prompts
  - Direct optimization, such as prefix tuning, is difficult to optimize and performance varies with the number of trainable parameters

  - Deserving sequence length for adaptation reduces the length available for downstream tasks, potentially decreasing performance.

1.4 Methodology

1.4.1 Low-Rank-Parametrized update matrics

- By improving low-rank decomposition matrices, it modifies pre-trained models in an indirect manner, maintaining the frozen pre-trained weights.
- Benefits:
  - Shared models across tasks
  - Reduced storage requirements
  - Higher training efficiency
  - No additional inference latency
  - Compatibility with other methods like prefix-tuning
- Neural networks have dense layers performing matrix multiplication with typically full-rank weight matrices.
- Pre-trained language models have a low "intrinsic dimension," enabling efficient learning in smaller subspaces.
- Hypothesis: Weight updates during adaptation have a low "intrinsic rank."
- Constrain updates with low-rank decomposition $W_0 + W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$, with $r \leq \min(d, k)$.
- During training, W_0 is fixed, and only A and B are trainable.
- Modified forward pass: $h = W_0 x + Wx = W_0 x + BAx$.
- Initialization: A with a random Gaussian and B with zeros, resulting in $W = BA = 0$ initially.
- Scale Wx by a constant α to reduce hyperparameter tuning needs.
- LoRA generalizes fine-tuning by not requiring full-rank gradient updates, matching full fine-tuning expressiveness as r increases.
- No additional inference latency: $W = W_0 + BA$ can be pre-computed and stored, with quick task switching by updating BA, maintaining efficiency.

1.4.2 Applying LoRA to Transformer

- LoRA can be applied to any subset of weight matrices in a neural network to reduce trainable parameters.
- In Transformer, it treats W_q (or W_k, W_v) as a single matrix of dimension $d_{\text{model}} \times d_{\text{model}}$, despite the output being divided into attention heads.
- The study focuses on adapting attention weights for downstream tasks while freezing MLP modules to maintain simplicity and parameter efficiency.

1.4.3 Understanding the Low-Rank Updates

Parameter Budget Efficiency

- Focuses on which weight matrix subsets to adapt using LoRA for best downstream performance.

- Adapting both query (W_q) and value (W_v) weights in the self-attention module yields the best results.
- Indicates that low-rank adaptation captures sufficient information for effective adaptation.

Optimal Rank Determination

- Shows that even a rank as low as one can provide competitive performance.
- Suggests that adaptation matrices have an "intrinsic rank," making low-rank adaptations sufficient.
- Increasing the rank does not necessarily cover a more meaningful subspace.

Subspace Similarity

- Normalized subspace similarity measured using the Grassmann distance shows significant overlap in top singular vector directions between different ranks.
- Supports the idea of a low "intrinsic rank."

Adaptation Matrix vs. Pre-trained Weights

- Adaptation matrix ΔW correlates with pre-trained weights W.
- Amplifies features present in W but not emphasized, catering to specific downstream tasks.
- **Parameter Budget Efficiency**: Focuses on which weight matrix subsets to adapt using LoRA for best downstream performance.
- **Optimal Rank Determination**: Shows that even a rank as low as one can provide competitive performance.
- **Subspace Similarity**: Normalized subspace similarity measured using the Grassmann distance shows significant overlap in top singular vector directions between different ranks.
- **Adaptation Matrix vs. Pre-trained Weights**: Adaptation matrix ΔW correlates with pre-trained weights W.

1.5 Conclusion

- Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, no additional inference latency.
- LoRA is a cost-effective strategy for adapting large language models, reducing hardware and storage demands.
- Enables rapid task switching without extra latency or reduced input sequence length. Ideal for service deployment with parameter-sharing capabilities.
- Applicable to Transformer models and other neural networks with dense layers.
- Future Directions:
  - Integrate with other adaptation methods, explore feature transformation, and develop systematic weight matrix selection.
  - Investigate rank-deficiency in pre-trained and adaptation matrices for training improvements.

2 PARAMETER-EFFICIENT TRANSFER LEARNING FOR NLP

2.1 Introduction

This paper proposes an alternative approach - transfer with adapter modules instead of fine-tuning. Adapter modules offer a compact, extensible model by adding only a few trainable parameters per task, yielding a high degree of parameter sharing.

2.2 Adapter Model

- **Adapter Modules**
  - Adapters are new modules added between layers of a pre-trained network.
  - Different from feature-based transfer and fine-tuning; adapters modify the function $w(x)$ to $w_v(x)$ with pre-trained parameters w and new task-specific parameters v.
- **Training Process:**
  - Initial parameters v_0 are set to make $w_v(x)$ resemble $w(x)$.
  - During training, only v parameters are tuned, while w remains fixed.
- **Efficiency and Expansion:**
  - Adapter tuning is parameter-efficient, requiring fewer task-specific parameters compared to fine-tuning.
  - Allows extension to new tasks without affecting previous ones, unlike multi-task and continual learning.
- **Comparison with Other Methods:**
  - Multi-task learning requires simultaneous task access, whereas adapters do not.
  - Adapters prevent forgetting previous tasks, a common issue in continual learning.
- **Performance:**
  - Adapters almost match the performance of fully fine-tuned BERT on the GLUE benchmark with only 3% task-specific parameters.
  - Similar results observed on 17 public text datasets and SQuAD extractive question answering.

Adapter-based tuning provides a single, extensible model with near state-of-the-art performance in text classification.

2.3 Adapter tuning for NLP

2.3.1 Bottleneck Architecture:

- Adopts a bottleneck architecture to limit the number of parameters.
- Original d-dimensional features are projected into a smaller dimension m, undergo nonlinearity, then projected back to d dimensions.
- Total parameters added per layer, including biases, is $2md + d + m$.
- By setting $m < d$, the number of added parameters per task is limited, typically using around 0.5-8% of the original model's parameters.
- Bottleneck dimension m allows trading off performance with parameter efficiency.

2.3.2 Skip-Connection:

- Initializing projection layer parameters near-zero initializes the module to an approximate identity function.

2.3.3 Layer Normalization Parameters:

- Trains new layer normalization parameters per task.
- Results in parameter-efficient adaptation with only $2d$ parameters per layer.

2.4 Experiments

Adapters efficiently transfer parameters for text tasks. On the GLUE benchmark, they're almost as effective as full fine-tuning of BERT, with just a 0.4% difference, while using only 3% of the parameters. This holds true across 17 other tasks. Adapter tuning automatically focuses on higher layers of the network.

2.5 Conclusions

- **Single Layer Impact**: Removing adapters from any single layer has a minimal impact on performance, with the largest performance drop being 2%.
- **Overall Effect**: However, removing all adapters substantially decreases performance, dropping to 37% on MNLI and 69% on CoLA, similar to predicting the majority class.
- **Layer Importance**: Adapters on lower layers have less impact than those on higher layers, with adapters on layers 0-4 barely affecting performance on MNLI.
- **Robustness Analysis**:
  - **Initialization Scale**: Adapters' performance is robust for initialization standard deviations below 10^{-2}. Larger initializations degrade performance, particularly on CoLA.
  - **Number of Neurons**: Performance remains stable across adapter sizes (8, 64, and 256), with small detriment to performance for fixed adapter sizes across tasks.
- **Extensions**:
  - Experimented with batch/layer normalization, increased layers per adapter, different activation functions, inserting adapters inside attention layers, adding adapters in parallel, but found no significant performance boost compared to the original bottleneck architecture.
- **Recommendation**: Due to its simplicity and strong performance, the original adapter architecture is recommended.

This comprehensive analysis highlights the importance of adapters in task adaptation and provides insights into their architecture and robustness.