LoRA: Low-Rank Adaptation of Large Language Models

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

Microsoft, ICLR 2022

Introduction

Pre-trained and Fine-tuning

Many applications in natural language processing rely on applying a large-scale, pre-trained language model to multiple downstream applications through fine-tuning.

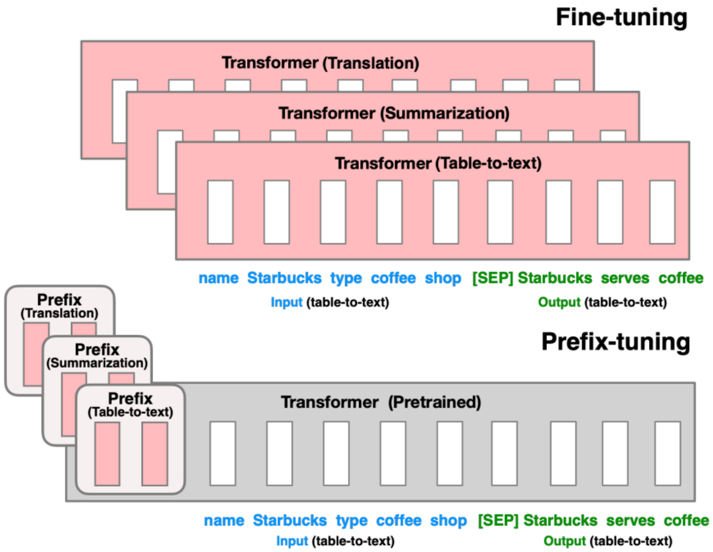
However, the fine-tuned model according to different tasks has the same size parameters as the pre-trained model, This requires a lot of space to store the models of these specialized tasks.

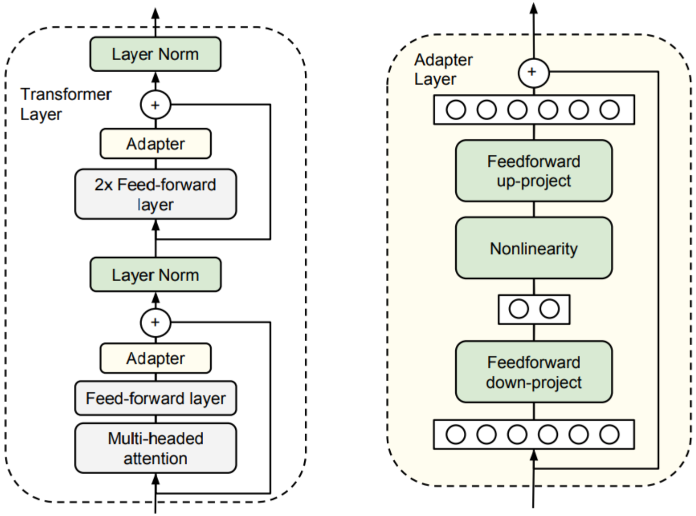
Introduction (cont.)

Existing Solution

Many sought to mitigate this by adapting only some parameters or learning external modules for new tasks.

e.g., Bias-only Tuning, Prefix-tuninig, Adapter Layer.





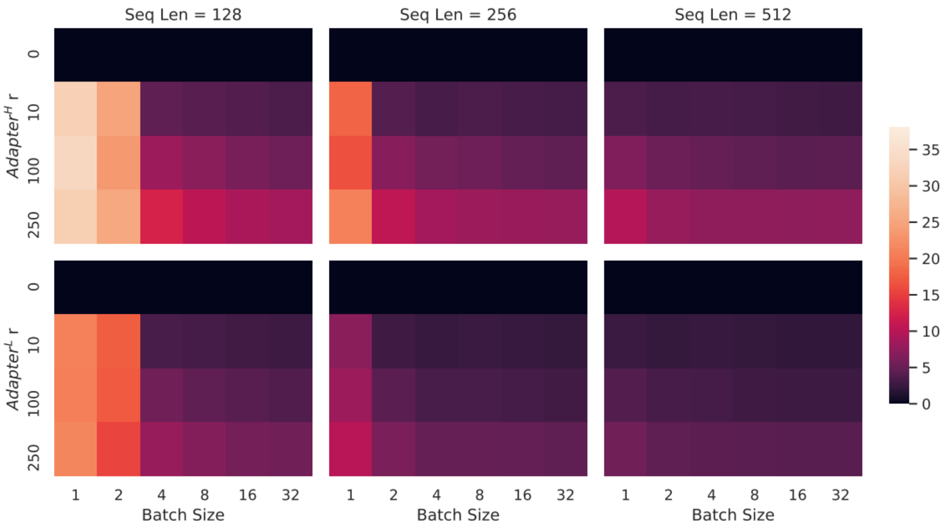
Introduction (cont.)

Aren’t Existing Solutions Good Enough ?

Nevertheless, techniques prior to LoRA often introduce

- inference latency by extending model depth,
- reduce the model’s usable sequence length.

more importantly, these methods often fail to match the fine-tuning baselines, posing a trade-off between efficiency and model quality.

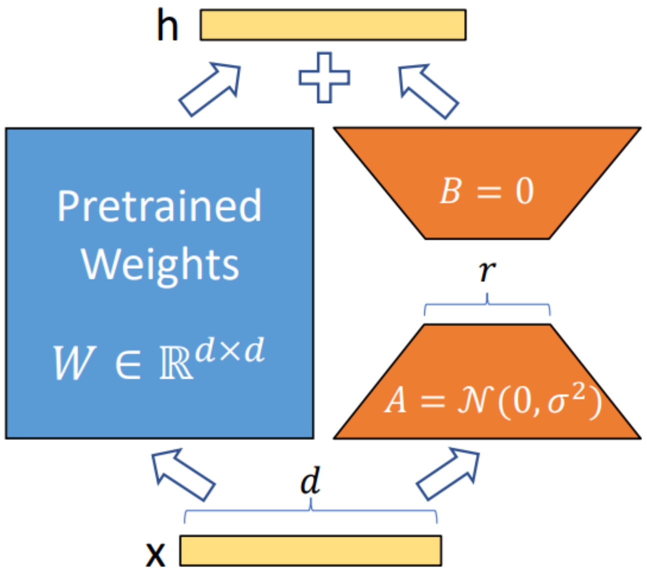


When the generation length is shorter and the batch is smaller, the additional computational cost of the Adapter Layer is more obvious.

Low Rank Adaptation

Write the weight matrix of fine-tuning in the form of $W + \Delta W$, and $W, \Delta W \in \mathbb{R}^{d \times k}$; W is the pre-trained weight matrix.

LoRA constrain finr-tuned update ΔW by representing the latter with a low-rank decomposition $\Delta W = \frac{\alpha}{r} BA^{\top}, A \in \mathbb{R}^{k \times r}$, $B \in \mathbb{R}^{d \times r}$ and $r \ll min(d, k)$, α is a constant scalar.



Low Rank Adaptation (cont.)

ΔW of LoRA = $BA^\top$

- LoRA's trainable parameters are much smaller than full fine-tuning due to $r \ll d$.
- A Generalization of Full Fine-tuning.

  Roughly recover the expressiveness of full fine-tuning by setting the LoRA rank r to the rank of the pre-trained weight matrices.
- No Additional Inference Latency.

  Adding the pre-training weights W to the LoRA parameters $BA^\top$ first will not increase the amount of calculation during inference.

Experiment

Comparing the performance on GPT-3

Model&Method	# Trainable Parameters	WikiSQL	MNLI-m	SAMSum
		Acc. (%)	Acc. (%)	R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^{H})	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^{H})	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

In the experiments of this paper, only the attention layer adds LoRA parameters.

the Optimal Rank r for LoRA

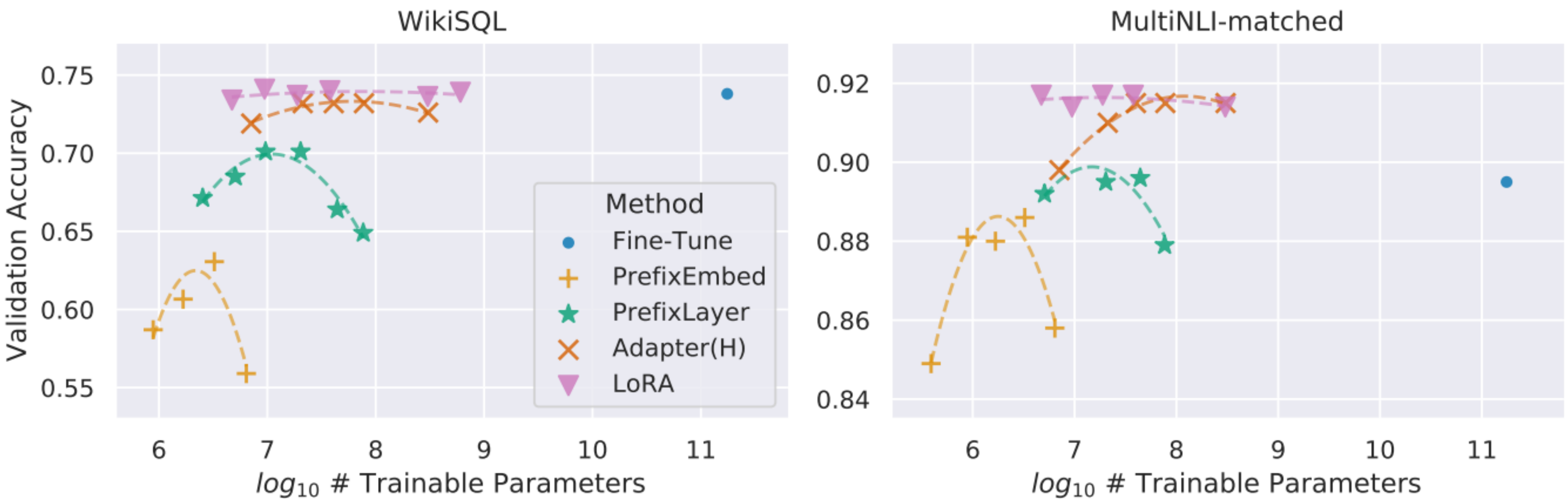
GPT-3

	# of Trainable Parameters = 18M						
Weight Type Rank r	W_q 8	W_k 8	W_v 8	W_o 8	W_q, W_k 4	W_q, W_v 4	W_q, W_k, W_v, W_o 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**

GPT-2

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	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 ($\pm 0.1\%$)	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

Experiment (cont.)



- Prefix-tuning is difficult to optimize and that its performance changes non-monotonically in trainable parameters.
- LoRA has stable performance under various number of parameters.

Experiment (cont.)

Low-Data Regime

Method	MNLI(m)-100	MNLI(m)-1k	MNLI(m)-10k	MNLI(m)-392K
GPT-3 (Fine-Tune)	60.2	**85.8**	88.9	89.5
GPT-3 (PrefixEmbed)	37.6	75.2	79.5	88.6
GPT-3 (PrefixLayer)	48.3	82.5	85.9	89.6
GPT-3 (LoRA)	**63.8**	85.6	**89.2**	**91.7**

Compared with the Prefix-tuning method with obvious performance degradation, LoRA still shows good performance with only a small amount of data.

Experiment (cont.)

Combined with Other Adaptation Methods

Method	Hyperparameters	# Trainable Parameters	WikiSQL	MNLI-m
Fine-Tune	-	175B	73.8	89.5
LoRA	$r_v = 2$	4.7 M	73.4	**91.7**
	$r_q = r_v = 1$	4.7 M	73.4	91.3
	$r_q = r_v = 2$	9.4 M	73.3	91.4
	$r_q = r_k = r_v = r_o = 1$	9.4 M	74.1	91.2
	$r_q = r_v = 4$	18.8 M	73.7	91.3
	$r_q = r_k = r_v = r_o = 2$	18.8 M	73.7	**91.7**
	$r_q = r_v = 8$	37.7 M	73.8	**91.6**
	$r_q = r_k = r_v = r_o = 4$	37.7 M	74.0	**91.7**
	$r_q = r_v = 64$	301.9 M	73.6	91.4
	$r_q = r_k = r_v = r_o = 64$	603.8 M	73.9	91.4
LoRA+PE	$r_q = r_v = 8, l_p = 8, l_i = 4$	37.8 M	75.0	91.4
	$r_q = r_v = 32, l_p = 8, l_i = 4$	151.1 M	**75.9**	91.1
	$r_q = r_v = 64, l_p = 8, l_i = 4$	302.1 M	**76.2**	91.3

Conclusions

This study proposes LoRA to replace fine-tuning

- With the same pre-trained weights, only a small number of LoRA parameters need to be saved for different tasks.
- That neither introduces inference latency nor reduces input sequence length while retaining high model quality.
- Can be combined with other efficient adaptation methods, potentially providing orthogonal improvement.

Next Version

AdaLoRA, which adaptively allocates the parameter budget among weight matrices according to their importance score.