LoRA:
Low-Rank Adaptation of Large Language Models

김성진, 신유진, 황인준

Contents

1 Problem statement

Adapting autoregressive language model $1. P_{\Phi}(y|x)$ to downstream conditional text generation tasks
ex)

1) Summarization
2) Machine Reading Comprehension
3) Natural Language to SQL (NL2SQL)

and each downstream task is represented by a training dataset of context-target pairs,
$2. \mathbb{Z} = \{(x_i, y_i)\}_{i=1, ..., N}$ where both x_i and y_i sequences of tokens

$$P_\Phi(y|x)$$

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_\Phi(y_t|x, y_{<t})\right)$$

$$P_\Phi(y|x)$$

$$P_\Phi(y|x) = \prod_{t=1}^{T} P_\Phi(y_t|x_t, y_{<t})$$

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_\Phi(y_t|x, y_{<t})\right)$$

$$P_\Phi(y|x)$$

$$P_\Phi(y|x) = \prod_{t=1}^{T} P_\Phi(y_t|x_t, y_{<t})$$

$$\begin{aligned}
\mathcal{L}(\Phi) &= -\frac{1}{N}\sum_{i=1}^{N} \log P_\Phi(y_i|x_i) \\
&= -\frac{1}{N}\sum_{i=1}^{N} \log\left(\prod_{t=1}^{T} P_\Phi(y_{i,t}|x_i, y_{i,<t})\right) \\
&= -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \log P_\Phi(y_{i,t}|x_i, y_{i,<t})
\end{aligned}$$

$$\max_\Phi \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_\Phi(y_t|x, y_{<t})\right)$$

$$P_\Phi(y|x)$$

$$P_\Phi(y|x) = \prod_{t=1}^{T} P_\Phi(y_t|x_t, y_{<t})$$

$$\begin{aligned}\Phi^* &= \arg\max_\Phi \left(-\mathcal{L}(\Phi)\right) \\ &= \arg\max_\Phi \left(\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \log P_\Phi(y_{i,t}|x_i, y_{i,<t})\right)\end{aligned}$$

$$\begin{aligned}\mathcal{L}(\Phi) &= -\frac{1}{N}\sum_{i=1}^{N} \log P_\Phi(y_i|x_i) \\ &= -\frac{1}{N}\sum_{i=1}^{N} \log \left(\prod_{t=1}^{T} P_\Phi(y_{i,t}|x_i, y_{i,<t})\right) \\ &= -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \log P_\Phi(y_{i,t}|x_i, y_{i,<t})\end{aligned}$$

$$\max_\Phi \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_\Phi(y_t|x, y_{<t})\right)$$

The model is initialized to pre-trained weights Φ_0 and updated $\Phi_0 + \triangle\Phi$ by repeatedly following the gradient to maximize the conditional language modeling objective:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi}(y_t | x, y_{<t})\right)$$

Problem: For each downstream task, we learn a different set of parameters,
If the pre-trained **model** is **large, storing and deploying** many independent instances of fine-tuned **models** can be **challenging.**

We adopt a more **parameter-efficient approach:**

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t | x, y_{<t})\right)$$

2 Aren’t existing solutions good enough?

① **Adapter Layers Introduce Latency:**
Large Neural networks rely on hardware parallelism to keep the latency low, and adapter layers have to be processed sequentially.

Batch Size Sequence Length $\|\Theta\|$	32 512 0.5M	16 256 11M	1 128 11M
Fine-Tune/LoRA	1449.4±0.8	338.0±0.6	19.8±2.7
Adapter^L	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
Adapter^H	1492.2±1.0 (+3.0%)	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)

Table 1: Infernece latency of a single forward pass in GPT-2 medium measured in milliseconds, averaged over 100 trials. We use an NVIDIA Quadro RTX8000. “$|\Theta|$” denotes the number of trainable parameters in adapter layers. Adapter^L and Adapter^H are two variants of adapter tuning, which we describe in Section 5.1. The inference latency introduced by adapter layers can be significant in an online, short-sequence-length scenario. See the full study in Appendix B.

This problems gets worse when we need to shard the model, because the additional depth requires more synchronous GPU operations.

② **Directly Optimizing the Prompt is Hard**

3 Method

3.1 Low-Rank-Parameterized Update Matrices

- hypothesis: the weight updates have a low “intrinsic rank”
- W_0 : pre-trained, frozen
- ΔW = BA : trainable
    - init. A= random Gaussian, B=0
    - scale. ΔWx by α/r
        - r is rank, α is constant in r

$$h = W_0x + \Delta W x = W_0x + BAx$$

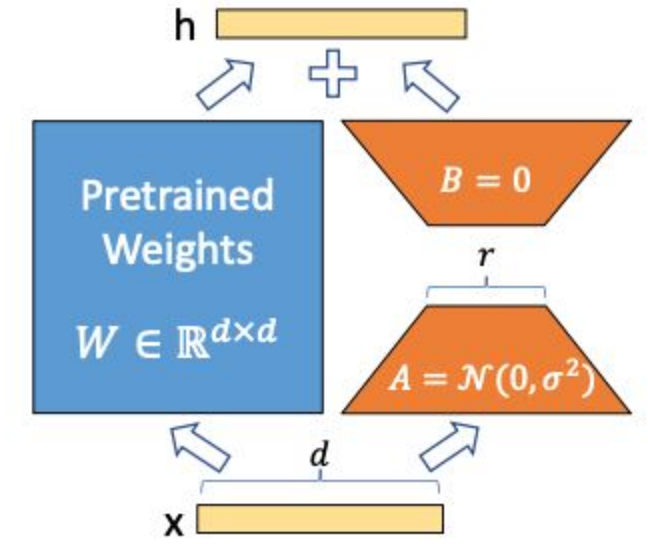

Figure 1: Our reparametrization. We only train A and B.

- Generalization of Full Fine-tuning
    - full fine-tuning by setting rank ***r***
    - cf. MLP and prefix-based are cannot take long input seq.

- No Additional Inference Latency
    - $W = W_0 + BA$
    - Able to recover W_0 by subtracting BA

3.2 Applying LoRA to Transformer

- Transformer architecture
  - Four weight matrices in the self-attention module (W_q , W_k , W_v , W_o)
  - two in the MLP module

- Adapting the attention weights only (in this study)
  - freezing the MLP modules
  - for simplicity and efficacy

- Practical Benefits
    - reduction in memory and storage usage
    - switching between tasks
        - deploying only swapping the LoRA weights (customized models)
    - result (on GPT-3 175B)
        - VRAM usage: ⅓ (1.2TB to 350GB)
        - checkpoint size: 10,000x
        - 25% speedup during training

- Limitations
    - not straightforward to batch inputs to different tasks
    - not to merge weights if latency is not critical

4 Understanding the low-rank updates

We focus our study on *GPT-3 175B*, where we achieved the largest reduction of trainable parameters (up to 10,000×) *without adversely affecting task performances*.

1) Given a parameter budget constraint, which subset of weight matrices in a pre-trained Transformer should we adapt to maximize downstream performance?

2) Is the “optimal” adaptation matrix ΔW really rank deficient? If so, what is a good rank to use in practice?

3) What is the connection between ΔW and W? Does ΔW highly correlate with W? How large is ΔW comparing to W?

1) Given a parameter budget constraint, which subset of weight matrices in a pre-trained Transformer should we adapt to maximize downstream performance?

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both W_q and W_v gives the best performance overall. We find the standard deviation across random seeds to be consistent for a given dataset, which we report in the first column.

2) Is the "optimal" adaptation matrix ΔW really rank deficient? If so, what is a good rank to use in practice?

	Weight Type	$r=1$	$r=2$	$r=4$	$r=8$	$r=64$
WikiSQL($\pm0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

2) Is the "optimal" adaptation matrix ΔW really rank deficient? If so, what is a good rank to use in practice?

$$\phi(A_{r=8}, A_{r=64}, i, j) = \frac{||U_{A_{r=8}}^{i\top} U_{A_{r=64}}^{j}||_F^2}{\min(i, j)} \in [0, 1]$$

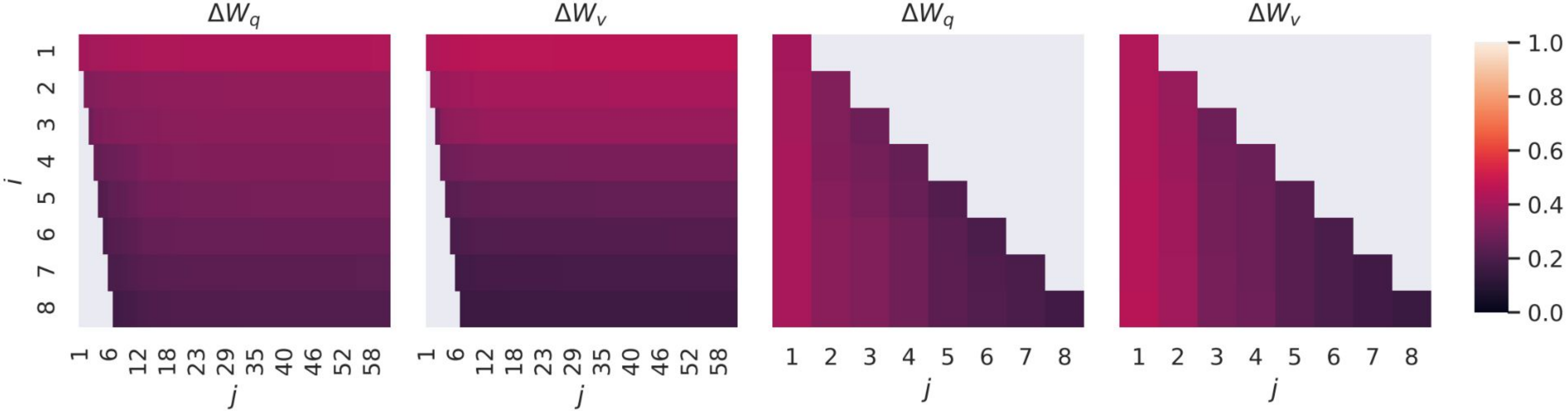

Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

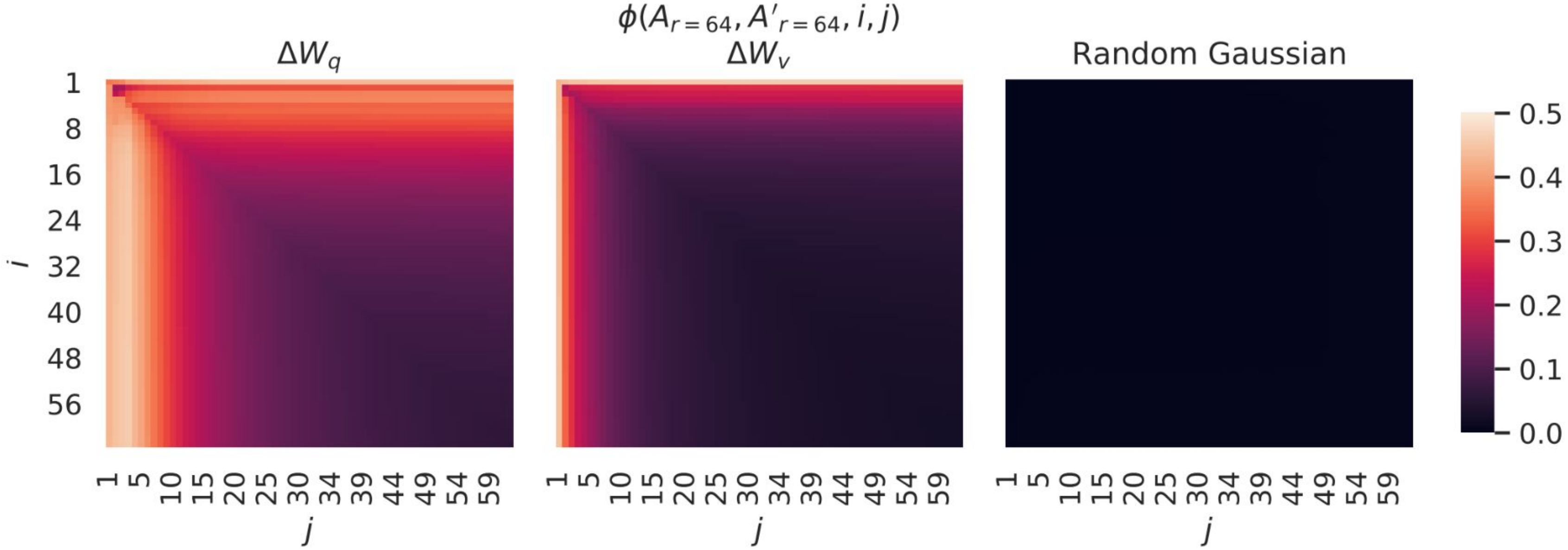

Figure 4: **Left and Middle:** Normalized subspace similarity between the column vectors of $A_{r=64}$ from two random seeds, for both ΔW_q and ΔW_v in the 48-th layer. **Right:** the same heat-map between the column vectors of two random Gaussian matrices. See Section H.1 for other layers.

3) What is the connection between ΔW and W? Does ΔW highly correlate with W? How large is ΔW comparing to W?

	$r = 4$			$r = 64$		
	ΔW_q	W_q	Random	ΔW_q	W_q	Random
$\|\|U^\top W_q V^\top\|\|_F =$	0.32	21.67	0.02	1.90	37.71	0.33
$\|\|W_q\|\|_F = 61.95$	$\|\|\Delta W_q\|\|_F = 6.91$			$\|\|\Delta W_q\|\|_F = 3.57$		

Table 7: The Frobenius norm of $U^\top W_q V^\top$ where U and V are the left/right top r singular vector directions of either (1) ΔW_q, (2) W_q, or (3) a random matrix. The weight matrices are taken from the 48th layer of GPT-3.

Q&A

Thank you

Reference

https://velog.io/@quasar529/%EB%85%BC%EB%AC%B8%EB%A6%AC%EB%B7%B0-LoRA-Low-Rank-Adaptation-of-Large-Language-Models

https://taeyuplab.tistory.com/12

Low Rank Adaptation: A Technical Deep Dive

What is low-rank adaptation (LoRA)?

Understanding LoRA with a minimal example

[LoRA 논문 리뷰] - LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

[논문리뷰] LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS