LoRA (Low-Rank Adaptation of LLMs)

길종현(github.com/hyeon-n-off)

개요

NLP 에서 중요한 패러다임은 일반적인 도메인(general domain) 데이터와 특정 태스크나 도메인(particular tasks or domains)에 적응하는 사전학습(pre-training)으로 구성된다.

우리가 더 큰 모델들을 사전학습하면 할수록 모델의 모든 파라미터를 재 훈련하는 Full Fine-Tuning 은 실현 가능성이 낮아진다.

우리는 LoRA(Low-Lank Adaptation)을 제안한다. 이것은 사전학습된 모델의 가중치(weights)를 고정(freeze)시키고, 각 Transformer 구조의 layer 에 Rank Decomposition Matrices 를 주입(inject)한다. 이를 통해 downstream task 를 위한 훈련가능한 파라미터 수를 획기적으로 줄일 수 있다.

Adam 으로 미세조정(fine-tuned)된 1 조 7,500 억개의 파라미터를 가진 GPT-3 175B 과 비교했을 때, LoRA 는 훈련가능한 파라미터 수를 1 만배, 요구되는 GPU 메모리를 3 배정도 줄일 수 있다.

LoRA 는 더 적은 훈련 파라미터, 높은 처리량, 그리고 어댑터(adapter)와 다르게 추가 추론 대기시간 없이 RoBERTa, DeBERTa, GPT-2 등 미세조정된 여러 모델들과 비교했을 때 동등하거나 더 좋은 성능을 발휘한다.

1. 소개

NLP 분야에서 많은 어플리케이션들은 하나의 대규모 사전학습된 언어모델을 다양한 하위 분야(downstream) 태스크에 적용(adapt)시키는데 의존한다. 이러한 적용(adapt)은 주로 사전학습된 모델의 모든 파라미터를 갱신하는 미세조정을 통해 이루어진다. 주요 하위분야 태스크에서 미세조정을 하게 되면 새로운 모델이 원본 모델만큼의 파라미터 수를 포함하고 있다. 이는 매 순간 더 큰 모델이 학습될 때마다 단순한 불편함(inconvenience)에서 중요한 배포 도전 과제(deployment challenge)로 변하고 있다.

많은 사람들이 단 소수의 파라미터만을 갱신하거나 새로운 태스크를 위한 외부 모듈을 학습하는 방식을 통해 이를 완화하려고 했다. 이렇게 하면 각 태스크에 대해 사전학습된 모델 외에 소수의 태스크에 특화된(task-specific) 파라미터만 저장하고 불러오면 되므로 배포 시 운영 효율성이 크게 향상된다.

그러나, 기존의 기술들은 모델의 깊이를 확장하거나 모델의 사용가능한 시퀀스 길이를 줄임으로써 추론 지연시간(inference latency)이 도입되곤 한다. 더 중요한 건 이러한 방법은 미세조정의 베이스라인을 맞추지 못해 종종 성능과 효율성 사이의 균형을 깨지게 만든다.

학습된 과매개변수화된(over-parametrized) 모델이 실제로는 낮은 내재적 차원(low intrinsic dimension)에 위치한다. 우리는 모델이 미세조정(model adaptation(= fine-tuning)) 중에 가중치의 변화가 낮은 내재적 순위(intrinsic rank)를 가진다는 가설을 세우며, 이는 우리가 제안한 LoRA 접근 방식으로 이어진다.

1. 과매개변수화된 모델: 딥러닝 모델은 매우 많은 파라미터를 갖고 있다.
2. 낮은 내재적 차원: 모델이 실제로는 모든 파라미터를 다 사용하지 않는다. 핵심적인 소수의 파라미터만으로도 대부분의 작업을 수행할 수 있다.
3. 가중치 변화의 특성: 모델을 미세조정할 때, 모든 가중치가 크게 변하지 않는다. 오히려 매우 제한된 방향으로만 변화한다.
4. 주요가설: 모델 미세조정 과정에서 가중치 변화는 낮은 '내재적 순위'를 가진다. 즉, 소수의 핵심 방향으로만 변화한다.

LoRA 는 미세조정 중에 밀집한 레이어(dense layers)의 변화를 순위 분해 행렬(rank decomposition matrices)로 최적화함으로써 사전훈련된 가중치를 고정(frozen)하면서 신경망의 일부 밀집한 레이어(some dense layers)를 간접적으로 훈련하게 한다.

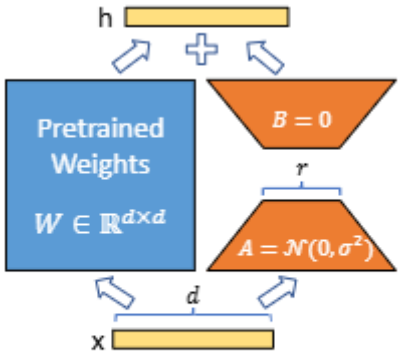

Figure 1: Our reparametrization. We only train A and B.

GPT-3 를 예로 들면, 우리는 전체 랭크(full rank) (i.e., d)가 12,288 만큼 높은 경우에도 매우 낮은 랭크(low rank)(i.e., Figure 1 의 r 이 하나 또는 둘)로 충분하다는 것을 보일 수 있다. 이는 LoRA 를 저장-, 계산-효율적으로 만든다.

LoRA 는 몇 가지 주요 장점을 갖고 있다.

- 하나의 사전학습된 모델을 공유하여 다른 태스크를 위한 많은 소형 LoRA 모듈을 구축하는데 사용될 수 있다. 우리는 공유 모델을 고정(freeze)하고 Figure 1 에서 A 와 B 행렬을 바꿈으로써 효율적으로 태스크를 변경할 수 있다.
- LoRA 는 학습 효율성을 높이고 하드웨어 진입장벽(hardware barrier)을 adaptive optimizer 를 사용할 때보다 3 배 낮춘다. (우리는 변화량을 계산하거나 대부분의 파라미터들을 최적화 상태로 유지할 필요가 없기 때문이다.) 대신, 우리는 오직 더 작은 주입된(injected) 낮은 랭크 행렬(low-rank matrices)만을 최적화한다.
- 우리의 간단한 선형 설계(linear design)를 통해 배포 시에 학습가능한 행렬과 고정된 가중치들을 병합할 수 있으며, fully fine-tuned model 에 비해 추론 지연 시간(inference latency)이 전혀 발생하지 않는다.
- LoRA 는 이전의 많은 방법론들과 직교(orthogonal)하며, 그들 중 다수와 결합되어 사용될 수 있다.

용어 및 규칙

d_{model}: Transformer 레이어의 입력과 출력 차원 크기

W_q, W_k, W_v, W_o: Self-Attention 모듈의 Query, Key, Value 와 Output projection matrices

W, W_0: 사전학습된(pre-trained) 가중치 행렬

ΔW: 미세조정(adaptation) 중 업데이트된 변화량

r: LoRA 모듈의 rank

$d_{ffn} = 4 \times d_{model}$: feed-forward network dimension

2. 연구 문제 진술

우리의 목표는 훈련 목표(training objective)와 무관하지만, 우리는 언어 모델링에 중점을 둔다. 아래는 언어 모델링 문제와 특히 task-specific prompt 가 주어졌을 때 조건부 확률(conditional probabilities)의 최대화에 대한 간결한 설명이다.

우리에게 Φ에 의해 파라미터화 된 사전학습된 자동회귀(autoregressive) 언어 모델 $P_\Phi(y|x)$이 주어졌다고 가정하자. 예를 들어, $P_\Phi(y|x)$은 Transformer 구조를 기반으로 한 GPT 와 같이 일반적인 다중 태스크 학습자(multi-task learner)가 될 수 있다고 하자. 이 사전학습된 모델을 downstream 조건부 텍스트 생성 태스크(e.g. summarization, MRC, NL2SQL)들을 수행하도록 미세조정한다고 생각해보자. 각각의 downstream task 는 context-target pairs: $Z = \{(x_i, y_i)\}_{i=1,\dots,N}$의 훈련 데이터셋에 의해 나타난다. (여기서 x_i와 y_i는 토큰의 시퀀스이다.)

즉, Z는 해당 태스크의 전체 훈련 데이터셋, x_i와 y_i는 각각 입력 문장과 target 를 의미한다.

Full fine-tuning 동안에, 모델은 사전학습된 가중치 Φ_0를 초기화하고 목표를 최대화하기 위해 변화량(gradient)을 반복적으로 따라가며 $\Phi_0 + \Delta\Phi$를 업데이트한다.

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

1. $\max_{\Phi}$: 모델 가중치 Φ를 최대화하는 것이 목표
2. $sum((x,y) \in Z)$: 전체 훈련 데이터셋 Z 의 모든 (x,y) 쌍에 대해 합을 계산
3. $|y|$: 출력 시퀀스 y 의 길이
4. ∑t=1: 시퀀스의 각 토큰 위치에 대해 합을 계산
5. $\log\left(P_{\Phi}(y_t|x, y_{<t})\right)$:
    - y_t: t 번째 출력 토큰
    - x: 입력 컨텍스트
    - $y_{<t}$: t 이전의 모든 출력 토큰
    - P_{Φ}: 모델이 예측하는 조건부 확률

즉, "주어진 컨텍스트와 이전 출력 토큰들을 고려했을 때, 다음 토큰을 얼마나 정확하게 예측하는지"를 최대화하는 것이다.

전체 미세조정의 주요 단점 중 하나는 아래와 같다.

각 downstream task 에 대해 우리는 서로 다른 ($|\Phi_0|$와 같은 차원의) 파라미터 조합 $\Delta\Phi$ 을 학습한다. 따라서, 만약 사전학습된 모델이 GPT-3 와 같이 거대하다면 미세조정된 모델의 많은 독립적인 인스턴스들을 저장하고, 배포하는 것은 불가능에 가깝다.

이 논문에서 우리는 더 parameter-efficient 한 접근방식을 채택한다. 여기서 task-specific 파라미터 증분 $\Delta\Phi = \Delta\Phi(\Theta)$은 훨씬 더 작은 크기($|\Theta| \ll |\Phi_0|$)의 파라미터 조합 Θ에 의해 인코딩된다. $\Delta\Phi$을 찾는 작업은 Θ에 대해 최적화된다.

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

사전학습 모델로 GPT-3 175B 를 사용했을 때, 훈련가능한 파라미터 $|\Theta|$의 수는 $|\Phi_0|$의 0.01%만큼 작았다.

3. 존재하는 해결방법들은 충분하지 않은가?

우리가 해결하려고 하는 문제는 결과 새로운 것이 아니다. 전이 학습(transfer learning)이 시작된 이후로 수십개의 작업들이 모델을 미세조정하는데 있어 파라미터적으로, 컴퓨팅적으로 더욱 효율적으로 만들려는 움직임을 보여줬다. 두 가지 효과적인 전략(adding adapter layers, optimizing some forms of the input layer activations)이 있다. 그러나, 두 전략 모두 그들만의 한계점이 존재한다. (대규모에서 성능 저하, 지연 시간에 민감한 프로덕션 환경에서 비효율적)

Adapter Layers Introduce Latency

Adapter 는 많은 변형이 있다.

Transformer block 당 두 개의 adapter layer 가 존재하는 원본 디자인(Houlsby et al. (2019))과 block 당 하나의 adapter layer 가 있지만 추가적인 LayerNorm 이 존재하는 디자인(Ba et al. (2016))에 초점을 맞춘다.

레이어를 가지치기(pruning)하거나 멀티태스킹 설정을 활용하여 전체 지연 시간을 줄일 수는 있지만, adapter layer 에서 추가 연산을 우회할 직접적인 방법은 없다.

Adapter layer 는 작은 병목 차원(bottleneck dimension)을 가지며 매우 적은 파라미터로 디자인 되어있기에 문제는 아닌 것처럼 보인다.

> 병목 차원(bottleneck dimension)이란?
>
> - 입력 → 중간 차원 → 원래 차원으로 확장 되는 구조에서, 레이어의 중간 차원을 의미한다.
> - 즉, 병목 차원을 작게 유지하여 계산량(FLOPs)를 최소화하여 모델의 성능을 크게 저해하지 않으면서 계산 비용을 최소화하여 문제가 되지 않을 것이다.

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%)

그러나, 큰 신경망은 지연 시간을 낮게 유지하는 병렬처리 하드웨어(GPU)에 의존하지만 adapter layer 는 순차적으로 처리되어야 한다. 이는 배치 크기가 일반적으로 1 만큼 작은 온라인 추론 설정에 차이를 만든다. 모델 병렬성(하나의 GPU)이 없는 일반적인 시나리오에서, 우리는 매우 작은 병목 차원을 가진 adapter 를 사용할 때도 주목할만한 지연시간의 증가를 관찰하였다.

이 문제는 우리가 모델을 공유(shard)해야 할 때 심각해진다. adapter 파라미터를 여러 번 중복하여 저장하지 않는 한, 추가적인 depth 는 AllReduce, Broadcast 와 같은 더 많은 동기 GPU 작업을 요구하기 때문이다.

Directly Optimizing the Prompt is Hard

반대로, prefix tuning 는 다른 문제에 직면해있다. 우리는 prefix tuning 은 최적화 하기에 어려울 뿐만 아니라 그것의 성능이 훈련가능한 파라미터에서 비단조롭게(non-monotonically) 변화하는 것을 관찰했다.

비단조롭다.

- 성능이 불규칙하고 예측 불가능하게 변동한다.
- 때로는 파라미터 조정 후 성능이 오히려 악화된다.
- 학습 과정에서 불안정한 반응을 보인다.

보다 근본적으로, 미세조정을 위한 시퀀스 길이의 일부분을 예약(reserving) (아마도, 길이에 제약을 두는 것 같음) 하는 것은 반드시 downstream task 를 처리하는데 사용할 수 있는 시퀀스 길이를 줄이며, 이로 인해 프롬프트 튜닝이 다른 방법에 비해 성능이 떨어질 것으로 예상된다.

4. 우리의 방법

우리는 단순한 디자인의 LoRA 와 그것의 실험적인 이점들을 설명한다. 여기서 설명된 원칙은 딥러닝 모델의 모든 신경망에 적용되지만, 우리는 실험에서 동기 부여 사례로 Transformer 기반의 언어모델의 특정 가중치에만 초점을 맞춘다.

4.1. Low-Rank-Parametrized Updated Matrices

신경망은 행렬곱(matrix multiplication)을 수행하는 많은 밀집한 레이어들(dense layers)을 포함한다. 이러한 레이어들의 가중치 행렬은 특히 full-rank 를 가진다.

선형 대수에서의 rank

- 행렬의 선형적으로 독립인 열의 최대 개수를 뜻한다.

$$\begin{bmatrix} 1 & 0 & 1 \\ -2 & -3 & 1 \\ 3 & 3 & 0 \end{bmatrix}$$

- 위 행렬에서 rank 는 2 이다. 세 번째 열의 경우 첫 번째 열에서 두 번째 열을 빼면 얻을 수 있기 때문이다.
- Full-rank 라는 것은 한 행이나 열에서 전부 다 선형 독립인 기저 벡터들을 가진 경우이다.

특정 태스크를 미세조정할 때, Aghajanyan et al. (2020)는 사전학습된 언어모델은 낮은 내재적 차원을 가지며, 더 작은 부분공간에 무작위로 투영되더라도 여전히 효율적으로 학습할 수 있음을 보여주었다. 이에 영감을 받아 우리는 미세조정하는 동안 가중치 업데이트는 낮은 내재적 순위(rank)를 가진다는 가설을 세웠다. 사전학습된 가중치 행렬 $W_0 \in \mathbb{R}^{d\times k}$의 경우, ΔW 를 낮은 순위 분해(low-rank decomposition) BA로 표현, 즉 $W0 + \Delta W = W0 + BA$로 표현하여 ΔW의 업데이트를 막는다. ($B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}, r \ll min(d,k)$)

훈련 중에 사전학습된 가중치 행렬 W_0은 고정되며, 기울기 업데이트를 진행하지 않지만 A와 B는 학습 가능한 파라미터이다. W_0와 $\Delta W = BA$ 모두 같은 입력값(x)이 곱해지고, 그들의 각각의 출력 벡터는 coordinate-wise 방식으로 더해진다.

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

우리는 행렬 A를 random Gaussian initialization 방식, 행렬 B를 0 으로 초기화한다. 따라서 $\Delta W = BA$는 훈련 시작 시에 0 에서 시작한다. 다음 우리는 $\Delta W\, x$를 $\frac{\alpha}{r}$로 scale 한다. (α는 임의의 값) Adam 으로 최적화할 때, α를 튜닝하는 것은 대충 learning rate 를 최적화하는 것과 같다고 볼 수 있다. 결과적으로, 우리는 α를 r로 정해두고 최적화하지 않았다.

A Generalization of Full Fine-tuning

더 일반적인 형태의 미세조정을 통해 사전학습된 파라미터의 하위 집합을 학습할 수 있다. LoRA 는 한 발자국 더 나아가 미세조정 과정에서 가중치 행렬에 대한 누적된 그래디언트 업데이트(accumulated gradient update)가 full-rank 를 가질 필요가 없다. 다시 말해서, 모든 가중치 행렬에 LoRA 를 적용하고 모든 편향(bias)을 훈련할 때, 우리는 LoRA 의 rank r을 사전학습된 가중치 행렬의 rank 로 설정함으로써 full fine-tuning 의 표현력을 대략적으로 회복할 수 있다. 즉, 훈련가능한 파라미터의 수를 늘릴수록 training LoRA 는 training original model 과 대략적으로 수렴하는 반면, adapter-based methods 는 MLP 로 수렴하고 prefix-based methods 는 긴 입력 시퀀스를 사용할 수 없는 모델로 수렴한다.

Adapter-based methods 의 MLP 수렴 이유

- Bottleneck 구조로 설계되었다. (input dim → bottleneck dim → original dim)
- 원본 모델의 가중치는 고정되어 있다.
- 파라미터의 개수가 커질수록, 결과적으로는 단순한 MLP 로 수렴한다.

No Additional Inference Latency

배포 시에, $W = W_0 + BA$를 명시적으로 계산하고 저장하여 평소와 같이 추론을 수행할 수 있다.

다른 downstream task 로 전환해야 할 때, BA를 빼서 W_0를 복구하고 그 대신 다른 $B'A'$를 더해주어 처리할 수 있다. 이 작업은 memory overhead 가 거의 없는 빠른 작업이다. 결정적으로 이는 추론 중에 추가 지연 시간이 발생하지 않도록 보장한다.

4.2. Applying LoRA to Transformer

원칙적으로 신경망 가중치 행렬의 어느 부분 집합에나 LoRA 를 적용하여 훈련가능한 파라미터의 수를 줄일 수 있다.

Transformer 구조에는 Self-Attention 모듈 내에 4 개의 가중치 행렬(W_q, W_k, W_v, W_o)과 MLP 모듈에 2 개의 가중치 행렬이 존재한다. 우리는 출력 차원이 attention head 로 슬라이싱 되더라도 W_q, W_k, W_v 행렬을 $\mathbb{R}^{d_{model} \times d_{model}}$ 차원의 단일 행렬로써 취급한다. 단순성(simplicity)과 효율성(parameter-efficiency)를 위해 downstream task 에 대한 attention 가중치만 조정하고 MLP 모듈의 가중치는 고정한다. 7.1 섹션에서 W_q, W_k, W_v 행렬이 가지는 차이점에 대해서도 다뤘다.

5. 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	$87.5_{\pm.3}$	**$95.1_{\pm.2}$**	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	**$93.3_{\pm.3}$**	$90.8_{\pm.1}$	**$86.6_{\pm.7}$**	**$91.5_{\pm.2}$**	**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	**$90.6_{\pm.2}$**	$96.2_{\pm.5}$	**$90.9_{\pm1.2}$**	**$68.2_{\pm1.9}$**	**$94.9_{\pm.3}$**	$91.6_{\pm.1}$	**$87.4_{\pm2.5}$**	**$92.6_{\pm.2}$**	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	**$68.3_{\pm1.0}$**	**$94.8_{\pm.2}$**	**$91.9_{\pm.1}$**	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	**$90.5_{\pm.3}$**	**$96.6_{\pm.2}$**	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	**$94.8_{\pm.3}$**	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	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	**$90.6_{\pm.2}$**	$96.2_{\pm.5}$	**$90.2_{\pm1.0}$**	$68.2_{\pm1.9}$	**$94.8_{\pm.3}$**	$91.6_{\pm.2}$	**$85.2_{\pm1.1}$**	**$92.3_{\pm.5}$**	**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	**$91.9_{\pm.2}$**	$96.9_{\pm.2}$	**$92.6_{\pm.6}$**	**$72.4_{\pm1.1}$**	**$96.0_{\pm.1}$**	**$92.9_{\pm.1}$**	**$94.9_{\pm.4}$**	**$93.0_{\pm.2}$**	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

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^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{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 (Adapter^{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 (Adapter^{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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

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 (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

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm 0.5\%$, MNLI-m around $\pm 0.1\%$, and SAMSum around $\pm 0.2/\pm 0.2/\pm 0.1$ for the three metrics.

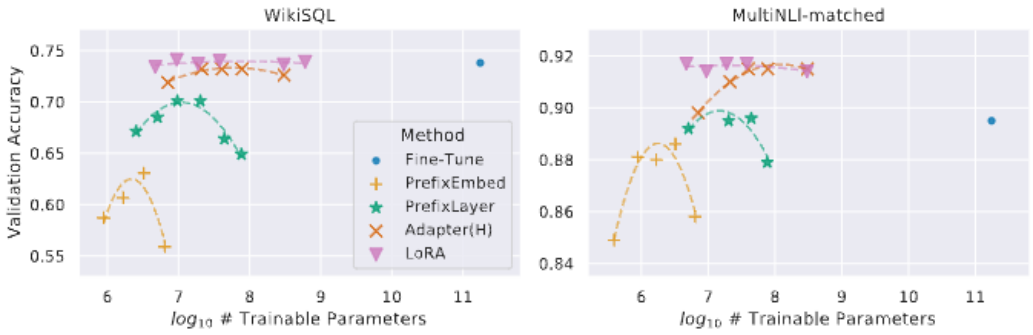

Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance. See Section F.2 for more details on the plotted data points.

7. Low-Rank Update 이해하기

low-rank 구조는 여러 실험을 병렬적으로 이끄는 하드웨어 진입장벽을 낮출 뿐 아니라, 사전학습된 가중치와 업데이트된 가중치 사이에 어떤 상관관계가 있는지 더 잘 이해할 수 있게 도와준다.

1) 파라미터 크기 제약이 있을 때, 사전학습된 Transformer 모델에서 downstream 성능을 극대화하기 위해 미세조정해야 할 가중치 행렬의 하위 집합은 무엇인가?

트랜스포머의 가중치 행렬 중 어떤 행렬을 미세조정해야 성능을 최대화할 수 있을까?

- 모델의 모든 가중치 행렬을 모두 바꾸는 것은 비효율적
- 어떤 특정 행렬들이 성능 향상에 더 중요하게 작용하는지 탐구

	# 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_q, W_k 행렬을 전부 적용할 경우 성능이 낮게 나왔다. 반면, W_v, W_o 행렬을 모두 적용했을 때는 성능이 가장 높게 나왔다. 이는, rank 가 낮더라도 여러 개의 가중치 행렬을 미세조정하는 것이 더 바람직하다는 것을 시사한다.

2) "최적의" 미세조정 행렬 ΔW은 정말 rank-deficient 한가? 그렇다면 실제로 사용하기에 good-rank 는 무엇인가?

- 실제 적용 시 최적의 랭크 찾기

	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 r로도 견줄만한 성능을 발휘하는 것을 보여준다. 이는 ΔW 행렬이 매우 작은 "intrinsic rank"를 가질 수 있음을 시사한다. r을 증가시키는 것이 더 의미 있는 정보를 포함하지 않으며, 낮은 순위의 미세조정 행렬(low-rank adaptation matrix)로도 충분하다는 것을 시사한다.

3) ΔW와 W의 연관성은 무엇인가? ΔW와 W가 높은 상관관계가 있나? ΔW는 W와 비교했을 때 어느정도 크기인가?

- 원본 가중치와 미세조정 가중치 사이의 상관관계 분석
- 원본 행렬과 미세조정 행렬의 크기 비교

	$r=4$			$r=64$		
	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
$\|\|W_q\|\|_F = 61.95$	$\|\|\Delta W_q\|\|_F = 6.91$			$\|\|\Delta W_q\|\|_F = 3.57$		

ΔW는 랜덤 행렬에 비해 W와 더 강한 상관관계를 가지며, 이는 ΔW가 이미 W에 있는 일부 특징을 증폭시킨다는 것을 나타낸다. ΔW는 W의 top singular direction 을 반복하는 대신 W에서 강조되지 않는 방향만 증폭한다. 이는 low-rank adaptation matrix 가 일반 사전학습 모델에서 학습되었지만 강조되지 않은 특정 downstream task 에 대한 중요한 특징을 잠재적으로 증폭시킬 수 있음을 시사한다.