

# LoRA Review

Paper: Low-Rank Adaptation of Large Language Models

Sangho Kim

# Contents

- Introduction
- Problem Statement
- Existing Solutions
- Method
- Experimental Results
- Conclusion

# Introduction

- 최근 GPT와 같은 LLM 또는 Foundation model이 등장함에 따라 task-specific model을 위한 fine-tuning 방법이 소개됨
- 이때 full fine-tuning을 수행하게 되면 무수히 많은 GPU가 필요
- 한정된 하드웨어로 fine-tuning을 원활히 수행하고 충분히 좋은 성능의 모델을 뽑아내기 위해 LoRA를 소개
- 간단히 말해, pre-trained weight는 freezing하고 trainable rank decomposition matrices를 network 중간에 삽입
- 이후 기존 모델에 비해 훨씬 더 적은 parameter를 학습
- 논문에서는 GPT-3와 같은 original model에 비해 trainable parameter가 약 10,000배 줄어든다고 소개
- 또한 추가적인 inference latency가 없다는 장점도 존재
- 다양한 LLM model에 LoRA를 적용함에도 불구하고 비슷하거나 오히려 더 좋은 성능을 달성하는 실험 결과도 존재

# Introduction

- LoRA의 구조를 간단하게 그리면 오른쪽 그림과 같음
- 즉, SVD의 개념처럼 Decomposition Low Rank Matrix를 삽입
- 이때  $B$ 는 0으로 initialization
- $A$ 는  $N(0, \sigma^2)$ 를 따르는 random variable로 initialization

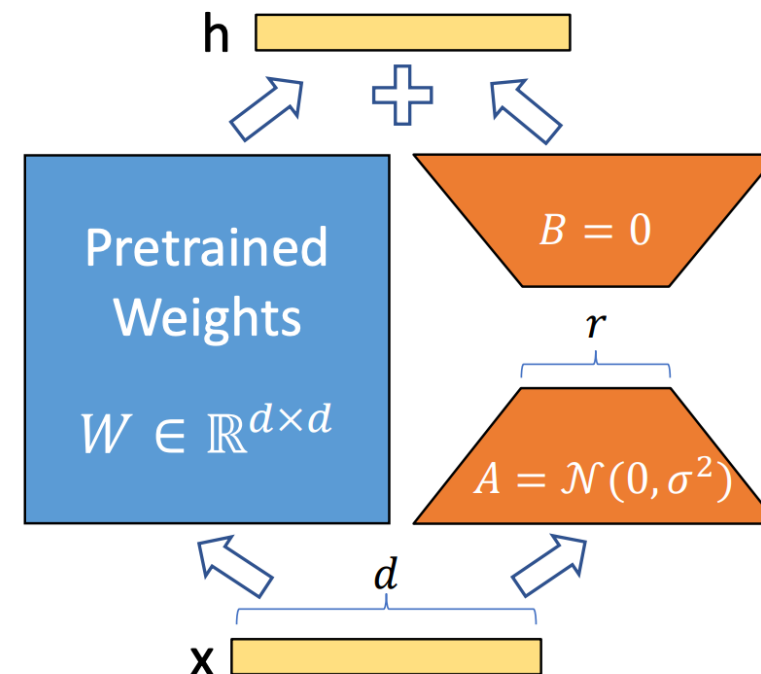


Figure 1: Our reparametrization. We only train  $A$  and  $B$ .

# Contents

- Introduction
- Problem Statement
- Existing Solutions
- Method
- Experimental Results
- Conclusion

# Problem Statement

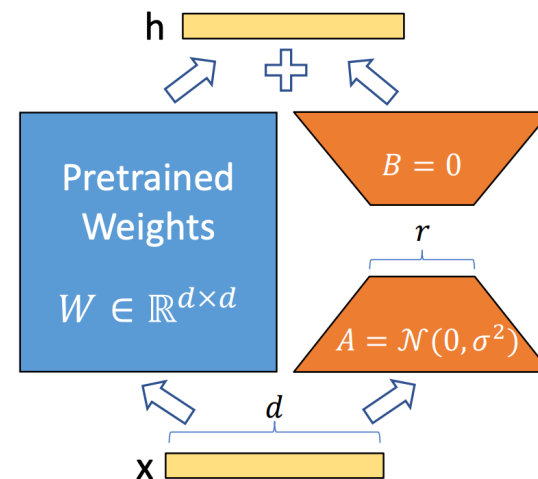
- 먼저, pre-trained weight인  $\Phi$ 가 존재하는 model  $P_{\Phi}(y|x)$ 가 주어지며 이는 GPT와 같이 LLM으로 간주
- 이후, downstream task에 맞춰 tuning하기 위한 context-target pairs가 필요
- 이를  $Z = \{(x_i, y_i)\}_{i=1, \dots, N}$ 이라 가정, 이때  $x_i, y_i$ 는 sequences of tokens
- 만약 full fine-tuning을 수행하게 된다면 pre-trained weight  $\Phi_0$ 와 추가적인 weight  $\Delta\Phi$ 를 가지고 initialization
- 즉, 식으로 표현하면 다음과 같음

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

- 그러나 LoRA는 다음의 objective를 설정 가능

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

- 즉, pre-trained weight인  $\Phi_0$ 는 freezing,  $\Delta\Phi$ 만 학습 (오른쪽 그림에서 주황색 부분)



# Contents

- Introduction
- Problem Statement
- Existing Solutions
- Method
- Experimental Results
- Conclusion

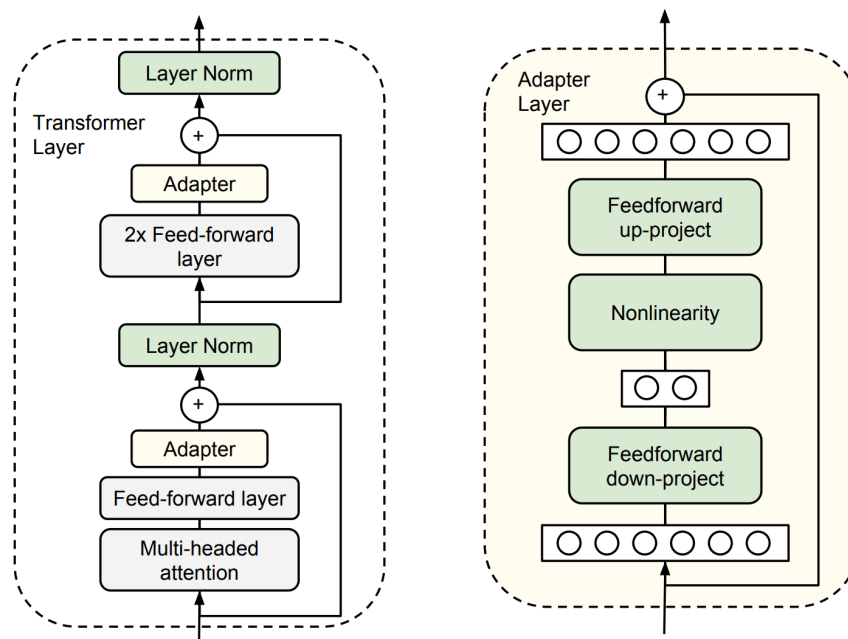
# Existing Solutions

- 본 연구에서 해결하고자 하는 문제에 대한 기존 방법론들은 크게 다음 2가지로 나뉘어짐
  - 1. adding adapter layer
  - 2. optimizing some forms of the input layer activation



# Existing Solutions

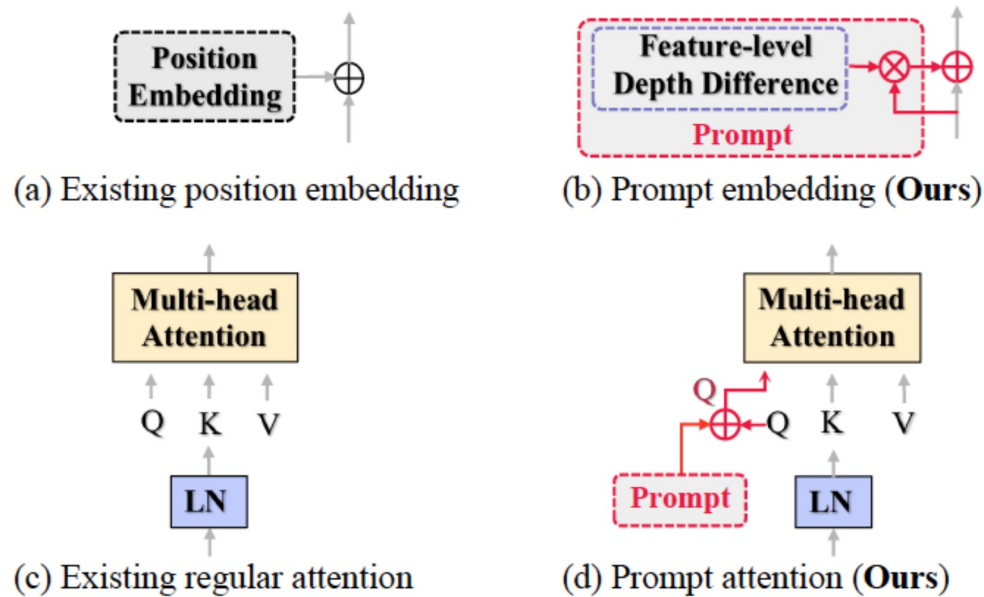
- 이 중 1번 방법론은 다음 그림과 같음



- 즉, Transformer Block 안에 작은 사이즈의 trainable module을 추가해 fine-tuning하는 방법

# Existing Solutions

- 2번 방법론은 다음 그림과 같음



- 즉, input embedding에 prompt embedding을 추가하고 이를 다양한 방법으로 학습해 fine-tuning 대체

# Existing Solutions

- 그러나 1번 방법론은 다음의 문제점을 가지고 있음
- Transformer는 Parallelism에 의존하는데, 1번 스타일의 Adapter를 적용하면 **additional computing + bottleneck** 발생
- 이에 따른 실험은 다음 table에서 볼 수 있음

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

Table 1: Inference 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<sup>L</sup> and Adapter<sup>H</sup> 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.

# Existing Solutions

- 2번 방법론도 문제점을 가지고 있음
- 이 방법론의 대표적인 예로 prefix tuning이 존재하는데 이는 optimize하기 어려움
- 성능 또한 trainable parameter에 따라 단조적으로 증가하지 않는다는 문제점도 존재

# Contents

- Introduction
- Problem Statement
- Existing Solutions
- Method
- Experimental Results
- Conclusion

# Method

- 이제 본격적으로 LoRA에 대해 자세히 알아보자
- $W_0 \in R^{d \times k}$ 는 neural network의 pre-trained weight (freezing parameters)
- LoRA에 해당하는 adapter는  $\Delta W = BA$ 이며 이때  $B \in R^{d \times r}, A \in R^{r \times k}$  (trainable parameters)
- 여기서  $r \ll \min(d, k)$ 은 hyperparameter로 LoRA의 rank를 결정
- $x$ 를 input,  $h$ 를 LoRA가 탑재된 network module의 output이라고 하면 다음과 같이 쓸 수 있음

$$h = W_0x + \Delta Wx = W_0x + BAx$$

- 여기서  $\Delta Wx$ 는  $\frac{\alpha}{r}$ 에 의해 scaling

# Method

- Generalization of Full Fine-tuning
  - LoRA를 적용해도 full fine-tuning을 할 때만큼의 expressiveness를 가질 수 있음
- 앞에서 소개했던 기존 2가지 방법론은 문제를 야기
  - 1. Adapter-based method:
    - 추가적인 MLP layer 학습 필요
  - 2. Prefix-based method:
    - input에 prompt embedding을 추가해야 하기 때문에 input sequence를 추가하는 데 한계가 존재
- No Additional Inference Latency
  - 특정 task에 적용하기 위해  $BA$  matrix를 fine-tuning한 후 다른 task에 적용하고 싶다면 새로운  $BA$  matrix를 fine-tuning을 수행하면 된다는 장점
  - 이 때문에 no additional inference latency라고 표현한 듯 하다

# Method

- 이 논문에서는 attention module에만 LoRA를 적용
- LoRA의 가장 큰 장점은 trainable parameter의 수가 적다는 것 + efficient memory storage usage
- VRAM을 최대 2/3까지 줄일 수 있으며  $r \ll d_{model}$  이어야 함
- GPT-3를 예로 들자면
  - 원래 GPT-3 (175B)를 학습하는 데 필요한 VRAM은 1.2TB
  - Attention Module에 LoRA를 적용하면 350GB까지 감소 가능
  - 또한  $r = 4$ 로 설정하고 query, value projection matrices에만 적용한다면 35MB까지 감소
  - 이를 통해 25%의 speedup을 유도
- LoRA의 한계점
  - 다른 downstream task에 적용할 때마다  $BA$ 를 바꿔줘야 한다는 것

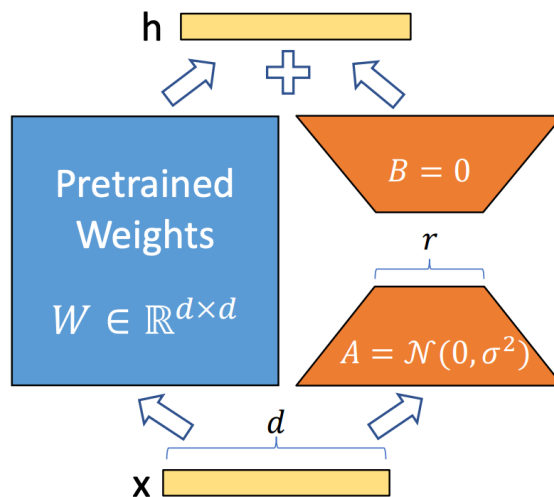


# Method

- LoRA가 왜 엄청나게 효율적인 모델인지 살펴보자
- 먼저 LoRA를 식으로 표현하면 다음과 같다

$$h = W_0x + \Delta Wx = W_0x + BAx$$

- 이때  $BA$  부분만 학습을 한 후,  $(W_0 + BA)x = W_{new}x$ 로 나타낼 수 있음
- 즉, 새로 학습한 weight는  $W_{new}$ 로 표현할 수 있고 이때 모델의 parameter 수는 변함이 없음
- 또한 다양한 task에 대해 fine-tuning하여 얻은 여러  $BA$ 들을 가지고 적재적소에 활용 가능



# Contents

- Introduction
- Problem Statement
- Existing Solutions
- Method
- Experimental Results
- Conclusion

# Experimental Results

- 실험 내용들을 보기 전에 몇 가지 notation을 정리하고 가자
  - $FT$  : Fine-Tuning
  - $FT^{Top2}$  : 마지막 2개의 layer만 Tuning
  - BitFit (Bias-only) : bias vector만 training
  - Prefix-embedding tuning : input token 사이에 special token을 삽입
  - Prefix-layer tuning : 논문에서는 we learn the activations after every Transformer Layer라고 표현을 했지만 나름 내가 이해하기로는 "activation을 취한 이후의 output을 scaling한다는 것 아닐까? 추측한다"
  - Adapter tuning
    - $Adapter^H$ : Self-attention, MLP, Resudial connection 이후에 adapter layer 탑재
    - $Adapter^P$ : MLP, LayerNorm 이후에만 적용
    - $Adapter^D$ : Dropout concept을 활용한 Adapter Layer

# Experimental Results

- 다양한 adaptation method를 GLUE benchmark에 실험한 결과

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB <sub>base</sub> (FT)*	125.0M	<b>87.6</b>	94.8	90.2	<b>63.6</b>	92.8	<b>91.9</b>	78.7	91.2	86.4
RoB <sub>base</sub> (BitFit)*	0.1M	84.7	93.7	<b>92.7</b>	62.0	91.8	84.0	81.5	90.8	85.2
RoB <sub>base</sub> (Adpt <sup>D</sup> )*	0.3M	87.1 $\pm$ .0	94.2 $\pm$ .1	88.5 $\pm$ 1.1	60.8 $\pm$ .4	93.1 $\pm$ .1	90.2 $\pm$ .0	71.5 $\pm$ 2.7	89.7 $\pm$ .3	84.4
RoB <sub>base</sub> (Adpt <sup>D</sup> )*	0.9M	87.3 $\pm$ .1	94.7 $\pm$ .3	88.4 $\pm$ .1	62.6 $\pm$ .9	93.0 $\pm$ .2	90.6 $\pm$ .0	75.9 $\pm$ 2.2	90.3 $\pm$ .1	85.4
RoB <sub>base</sub> (LoRA)	0.3M	87.5 $\pm$ .3	<b>95.1<math>\pm</math>.2</b>	89.7 $\pm$ .7	63.4 $\pm$ 1.2	<b>93.3<math>\pm</math>.3</b>	90.8 $\pm$ .1	<b>86.6<math>\pm</math>.7</b>	<b>91.5<math>\pm</math>.2</b>	<b>87.2</b>
RoB <sub>large</sub> (FT)*	355.0M	90.2	<b>96.4</b>	<b>90.9</b>	68.0	94.7	<b>92.2</b>	86.6	92.4	88.9
RoB <sub>large</sub> (LoRA)	0.8M	<b>90.6<math>\pm</math>.2</b>	96.2 $\pm$ .5	<b>90.9<math>\pm</math>1.2</b>	<b>68.2<math>\pm</math>1.9</b>	<b>94.9<math>\pm</math>.3</b>	91.6 $\pm$ .1	<b>87.4<math>\pm</math>2.5</b>	<b>92.6<math>\pm</math>.2</b>	<b>89.0</b>
RoB <sub>large</sub> (Adpt <sup>P</sup> )†	3.0M	90.2 $\pm$ .3	96.1 $\pm$ .3	90.2 $\pm$ .7	<b>68.3<math>\pm</math>1.0</b>	<b>94.8<math>\pm</math>.2</b>	<b>91.9<math>\pm</math>.1</b>	83.8 $\pm$ 2.9	92.1 $\pm$ .7	88.4
RoB <sub>large</sub> (Adpt <sup>P</sup> )†	0.8M	<b>90.5<math>\pm</math>.3</b>	<b>96.6<math>\pm</math>.2</b>	89.7 $\pm$ 1.2	67.8 $\pm$ 2.5	<b>94.8<math>\pm</math>.3</b>	91.7 $\pm$ .2	80.1 $\pm$ 2.9	91.9 $\pm$ .4	87.9
RoB <sub>large</sub> (Adpt <sup>H</sup> )†	6.0M	89.9 $\pm$ .5	96.2 $\pm$ .3	88.7 $\pm$ 2.9	66.5 $\pm$ 4.4	94.7 $\pm$ .2	92.1 $\pm$ .1	83.4 $\pm$ 1.1	91.0 $\pm$ 1.7	87.8
RoB <sub>large</sub> (Adpt <sup>H</sup> )†	0.8M	90.3 $\pm$ .3	96.3 $\pm$ .5	87.7 $\pm$ 1.7	66.3 $\pm$ 2.0	94.7 $\pm$ .2	91.5 $\pm$ .1	72.9 $\pm$ 2.9	91.5 $\pm$ .5	86.4
RoB <sub>large</sub> (LoRA)†	0.8M	<b>90.6<math>\pm</math>.2</b>	96.2 $\pm$ .5	<b>90.2<math>\pm</math>1.0</b>	68.2 $\pm$ 1.9	<b>94.8<math>\pm</math>.3</b>	91.6 $\pm$ .2	<b>85.2<math>\pm</math>1.1</b>	<b>92.3<math>\pm</math>.5</b>	<b>88.6</b>
DeB <sub>XXL</sub> (FT)*	1500.0M	91.8	<b>97.2</b>	92.0	72.0	<b>96.0</b>	92.7	93.9	92.9	91.1
DeB <sub>XXL</sub> (LoRA)	4.7M	<b>91.9<math>\pm</math>.2</b>	96.9 $\pm$ .2	<b>92.6<math>\pm</math>.6</b>	<b>72.4<math>\pm</math>1.1</b>	<b>96.0<math>\pm</math>.1</b>	<b>92.9<math>\pm</math>.1</b>	<b>94.9<math>\pm</math>.4</b>	<b>93.0<math>\pm</math>.2</b>	<b>91.3</b>

- LoRA는 다른 method들과 비슷하거나 오히려 더 성능이 좋은 경우가 많음
- 여기서 주목할 만한 점은 trainable parameter의 갯수이다

# Experimental Results

- E2E NLG Challenge에서 GPT-2를 가지고 다양한 adapter를 적용한 실험

Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter <sup>L</sup> )*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter <sup>L</sup> )*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter <sup>H</sup> )	11.09M	67.3 $\pm$ .6	8.50 $\pm$ .07	46.0 $\pm$ .2	70.7 $\pm$ .2	2.44 $\pm$ .01
GPT-2 M (FT <sup>Top2</sup> )*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	<b>70.4<math>\pm</math>.1</b>	<b>8.85<math>\pm</math>.02</b>	<b>46.8<math>\pm</math>.2</b>	<b>71.8<math>\pm</math>.1</b>	<b>2.53<math>\pm</math>.02</b>
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter <sup>L</sup> )	0.88M	69.1 $\pm$ .1	8.68 $\pm$ .03	46.3 $\pm$ .0	71.4 $\pm$ .2	<b>2.49<math>\pm</math>.0</b>
GPT-2 L (Adapter <sup>L</sup> )	23.00M	68.9 $\pm$ .3	8.70 $\pm$ .04	46.1 $\pm$ .1	71.3 $\pm$ .2	2.45 $\pm$ .02
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	<b>70.4<math>\pm</math>.1</b>	<b>8.89<math>\pm</math>.02</b>	<b>46.8<math>\pm</math>.2</b>	<b>72.0<math>\pm</math>.2</b>	2.47 $\pm$ .02

- 이 실험에서는 LoRA가 대부분의 실험에서 근소하게 가장 좋은 성능을 보여주고 있음
- 여기서도 역시 trainable parameter 수에 비해 좋은 성능을 뽑아내는 엄청난 효율성을 보여줌

# Experimental Results

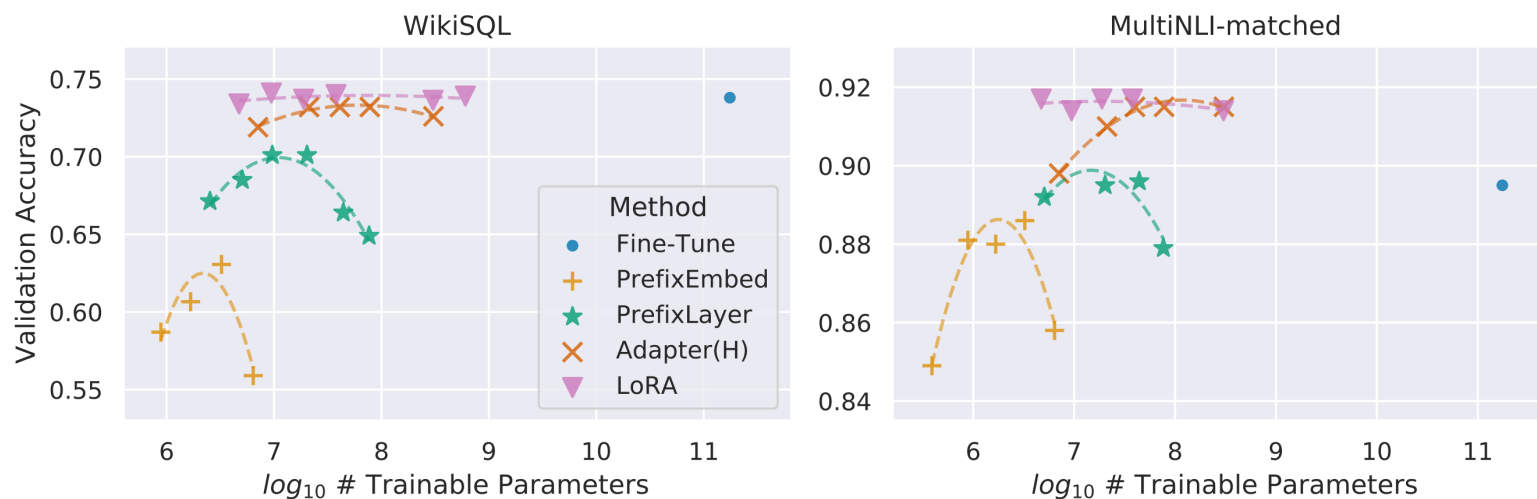
- WkikSQL에서 GPT-3를 가지고 다양한 adapter를 적용한 실험

Model&Method	# Trainable Parameters	WikiSQL	MNLI-m	SAMSum
		Acc. (%)	Acc. (%)	R1/R2/RL
GPT-3 (FT)	175,255.8M	<b>73.8</b>	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 <sup>H</sup> )	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter <sup>H</sup> )	40.1M	73.2	<b>91.5</b>	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	<b>91.7</b>	<b>53.8/29.8/45.9</b>
GPT-3 (LoRA)	37.7M	<b>74.0</b>	<b>91.6</b>	53.4/29.2/45.1

- 이 실험에서도 마찬가지로 LoRA는 다른 method와 비교해도 뒤지지 않는 성능 보유

# Experimental Results

- GPT-3에서 각 benchmark를 가지고 trainable parameter 수에 따른 성능 비교 실험



- 다른 method들은 trainable parameter의 수가 특정 임계값을 넘으면 성능이 떨어진다
- 반면 LoRA는 robust한 성능을 유지

# Contents

- Introduction
- Problem Statement
- Existing Solutions
- Method
- Experimental Results
- Conclusion



# Conclusion

- 넉넉하지 않은 GPU를 갖고 LLM에서 downstream task에 적용하기 위한 method인 LoRA를 소개
  - 다양한 실험을 통해 LoRA의 엄청난 효율성과 성능을 입증
  - 특히, GPT-3의 경우, trainable parameter의 수를 10,000배까지 감소
- 
- 개인적 의견
    - Low Rank Decomposition이라는 간단한 아이디어로 누구나 LLM을 Fine-tuning할 수 있게 만들었던 것이 대단하다
    - 아마 내가 읽은 딥러닝 논문 중 다섯 손가락 안에 들 정도로 굉장한 논문이지 않을까 생각한다
    - 오랜만에 논문다운 논문을 읽은 것 같다