LoRA: Low-Rank Adaptation of Large Language Models 논문 리뷰

1. Introduction

- 최근의 NLP: **하나**의 pre-trained model → **여러 개**의 downstream application에 적용
- **fine-tuning**: pre-trained model의 '모든' 파라미터를 특정 작업에 맞게 업데이트

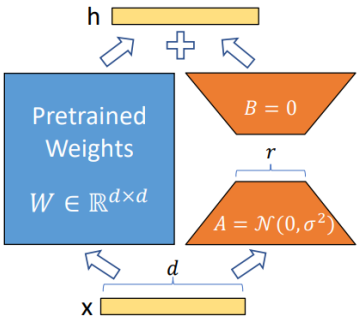
  → 기존 모델의 파라미터 개수만큼의 훈련이 필요
- 파라미터의 일부만 adapt하거나, external module을 활용하는 방법에 대한 연구들이 이루어지고 있음

  [장점]

  1. task-specific한 파라미터만 필요로 함
  2. efficiency 향상

  [한계]

  1. inference latency 발생
  2. model이 사용 가능한 sequence length ↓
- **LoRA(Low-Rank Adaption)**의 등장



  - chang in wieghts는 '**low intrinsic rank**'를 가진다고 가정 (*reside on a low intrinsic dimension*)

    : 가중치 변화량은 높은 차원을 가진 행렬이지만, 모델 학습을 위해 필요한 **핵심적인(실질적인) 정보는 낮은 차원으로 압축**될 수 있음
  - **pre-trained weights는 고정**하고, **rank decomposition matrices(A, B)만 최적화**하는 방식
  - LoRA의 장점
    1. 그림에서의 행렬 A, B만 바꿔가면서 서로 다른 task를 위한 LoRA module 쉽게 설계 가능
    2. 효율적이며 하드웨어 장벽을 낮추는 학습 방식
    3. simple한 linear design으로, inference latency 도입 x
    4. 기존 방법(ex. prefix-tuning)들과 orthogonal → 독립적으로 결합 가능

---

2. Problem Statement

[Language Modeling Problem의 정의]

- pre-trained language model $P_\Phi(y \mid x)$가 주어짐
- full fine-tuning의 objective:

- $\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi}(y_t \mid x, y_{<t})\right)$
- Φ_0(pre-trained 모델의 파라미터)에서 시작하여, $\Phi_0 + \Delta\Phi$로 업데이트
- 이 때, $|\Phi_0| = |\Delta\Phi|$라는 한계가 존재

- LoRA의 objectvie:
  - $\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log p_{\Phi_0+\Delta\Phi(\Theta)}\left(y_t \mid x, y_{<t}\right)$
  - Φ_0는 고정, $\Delta\Phi = \Delta\Phi(\Theta)$는 Θ를 학습함으로써 계산됨
  - 이 때, $|\Theta| \ll |\Phi_0|$로, **much smaller-sized** set of parameters만 사용됨

---

3. Aren't Existing Solutions Good Enough?

: 기존의 efficient fine-tuning 방법들과 그들의 한계

1. **Adater Layer**를 추가하는 방식
   - Transformer block마다 특정 개수의 adapter layer를 추가해서 이것들만 학습시키는 방식
   - adapter layer는 few parameter를 가지지만, '**sequentially**' processed 되어야 함
   - large neural network는 하드웨어 병렬 처리에 의존하지만, adapter layer는 **불가능 → inference latency 발생**

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
$\text{Adapter}^{\text{L}}$	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
$\text{Adapter}^{\text{H}}$	1492.2±1.0 (+3.0%)	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)

2. **Prefix Tuning**
   - adaptation을 위한 sequence length를 필요로 함
   - 따라서, downstream task를 위해 **사용 가능한 sequence length 감소**
   - 이로 인해, prompt를 tuning하는 것은 less performant

---

4. Method

4.1 Low-Rank-Parametrized Update Matrices

✅ 알려진 사실: pre-trained language models have a **low instrisic dimension**

💡 가정: the updates to the weights also have a **low intrinsic rank** during adaptation

$$h = W_0 x + \Delta W\, x = W_0 x + BAx$$
$$W_0 \in \mathbb{R}^{d\times k},\ B \in \mathbb{R}^{d\times r},\ A \in \mathbb{R}^{r\times k}$$

- rank $r \ll min(d, k)$ → **low-rank decomposition**: $\Delta W = BA$로 계산

- A는 random Gaussian 분포로 초기화, B는 0으로 초기화 → 초기 BA는 0으로 시작
- W_0와 BA 각각에 동일한 input x를 곱함 → coordinate-wise sum으로 최종 h 계산

[A Generalization of Full Fine-tuning]

- **LoRA**: rank r을 increase 할수록 → full fine-tuning의 성능에 수렴함
- 기존 adapter-based 방법: MLP의 성능에 수렴
- prefix-based 방법: 길이가 긴 input sequence를 받을 수 없는 모델에 수렴

[No Additional Inference Latency]

- $W = W_0 + BA$로 계산되므로, BA를 빼서 W_0 복원 가능
- 그 후, 다른 decomposition인 $B'A'$을 더하는 간단하고 빠른 방법으로 다른 downstream task로 전환
- 따라서 memory 차지도 적으며, 어떠한 **additional inference latency도 추가되지 않음**

4.2 Applying LoRA to Transformer

- LoRA는 신경망의 weight matrices의 어떤 subset이든 적용 가능
- Transformer의 경우, 아래와 같이 downstream task에 적용
  - MLP module은 고정 → simplicity & efficiency
  - **attention weights(W_q or W_k or W_v)에만** LoRA 적용
- 이를 통해, 기존에 175B 개의 파라미터를 가진 GPT-3에 대해, VRAM 소비량이 1.2TB에서 350GB로 감소

---

5. Empirical Experiments

- LoRA 적용 모델: RoBERAa(base, large), DeBERTa(XXL), GPT-2(medium, large), GPT-3 175B
- 실험 결과

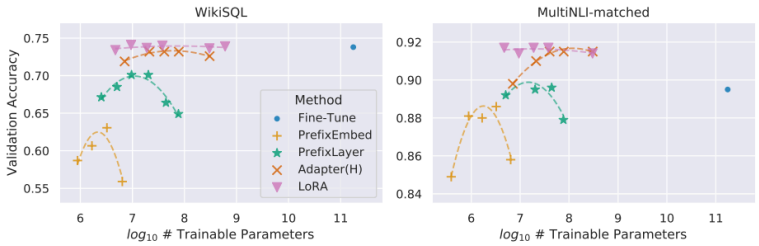
Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

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 ($\text{Adapter}^{\text{L}}$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($\text{Adapter}^{\text{H}}$)	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^{Top2})*	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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($\text{Adapter}^{\text{L}}$)	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L ($\text{Adapter}^{\text{L}}$)	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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum 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 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	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

- 대부분의 task에서 더 적은 개수의 trainable parameter만으로도 LoRA가 높은 성능을 보임

- 심지어, 많은 경우에 Full Fine-tuning(FT)보다 좋은 성능을 보임
- 특히 GPT-3에서 LoRA를 이용했을 때 accuracy가 다른 method들에 비해 훨씬 높고 안정적이었음



---

6. Related Works

- Transformer Language Models
- Prompt Engineering and Fine-Tuning
- Parameter-Efficient Adaptation
- Low-Rank Structures in Deep Learning

---

7. Understanding the Low-Rank Updates

7.1 Which weight matrices in Transformer should we apply LoRA to?

	# 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 (±0.5%)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI (±0.1%)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

- **W_q와 W_v에 같이 LoRA를 적용**할 때 업데이트 할 가중치를 최소화하면서도 가장 좋은 성능을 보였음
- weight matrix 중 한 가지에만 LoRA를 적용하는 것보다 여러 개에 동시에 적용하는 것이 성능이 더 좋음

7.2 What is the optimal rank r for LoRA?

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL(±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 (±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

- 매우 작은 r(1 or 2)에서도 좋은 성능을 보임

  → 'ΔW가 **very small intrinsic rank**를 가질 수 있음'을 시사
- r을 증가시킨다고 해서 항상 더 많은 meaningful subspace를 cover할 수 있는 것은 아님!

  → **low-rank** adaptation matrix만으로도 충분함

7.3 How does the adaptation matrix ΔW compare 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
$\|\|W_q\|\|_F = 61.95$	$\|\|\Delta W_q\|\|_F = 6.91$			$\|\|\Delta W_q\|\|_F = 3.57$		

- random matrix와 비교했을 때, ΔW가 W와 더 강한 상관관계를 지님

  → ΔW가 W의 feature 일부를 증폭시킴
- ΔW는 W에서 강조(설명)되지 않은 방향을 향하고 있음
- ΔW의 amplification factor 값은 매우 큼

> 📌 low-rank adaptaion matrix BA가 [기존 사전학습된 모델의 W에서는 강조되지 않았지만,
> 특정 downstream task에서는 **중요한 feature**]를 **증폭**시키는 역할을 함!

---

8. Conclusion and Future Work

- LoRA 정리
  - efficient한 adaptation strategy
  - inference latency를 도입하지 않음
  - input sequence length가 감소하지 않음
  - 서로 다른 각각의 downstream task에 맞게 빠른 switch 가능