Lit Review- LoRA

Problem: Many applications in natural language processing rely on adapt- ing *one* large-scale, pre-trained language model to *multiple* downstream` applications. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive.

Problems with existing solutions-

a) Methods introducing inference latency:

- Some adaptation techniques add extra layers or modules to the pre-trained model.
- While this allows for task-specific adaptation, it increases the model's depth.
- Increased depth leads to longer processing times during inference, slowing down the model's response.

b) Methods reducing usable sequence length:

- Some techniques, like prefix-tuning or prompt-tuning, add trainable tokens at the beginning of the input.
- These tokens take up part of the model's maximum input length.

c) Performance gap compared to full fine-tuning:

- Many efficient adaptation methods struggle to match the performance of fully fine-tuned models.
- This creates a trade-off: improved efficiency often comes at the cost of reduced task performance.

Low-Rank Adaptation (LoRA)

- Research has shown that despite having millions or billions of parameters, large language models often operate in a much lower-dimensional space.

b) Low "intrinsic rank" hypothesis:

  - the differences between the original and adapted weights can be approximated by low-rank matrices.

c) Rank decomposition optimization:

- Instead of directly updating the dense layers' weights, LoRA optimizes low-rank decomposition matrices.
- These matrices represent the changes to be applied to the original weights.
- The rank of these matrices is typically very small (e.g., 1 or 2) compared to the full dimensionality of the layer.

d) Frozen pre-trained weights:

- The original pre-trained model weights remain unchanged.
- This preserves the model's general knowledge while allowing task-specific adaptations.
- During inference, the low-rank matrices are used to efficiently compute the effective adapted weights.

One of the main drawbacks for full fine-tuning is that for *each* downstream task, we learn a *different* set of parameters $\Delta\Phi$ whose dimension $|\Delta\Phi|$ equals $|\Phi_0|$.

Core Idea:

- Instead of updating the entire weight matrix, LoRA represents the update as a low-rank decomposition.
- For a pre-trained weight matrix W0, the update is represented as $\Delta W = BA$, where B and A are low-rank matrices.

Implementation:

- The forward pass becomes: h = W0x + BAx
- W0 is frozen, while A and B are trainable.
- A is initialized randomly, B is initialized to zero.

No additional inference latency: W0 + BA can be precomputed for inference.

Application to Transformers:

- Typically applied to attention weights (Wq, Wk, Wv, Wo) in Transformer models.
- MLP modules are usually frozen for efficiency.

Limitations

W0 + delta W calculated once initially and then applied to all inputs. However, if we want to perform 2-3 different tasks parallely, we can't. M

- We can't use the precomputed W = W0 + BA, as each task has a different BA.
- We'd need to compute W0x + BAx separately for each task in the batch.

This makes it challenging to efficiently batch process inputs for multiple tasks simultaneously.

Tasks:

- GLUE benchmark: A collection of diverse NLU tasks including sentence classification, sentiment analysis, and textual entailment.
- WikiSQL: A task involving generating SQL queries from natural language questions.

- SAMSum: A conversation summarization task, where the goal is to create concise summaries of dialogues.

Baselines:

1) Fine-Tuning (FT):
- Updates all model parameters
- Variant: FTTop2 (fine-tuning only the last two layers)

2) Bias-only or BitFit:

- This method only updates the bias terms in the neural network.
- All other parameters (weights) are kept frozen.
- It's a very parameter-efficient approach, as bias terms are typically a small fraction of total parameters.

3) Prefix-embedding tuning (PreEmbed):

- Adds new, trainable tokens to the input sequence.
- These tokens are not part of the original vocabulary and are task-specific.
- Variants:
    - Prefixing: New tokens are added at the beginning of the input.
    - Infixing: New tokens are added after the input prompt but before the target output.

4) Prefix layer tuning??