논문 분석

LoRA: Low-Rank ADAPTATION OF LARGE LANGUAGE MODELS

(Edward Hu, Yelong Shen, ...)

1. INTRODUCTION

NLP에서는 하나의 대규모 사전 학습된 언어 모델을 다양한 DownStream Application에 적용하는 경우가 많다. 이러한 적응(adaptation)은 일반적으로 fine-tuning을 통해 이루어지며, 이 과정에서 사전 학습된 모델의 모든 파라미터가 업데이트된다.

그러나, 파인튜닝의 가장 큰 단점은 새로운 모델이 원본 모델과 동일한 수의 파라미터를 포함해야 한다는 점이다. 대규모 모델의 등장으로 이러한 문제는 단순한 불편함에서 심각한 배포(deployment) 문제로 발전하고 있다.

이를 완화하기 위해 일부 파라미터만 적응(adapting)하거나, 새로운 작업을 위한 외부 모듈을 학습하는 방법이 연구되었다. 이러한 접근법은 사전 학습된 모델을 유지하면서도 작업별(task-specific)로 필요한 적은 수의 파라미터만 저장 및 로드하면 되므로, 배포 시 운영 효율성을 크게 향상시킬 수 있다.

그러나 기존 기법들은 몇 가지 한계를 가진다.

가. 추론 속도 저하(Inference Latency) : 일부 기법들은 모델의 깊이(depth)를 확장함으로써 추론 속도를 저하시킨다.
나. 시퀀스 길이 제한(Sequence Length Reduction) : 또 다른 방법들은 모델이 처리할 수 있는 시퀀스 길이를 줄이는 단점이 있다.
다. 파인튜닝 성능 미달(Performance Gap) : 무엇보다도, 기존 기법들은 fine-tuning 성능을 완전히 따라잡지 못하는 경우가 많아, 효율성과 모델 품질 간 trade-off가 발생한다.

이에, 저자는 **과적합(over-parameterized)된 모델이 실제로 저차원 내재 공간(low intrinsic dimension)에 위치한다는 사실**을 기반으로, 모델 적응(adaptation) 중 가중치 변화 또는 낮은 내재 랭크(intrinsic rank)를 가진다고 가정하고, 이를 기반으로 저랭크 적응(Low-Rank Adaptation, LoRA) 방법을 제안했다.

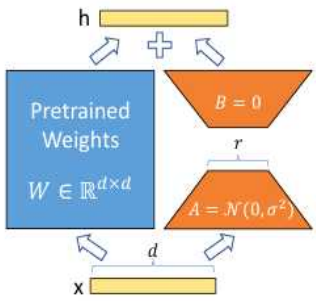

Figure 1: Our reparametrization. We only train A and B.

LoRA는 뉴럴 네트워크의 일부 밀집층(Dense Layers)을 직접 학습하는 대신, **적응 중 발생하는 가중치 변화(weight updates)의 저랭크 행렬 분해(rank decomposition)만은 최적**

화하도록 한다. 이를 통해 사전 학습된 가중치는 그대로 유지(frozen)한 채 학습이 이루어진다.

LoRA는 다음과 같은 주요 장점들을 갖는다.

가. 사전 학습된 모델을 공유하고 여러 LoRA 모듈을 구축 가능 : 사전 학습된 모델을 그대로 유지(freeze)하면서, 서로 다른 작업별로 작은 LoRA 모듈만을 추가할 수 있다. -> Figure 1에서 행렬 A와 B만 교체하면 새로운 작업으로 전환할 수 있어, 저장 공간(storage) 요구량과 작업 전환(task-switching) 비용이 크게 감소한다.

나. 훈련 효율 향상 및 하드웨어 요구사항 감소 : LoRA를 사용하면 대부분의 가중치에 대한 기울기 계산(gradient computation)과 옵티마이저 상태 유지(optimizer state maintenance)가 불필요하다. -> 저랭크 행렬(low-rank matrices)만 최적화하면 되므로, 적응형 옵티마이저(adaptive optimizers) 사용 시 최대 3배까지 하드웨어 요구량 감소 효과가 있다.

다. 추론 시 지연(latency) 없음 : LoRA의 단순한 선형 설계 덕분에, 학습된 LoRA 행렬을 사전 학습된 가중치와 병합(merge)하여 사용할 수 있다. -> 즉, 완전한 fine-tuning 모델과 비교해도 추가적인 추론 지연이 발생하지 않는다.

라. 기존 기법과 독립적으로 사용 가능하며, 조합이 용이함 : LoRA는 기존의 다양한 기법과 독립적(orthogoanl)이며, 결합해서 사용할 수도 있다.

※ Terminologies and Conventions (용어 및 표기 규칙)
LoRA 논문에서는 Transformer 아키텍처의 표준적인 용어 및 표기법을 사용한다.

1. **Transformer 차원 관련 용어**
   - d_{model} : Transformer 레이어의 입력 및 출력 차원 크기
   - $d_{ffn} = 4 \cdot d_{model}$: Transformer의 MLP(feedforward) 차원
2. **Self-Attention 모듈의 프로젝션 행렬**
   - W_q, W_k, W_v, W_o : 각각 **Query, Key, Value, Output 프로젝션 행렬**을 의미함
3. **가중치 및 LoRA 관련 표기**
   - W 또는 W^0 : 사전 학습된 가중치(pre-trained weight matrix)
   - ΔW : 적응(adaptation) 과정에서 업데이트된 누적 기울기(gradient update)
   - r : **LoRA 모듈의 랭크(rank)**
4. **최적화 방법**
   - **Adam 옵티마이저(Adam optimizer)** 사용 (참조: Loshchilov & Hutter, 2019; Kingma & Ba, 2017)

이 논문은 Vaswani et al. (2017) 및 Brown et al. (2020)에서 정의한 **Transformer 구조 및 표기 규칙을 따름.**

2. PROBLEM STATEMENT

LoRA는 특정 학습 목표(training objective)에 구애받지 않는 일반적인 방법론이지만, 이 논문에서는 언어 모델링(language modeling)을 주요 사례로 다룬다.

언어 모델은 주어진 입력 시퀀스 x = (x1, x2, ..., xT)에 대해 다음 단어를 예측하는 확률을 최대화하는 것이 목표이다. 즉, 조건부 확률 P(xt|x<t)를 최대화하는 것이 핵심이다.

LoRA는 task-specific 프롬프트가 주어졌을 때, 모델이 생성하는 조건부 확률을 최적화하는 방법을 제공하는데, 이는 모델이 프롬프트 기반 조건부 확률 P(y|x)를 학습하는 방식과 동일한 구조를 갖는다.

주어진 사전 학습된 autoregressive 언어 모델 P(y|x)을 다운스트림(conditional text generation) 태스크에 적응시키는 것이 목표이다. 여기서 P(y|x)는 GPT와 같은 멀티태스크 학습 모델(multi-task learner)이며, Transformer 기반으로 설계됐다.

LoRA는 다음과 같은 다양한 조건부 텍스트 생성 태스크(Conditional Text Generation Tasks)에 적용될 수 있다.

가. 요약(Summarization):
  1) 입력 xi : 기사 본문
  2) 출력 yi : 기사 요약문

나. 기계 독해 이해(Machine Reading Comprehension, MRC)
  1) 입력 xi : 문서와 질문(Document & Question)
  2) 출력 yi : 정답

다. 자연어 -> SQL 변환(NL2SQL)
  1) 입력 xi : 자연어 질의(Natural Language Query)
  2) 출력 yi : SQL 문장

각 다운스트림 태스크는 문맥-타겟(context-target) 쌍의 데이터 셋 Z로 표현되는데,

$$Z = \{(x_i, y_i)\}_{i=1}^{N}$$

각 xi는 입력 문맥(Context)이며, yi는 출력 텍스트(Target)을 나타낸다.

A. Full Fine-Tuning 과정

사전 학습된 모델의 가중치를 Φ0이라고 할 때, 모델을 다운스트림 태스크에 적응시키기 위해 전체 가중치(full parameters)를 Φ = Φ0 + ΔΦ 꼴로 기울기(gradient)를 따라 반복적으로 업데이트 된다. 이때, ΔΦ는 모델이 학습을 통해 변경한 가중치의 변화량이다.

언어 모델의 목표는 조건부 확률 P(y|x)를 최대화하는 것, 즉 주어진 입력 x에 대해 올바른

출력을 생성하도록 모델을 학습하는 것으로, 다음과 같은 목적 함수를 최대화하는 것과 동치이다

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

B. Full Fine-Tuning의 주요 단점

(1) 모든 다운스트림 태스크마다 별도의 모델 파라미터를 학습
  가. Fine-tuning을 할 때, 각 태스크마다 새로운 파라미터 세트를 학습해야 함.
  나. GPT-3(175B 파라미터)를 다양한 태스크에 맞게 각각 학습하면 각 모델마다 175B개의 파라미터를 저장해야 함

(2) 메모리 및 저장 공간 문제
  가. 거대한 모델을 여러 태스크에 대해 Fine-tuning하면 막대한 저장 공간이 필요함
  나. 여러 개의 Fine-tuned 모델을 배포하고 관리하는 것이 비효율적이고 거의 불가능한 수준이 될 수 있음

(3) 운영 부담
  가. 모델이 커질수록 학습과 배포가 어려워지고, 하드웨어 비용이 증가
  나. 여러 태스크를 지원하려면 매번 새로운 모델을 로드해야 하므로 태스크 간 전환 속도 저하

이 논문에서는 보다 효율적인 파라미터 활용 기법을 채택하여, 각 태스크별 파라미터 증가분 ΔΦ = Φ - Φ0를 더 작은 크기의 파라미터 집합으로 인코딩한다. 즉, Fine-tuning처럼 전체 모델을 업데이트하는 것이 아니라, **ΔΦ 자체를 작은 차원의 파라미터 행렬로 변환하여 학습하는 방식이다**. 결국 ΔΦ 값을 찾는 일은 다음 목적함수를 최적화하는 일과 동치가 된다.

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

이 논문에서는 저차원(low-rank) 표현을 활용해 ΔΦ를 계산 및 메모리 효율적인 방식으로 인코딩하는 방법을 제안하고 있다. 특히, GPT-3 175B 모델을 기준으로 학습을 해야 할 파라미터 수가 Φ0의 0.01% 수준까지 줄어들 수 있다.

3. AREN'T EXISTING SOLUTIONS GOOD ENOUGH?

전이 학습(Transfer Learning) 초기부터 수많은 연구들이 모델 적응(adaption)의 계산 및 파라미터 효율성을 개선하려고 했는데, 기존 접근 방식에는 다음과 같은 것들이 있다.

(1) Adapter Layers 추가

1) 대표 연구 : Houlsby et al., 2019; Rebuffi et al., 2017; Pfeiffer et al., 2021
2) 기존 Transformer 레이어 사이에 추가적인 작은 네트워크(어댑터)를 삽입
3) 장 : 파라미터 효율성 증가
4) 단 : 추론(inference) 속도가 느려지기 때문에 Latency가 중요한 환경에서는 부적합

(2) 입력 레이어 활성화 값 최적화

1) 대표 연구 : Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021
2) 모델의 입력 토큰 임베딩을 수정하는 방식
3) 장 : 추가적인 네트워크를 넣지 않아 비교적 단순
4) 단 : 모델의 학습 가능한 시퀀스 길이가 감소해 긴 문장을 처리하기 어려움

A. Adapter Layers의 문제점

Adapter Layers는 기본적으로 Transformer 블록마다 두 개의 어댑터 레이어를 추가하거나 하나의 어댑터 + 추가 LayerNorm를 사용하는데, Adatper Layers는 순차적으로 처리되어야 하기 때문에 하드웨어의 병렬 처리를 통해 속도 이득을 보는 대형 신경망에서는 병렬성 손실이 나타나게 된다. 특히, 온라인 추론(Inference)에서는 Batch Size가 작아지는 환경이 많아 성능 저하가 심각하다.

또, Adapter는 작은 병목 차원(Bottleneck Dimension)을 사용해 파라미터를 줄일 수 있는데, FLOP 연산량 증가를 피할 수 없기 때문에 지연 시간 증가는 필연적인 부정적 요소이다.

이 문제는 대형 모델을 여러 GPU에 나누어(Shard) 배포하는 기법인 대규모 모델 분할(Sharding)에서 두드러지게 나타나는데, 예로 GPT-3 같은 대형 모델은 단일 GPU에 올릴 수 없기 때문에 이 Sharding 기법은 필수적인 기법이지만,

(1) Adapter Layers를 사용하면 기존 모델보다 추가적인 깊이(Depth)를 가지므로 추가적인 GPU 간 동기화(Synchronization) 연산이 필요하다.

(2) 각 GPU가 Adapter Layer 파라미터를 중복 저장해야 하는데, 그렇지 않으면 모든 GPU에서 매번 동기화를 해야 하므로 연산 비용이 증가하게 된다.

B. 프롬프트 직접 최적화(Directly Optimizing the Prompt)의 어려움

기존 모델의 가중치는 그대로 두고 입력(prompt) 앞에 학습 가능한 토큰을 추가하는 방식인 Prefix Tuning은 LoRA처럼 추가적인 파라미터를 학습하지만, 네트워크 구조를 변경하지 않는다.

하지만 이 Prefix Tuning의 주요 문제점에는 다음과 같은 것들이 있는데,

(1) 최적화(Optimization) 난이도
    1) Prefix Tuning의 성능이 학습 가능한 파라미터 수에 따라 비선형적으로 변함
    2) 파라미터를 늘린다고 해서 성능이 항상 개선되지는 않음

(2) 사용 가능한 시퀀스 길이 감소
    1) 모델이 프롬프트를 학습하기 위해 입력 토큰 일부를 할당해야 함
    2) 프롬프트 길이만큼 실제 입력에 사용 가능한 시퀀스 길이가 줄어들기에, 긴 문맥을 필요로 하는 태스크에서 문제 발생 가능

4. OUR METHOD

이 논문에서는 LoRA의 간단한 설계와 그 실제적인 이점을 설명하는데, 여기서 설명하는 원칙은 심층 학습 모델의 모든 밀집(Dense) 레이어에 적용될 수 있지만, Transformer 언어 모델의 특정 가중치에만 초점을 맞추고 있다.

A. Low-Rank-Parameterized Update Matrices

신경망에는 일반적으로 행렬 곱셈을 수행하는 수많은 밀집(Dense) 레이어가 포함돼 있고, 이러한 레이어의 가중치 행렬은 일반적으로 풀 랭크(full-rank)를 가진다. 특정 작업에 적응할 때 사전 훈련된 언어 모델이 낮은 내재적 차원(intrinsic dimension)을 가지고 있기에 더 작은 부분공간으로의 무작위 투영에도 불구하고 효율적인 학습을 진행할 수 있는데, 이 논문에서는 이에 영감을 받아 가중치의 업데이트 적응 중에 낮은 내재적 랭크(intrinsic rank)를 가진다고 가정한다.

즉, 사전 훈련된 가중치 행렬 $W0 \in R^{d \times k}$에 대해, 이를 낮은 랭크 분해로 나타내어 업데이트를 다음과 같이 제약하게 된다. : $W0 + \triangle W = W0 + BA$, $B \in R^{d \times r}$, $A \in R^{r \times k}$, 그리고 랭크 r은 min(d, k)

훈련 중에 W0는 고정돼 있고, 기울기 업데이트를 받지 않으며, A와 B는 학습 가능한 파라미터를 포함하는데, W0와 $\triangle W=BA$는 동일한 입력과 곱해지고, 각각의 출력 벡터는 coordinate-wise로 더해진다. 즉, $h=W0x$ 에 대해, 수정된 순전파(forward pass)는 다음과 같이 계산된다.

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

A는 무작위 가우시안 초기화를 사용하고, B는 0으로 초기화하기 때문에 훈련 시작 시 △W = BA는 0이다. 이후, △Wx를 a/r로 스케일링하는데, a는 여기서 r 안의 상수이다. 이는 Adam으로 최적화할 때 초기화를 적절히 스케일링하면 튜닝은 학습률을 튜닝하는 것과 거의 동일하다. 결과적으로, 이 논문에서는 a을 처음에 설정한 r에 맞게 설정한 후 이를 튜닝하지 않기 때문에, r을 변경할 때마다 하이퍼파라미터를 재조정해야 할 필요를 줄이게 된다.

(1) A Generalization of Full Fine-tuning

더 일반적인 형태의 파인튜닝은 사전 훈련된 매개변수의 일부를 훈련하는 것을 허용하는데, LoRA는 이보다 더 발전해 적응 과정에서 가중치 행렬의 누적된 그래디언트 업데이트가 전체 순위(full-rank)를 요구하지 않는다. 이는 LoRA를 모든 가중치 행렬에 적용하고 모든 편향을 훈련할 때, LoRA rank r을 사전 훈련된 가중치 행렬의 순위로 설정함으로써 전체 파인튜닝의 표현력을 대략적으로 복구할 수 있음을 의미한다.

다시 말해서, 훈련 가능한 매개변수 수를 증가시키면 LoRA 훈련은 대략적으로 원본 모델 훈련과 수렴하는 반면, 어댑터 기반 방법은 MLP로 수렴하고, prefix-based 방법은 긴 입력 시컨스를 처리할 수 없는 모델로 수렴하게 된다.

(2) No Additional Inference Latency

프로덕션에 배포될 때, LoRA는 명시적으로 W = W0 + BA를 계산하고 저장한 후, 일반적인 방식으로 추론을 수행할 수 있다. W0과 BA가 모두 R ∈ R^(d × k) 안에 있기에, **다른 다운스트림 작업으로 전환해야 할 경우라도 BA를 빼고 다른 BA를 추가하여 W0를 복구할 수 있다**. 이렇게 하면, 구조상으로 파인튜닝된 모델에 비해 추론 중에 추가적인 지연을 발생시키지 않는다는 것을 보장하게 된다.

B. Applying LoRA To Transformer

원칙적으로, 신경망에서 학습 가능한 매개변수의 수를 줄이기 위해 LoRA를 weight matrix의 모든 부분집합에 적용할 수 있다. Transformer 아키텍처에서는 self-attention 모듈에 네 개의 weight matrix(Wq, Wk, Wv, Wo)와 MLP 모듈에 두 개의 weight matrix가 있는데, 이 논문에서는 단순성과 매개변수의 효율성을 위해 downstream 작업에서 attention weight만 조정하고 MLP 모듈은 고정(freeze)하여 downstream 작업에서 학습되지 않도록 설정하고, Wq(혹은 Wk, Wv)를 d(model) x d(model) 차원을 갖는 단일 matrix로 취급한다.

(1) Practical Benefits and Limitations

LoRA의 가장 큰 이점은 메모리 및 저장 공간 사용의 감소이다. Adam으로 학습된 대형 Transformer의 경우, frozen 매개변수에 대한 옵티마이저의 상태를 저장할 필요가 없기 때문에 VRAM 사용량을 최대 2~3배 줄일 수 있다. GPT-3 175B에서는 학습 중 BRAM 소비를 1.2TB에서 350GB로 줄였다. 예로 r=4이고 query와 value projection 매트릭스만 조정된 경우, 체크포인트의 크기는 약 10,000(350GB to 35MB)로 감소한다. 이는 훨씬 적은 수의 GPU로 학습할 수 있게 해주며, I/O 병목 현상을 피할 수 있게 한다.

또 다른 이점은 LoRA weights만 교체하여 작업 간 전환을 훨씬 더 낮은 비용으로 할 수 있다는 점이다. 이는 사전 학습된 가중치를 VRAM에 저장하는 머신에서 많은 맞춤형 모델을 생성하고 즉시 전환할 수 있게 해준다. 또한 GPT-3 175B에서 전체 fine-tuning에 비해 학습 속도가 25% 빨라지는 것을 관찰했으며, 이는 대부분의 매개변수에 대해 기울기를 계산할 필요가 없기 때문이라고 여겨진다.

LoRA에는 몇 가지 한계도 존재하는데, 예로 A와 B가 여러 다른 작업에 대한 입력을 하나의 순전파에서 배치 처리하는 것이 간단하지 않다는 점이다. A와 B를 W에 흡수하여 추가적인 추론 지연을 없애는 방법을 선택하는 경우에도 마찬가지인데, 지연 시간이 중요하지 않은 경우에는 가중치를 병합하지 않고 배치의 샘플에 대해 사용할 LoRA 모듈을 동적으로 선택하는 것을 통해 이를 해결할 수도 있다.

5. EMPIRICAL EXPERIMENTS

논문에서는 LoRA의 다운스트림 작업 성능을 RoBERTa, DeBERTa, 그리고 GPT-2에서 평가한 후, GPT-3 175B로 확장하여 실험을 진행했다. 실험은 자연어 이해(NLU)부터 생성(NLG)까지 다양한 작업을 포함하는데, 구체적으로 RoBERTa와 DeBERTa에 대해서는 GLUE 벤치마크에서 평가를 수행했다. 또, GPT-2에 대해서는 Li & Liang의 설정을 따라 직접 비교를 수행하며 GPT-3에 대한 대규모 실험을 위해 WikiSQL과 SAMSum을 추가적으로 평가했다.

A. BaseLines

다른 baseline과 폭넓게 비교하기 위해, 이전 연구에서 사용된 설정을 그대로 재현하고, 가능한 경우 기존 연구에서 보고된 수치를 재사용했다. 다음은 사용한 baseline 종류이다.

(1) Fine-Tuning(FT) : 파인튜닝은 적응(adaptation)을 위한 일반적인 방법이다. 파인튜닝 동안 모델은 사전 훈련된 가중치와 편향 값으로 초기화되며, 모든 모델 파라미터가 gradient 업데이트를 받는다. 한 가지 간단한 변형으로는 일부 레이어만 업데이트하고 나머지는 고정하는 방법이 있는데, 이 논문에서는 하나인 마지막 두 개의 레이어만 적응시키는 방법(FTTop2)이 적용된 GPT-2 기반 기준선 하나를 추가했다.

(2) Bias-only or BitFit : 편향(bias) 벡터를 제외한 모든 것을 freeze한 baseline

(3) Prefix-embedding tuning(PreEmbed) : 입력 토큰 사이에 특수 토큰을 삽입하는 방법으로, 이러한 특수 토큰은 학습 가능한 단어 임베딩을 가지며, 일반적으로 모델의 어휘에 포함되지 않는다. 이러한 토큰을 어디에 배치할 것인지가 성능에 영향을 미칠 수 있는데, 프롬프트 앞에 추가하는 prefixing과, 프롬프트 뒤에 추가하는 infixing 방법에 초점을 맞추고 각각의 토큰 개수를 lp, li로 표현할 때 학습 가능한 파라미터 수는 d(model) x (lp + li)라는 사실을 알 수 있다.

(4) Prefix-layer tuning(PreLayer) : PreEmbed의 확장 방식으로, 일부 특수 토큰에 대한 단어 임베딩만 학습하는 대신, 각 Transformer 레이어 이후의 활성화 값을 학습한다. 즉, 이전 레이어에서 계산된 활성화 값이 학습 가능한 값으로 대체되기 때문에, 학습 가능한 파라미터의 수는 L x d(model) x (lp + li) 임을 알 수 있다.

(5) Adapter tuning(as proposed in Houlsby et al.) : 셀프 어텐션 모듈(및 MLP 모듈)과 그 이후의 잔차 연결(residual connection) 사이에 어댑터 레이어를 삽입하는 방식으로, 어댑터 레이어는 두 개의 완전 연결(fully connected) 레이어와 bias를 포함하며, 그 사이에 비선형성을 갖는다. 이러한 디자인을 AdapterH 라고 부른다.

추가로, Lin et al. 는 보다 효율적인 어댑터 구조를 제안했는데, 어댑터 레이어를 MLP 모듈 이후와 LayerNorm 이후에만 적용하는 방식이다. 이를 AdapterL 이라고 부르고, Pfeiffer et al.이 제안한 또 다른, 유사한 디자인을 AdapterP라고 부른다.

또, Ruckle et al.이 제안한 AdapterDrop(AdapterD) 기법도 포함하는데, 이는 일부 어댑터 레이어를 제거하여 효율성을 높이는 방법이다.

모든 경우에서, 학습 가능한 파라미터의 수는 L(Adpt) x (2 x d(model) x r + r + d(model)) + 2L(LN) x d(model) 로 계산된다.
※ L(Adpt) : 어댑터 레이어 개수 / L(LN) : 학습 가능한 LayerNorm 개수

(6) LoRA : **LoRA는 기존 가중치 행렬과 병렬로 학습 가능한 저차원(rank decomposition) 행렬 쌍을 추가**한다. 위에서 언급했듯이, 위 논문에서는 실험의 단순성을 위해 Wq와 Wv에만 LoRA를 적용했기에, r에 의해 결정되는 학습 가능한 파라미터의 수는 2 x L(LoRA) x d(model) x r 로 계산된다.
※ L(LoRA) : LoRA를 적용하는 가중치 행렬의 개수

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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

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.

B. RoBERTa Base/Large

RoBERTa는 BERT에서 제안된 사전 학습 방식을 최적화하여, 많은 추가 학습 가능한 파라미터를 도입하지 않고 성능을 향상시켰다. RoBERTa는 최근 몇 년동안 GLUE 벤치마크와 같은 NLP 리더보드에서 더 큰 모델들에 의해 추월당했지만, 여전히 크기에 비해 경쟁력이 있고 실무에서 널리 사용되는 사전 학습된 모델이다.

이 논문에서는 Hugging Face Transformers 라이브러리에서 제공하는 RoBERTa-base(125M) 및 RoBERTa-large(355M)를 사용해 GLUE 벤치마크의 다양한 효율적인 적응 방법을 평가했다.

또한 Houlsby et al. 및 Pfeiffer et al.의 설정을 그대로 재현했는데, LoRA와 어댑터를 공정

하게 비교하기 위해 두 가지 중요한 변경 사항을 적용했다.

가. 모든 작업에서 동일한 배치 크기를 사용하고, 어댑터 베이스라인과 일치하도록 시퀀스 길이를 128로 설정한다.
나. MRPC, RTE 및 STS-B에 대해 모델을 사전 학습된 상태에서 초기화하며, 기존의 파인 튜닝 베이스라인처럼 MNLI에 먼저 적응된 모델을 사용하지 않는다.

C. DeBERTa XXL

DeBERTa는 BERT의 보다 최신 변형 모델로, 훨씬 더 대규모 데이터로 학습되었으며 GLUE 및 SuperGLUE와 같은 벤치마크에서 매우 경쟁력 있는 성능을 보였다. 이 논문에서는 LoRA가 완전히 파인튜닝된 DeBERTa-XXL(1.5B)과 성능 면에서 대등한지 평가한다.

D. GPT-2 MEDIUM/LARGE

LoRA가 자연어 이해(NLU)에서 완전한 파인튜닝과 경쟁할 수 있는 대안임을 보여준 후, LoRA가 자연어 생성(NLG) 모델에서도 우수한 성능을 보일 수 있는지 확인해보기 위해 GPT-2 Medium 및 Large 모델을 대상으로 실험을 진행했다.

E. Scaling up to GPT-3 175B

LoRA의 최종 stress test로, 1750억 개의 파라미터를 가진 GPT-3에 적용해 성능을 평가했다. 높은 학습 비용으로 인해, 각 실험마다 개별적인 표준 편차를 제공하는 대신, 주어진 작업에서의 전형적인 표준 편차만 보고했다.

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

LoRA는 모든 세 개의 데이터셋에서 파인 튜닝을 수행한 기존 모델과 동등하거나 더 나은 성능을 보였다.

※ Prefix-embedding tuning에서 256개 이상의 특수 토큰을 사용할 경우 성능 크게 감소
※ Prefix-layer tuning에서 32개 이상의 특수 토큰을 사용할 경우 성능 크게 감소

6. RELATED WORKS

A. Transformer Language Model

Transformer는 셀프 어텐션(self-attention) 매커니즘을 적극적으로 활용하는 시퀀스-투-시퀀스 아키텍처로, 일반적으로 Transformer 기반 언어 모델이 NLP 분야를 지배하며 다양한 작업에서 최신 성능(SOA)을 달성하고 있다. 그러다가 BERT가 등장하면서 새로운 학습 패러다임이 부상했는데, 이는 다량의 텍스트 데이터를 사전 학습(pre-training)한 후, 특정 작업(task-specific) 데이터로 미세 조정(fine-tuning)하면 성능이 크게 향상됨을 보였다. 이는 단순히 특정 작업 데이터만으로 모델을 훈련하는 것보다 훨씬 뛰어난 성능을 제공한다.

B. Prompt Engineering and Fine-Tuning

GPT-3 175B는 소수의 추가 학습 예제만으로도 동작을 조정할 수 있지만, 그 결과는 입력 프롬프트에 크게 의존한다. 이에, 모델이 원하는 작업에서 최상의 성능을 발휘하도록 프롬프트를 구성하고 형식을 조정하는 경험적 기법이 필요한데, 이를 프롬프트 엔지니어링 내지는 프롬프트 해킹이라고 한다.

파인튜닝은 일반 도메인에서 사전 학습된 모델을 특정 작업에 맞게 다시 훈련하는 과정을 의미하는데, 그 변형 중 하나로 일부 파라미터만 학습하는 방법이 있지만 최적의 다운스트림 성능을 위해 모든 파라미터를 다시 학습하는 것이 일반적으로 선호된다.

그러나 GPT-3 175B처럼 초대형 모델의 경우 일반적인 방식으로 파인튜닝하기 어려운데, 그 이유는 checkpoint의 크기가 매우 크며, 사전 학습과 동일한 수준의 메모리 용량이 필요해 높은 하드웨어 장벽이 존재하기 때문이다.

C. Parameter-Efficient Adaptation

신경망의 기존 레이어 사이에 어댑터 레이어(adapter layer)를 삽입하는 방식이 여러 연구에서 제안됐는데, 이 논문은 이와 유사한 병목 구조(bottleneck structure)를 활용해 가중치 업데이트에 저순위(low-rank) 제한을 부여한다. 그러나 다음과 같은 핵심적인 차이점이 존재하는데,

가. 학습된 가중치를 추론(inference) 시 메인 가중치와 병합할 수 있으며, 이 과정에서 추가적인 지연(latency)을 초래하지 않는다.
나. 기존 어댑터 레이어 방식에서는 추론 시 추가적인 연산 비용이 발생한다.

최근 연구 중 하나인 COMPACTER는 어댑터 레이어를 Kronecker product와 사전 정의된 가중치 공유(weight sharing) 기법을 활용해 매개변수화한다. LoRA는 이와 유사하게 다른 텐서 곱 기반 방법과 결합하면 파라미터 효율성이 더욱 향상될 가능성이 있다.

최근에는 파인튜닝을 대체할 방법으로 입력 단어 임베딩을 최적화하는 접근 방식도 많이 제안

됐는데, 대표적인 예로는 프롬프트 엔지니어링(prompt engineering)의 연속적이고 미분 가능한 일반화가 있다. 그러나 이런 접근 방식은 프롬프트 내에서 더 많은 특수 토큰(special token)을 사용하여 확장하는 방법에 의존하기에, 위치 임베딩(positional embedding)을 학습할 때 실제 작업을 위한 토큰이 차지할 수 있는 시퀀스 길이를 감소시키는 문제를 초래할 수 있다.

D. Low-Rank Structures in Deep Learning

low-rank structure는 기계 학습에서 매우 일반적으로 나타나는데, 많은 기계 학습 문제는 내재적으로 저순위 구조를 가지고 있다. 또한, 딥러닝 과업 중 특히 over-parametrized 신경망에서는 훈련 후 신경망이 저순위 성질을 가지는 것으로 알려져 있다.

이전 연구에서는 원래 신경망을 훈련할 때 명시적으로 저순위 제한(low-rank constraint)을 적용한 사례가 있지만, 고정된(frozen) 모델을 대상으로 저순위 업데이트를 적용하여 다운스트림 과업을 수행한 연구는 거의 없다.

이론적으로 보면, 신경망은 다음과 같은 조건에서 다른 고전적인 학습 방법보다 뛰어난 성능을 보인다.

가. 신경망의 기본 개념(class)이 특정 저순위 구조를 가질 때 -> 이는 신경망이 유한 폭(finite-width)의 neural tangent kernel을 포함한 전통적인 방법보다 더 우수한 성능을 낼 수 있음을 시사한다.
나. 저순위 적응(low-rank adaptation)이 적대적 학습(adversarial training)에도 유용할 수 있음

이와 같은 이론적 배경을 바탕으로, 이 논문이 제안하는 저순위 적응(low-rank adaptation) 업데이트는 기존 연구들과 연계하여 충분한 동기 부여가 가능함을 시사한다.

7. UNDERSTANDING THE LOW-RANK UPDATES

LoRA가 경험적으로 뛰어난 성능을 보이는 만큼, 이 논문은 다운스트림 과업에서 학습된 low-rank adaptation 특성을 더 깊이 분석하고자 한다.

특히 low-rank structure는 다음과 같은 두 가지 주요한 이점을 제공하는데,

라. 하드웨어 요구사항을 낮춤 : LoRA는 훈련해야 할 매개변수의 수를 크게 줄일 수 있어 하드웨어 장벽을 낮추고, 여러 개의 실험을 병렬로 실행할 수 있는 환경을 조성한다.
마. 업데이트 가중치의 해석 가능성을 향상 : LoRA는 저순위 행렬을 활용해 원래의 사전 학습된 가중치(pre-trained weights)와의 관계를 보다 명확하게 분석할 수 있도록 한다.

이 논문은 이러한 연구를 GPT-3(175B) 모델을 중심으로 진행했는데, LoRA를 적용함으로써 훈련해야 할 매개변수의 수를 최대 10,000배 줄이면서도, 다운스트림 과업의 성능을 저하시키지 않는 결과를 얻었다.

LoRA의 효과를 보다 깊이 이해하기 위해 논문에서는 일련의 실증 연구(empirical studies)를 수행했는데, 이를 통해 다음과 같은 핵심 질문에 답을 하고자 했다.

(1) 파라미터 예산이 제한된 상태에서, 사전 학습된 트랜스포머의 어떤 가중치 행렬을 조정해야 다운스트림 성능을 극대화할 수 있는가?
: LoRA는 기존 가중치 행렬을 직접 업데이트하는 것이 아니라, **특정 가중치 행렬에 대해 저순위(low-rank) 적응 행렬을 학습하는 방식**이다. 하지만 **모든 가중치 행렬을 동일하게 조정할 필요는 없으며, 어떤 행렬을 선택하느냐에 따라 성능이 달라질 수 있습니다.**

(2) 최적의 적응 행렬 W는 실제로 저순위인가?
LoRA는 **저순위 행렬을 이용하여 가중치를 조정하는 기법**이지만, 실제로 "최적의" 적응 행렬 W이 **저순위 성질(rank-deficient)을 가지는지**는 확실하지 않다. 만약 W가 실제로 저순위 행렬이라면, **실제 활용할 때 최적의 랭크(rank)는 얼마로 설정하는 것이 적절한가?**
이 논문에서는 실험을 통해 **실제 과업에서 필요한 최소한의 랭크를 찾고, LoRA의 성능과 효율성을 극대화할 수 있는 랭크 설정을 분석한다.**

(3) 적응 행렬 W와 원래 가중치 행렬 W의 관계는 무엇인가?
LoRA에서 학습된 가중치 행렬 W이 **원래의 사전 학습된 가중치 행렬 W와 얼마나 높은 상관관계를 가지는가?** 그리고 **W의 크기(scale)는 W와 비교했을 때 어느 정도인가?**

A. Which Weight Matrices In Transformer Should We Apply LoRA To?

다운스트림 과업에서 최고의 성능을 얻기 위해, LoRA를 적용할 최적의 가중치 유형은 무엇인가? 논문에서는 GPT-3 175B 모델에서 self-attention 모듈의 가중치 행렬만을 고려했다. 파라미터 예산(buget)은 18M(약 35MB)으로 설정했는데, r=8인 경우 하나의 어텐션 가중치 유형만을 조정 가능하고, r=4인 경우 두 개의 어텐션 가중치 유형을 조정 가능하다. 실험은 GPT-3 175B 모델의 96개 레이어 전체에 적용됐다.

실험 결과는 아래 표와 같은데, 특정 어텐션 가중치 유형을 선택하는 것이 다운스트림 성능에 큰 영향을 미친다는 사실을 확인할 수 있다.

	# 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**

모든 파라미터를 단순히 △Wq or △Wk에 할당하면 성능이 크게 저하되고, Wq와 Wv를 동시에 조정하면 최고의 성능을 보이고 있음을 확인해 볼 수 있다. 이는 rank=4 수준에서도 충분한 정보를 학습할 수 있음을 시사하며, **단일 가중치를 높은 rank로 조정하는 것보다 여러 가중치를 낮은 rank로 조정하는 것이 더 효과적임을 알 수 있다.**

B. 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

LoRA는 매우 작은 r 값에서도 경쟁력 있는 성능을 보였는데, 특히 Wq, Wv 조정이 Wq 단독 조정보다 더 효과적이었음을 확인해 볼 수 있다. 이는 업데이트 행렬 W가 매우 작은 intrinsic rank를 가지고 있을 가능성을 시사한다.

또, 서로 다른 r값과 무작위 시드로 학습한 모델들의 서브스페이스를 비교했는데, 결과적으로 r 증가가 더 의미 있는 서브스페이스를 학습한다고 보장할 수 없다는 사실을, **즉 낮은 랭크의 적응 행렬도 충분히 강력할 가능성이 크다는 사실**을 알아냈다.

C. Subspace similarity between different r

r=8과 r=64의 적응 행렬(Adaptation Matrix) A가 학습됐을 때, 이 두 서브 스페이스가 얼마나 유사한가?

이 질문에 답하기 위해 SVD(특이값 분해)를 수행했는데, 이를 통해 right-singular unitary matrices U8, U64를 얻었다. 이후 서브스페이스 포함 관계 분석을 위해 U8의 상위 i개의 특이벡터가 U64 상위 j개 벡터에 얼마나 포함되는지를 확인해야 하는데, 이는 Grassmann 거리 기반 정규화된 서브스페이스 유사도(Normalized Subspace Similarity)로 측정했다.

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

분석 결과 r=8과 r=64의 적응 행렬 A에서 최상위 특이벡터의 방향이 상당 부분 중첩됐다는 사실을 알아냈는데, 특히 A8의 Wv(or Wq) 와 A64의 Wv(or Wq)는 차원 1의 서브스페이스를 공유하는데, 이 서브스페이스의 정규화된 유사도는 0.5 이상이다.

이는 결국 LoRA에서 낮은 랭크로도 충분한 성능을 낼 수 있는 이유이자 저차원 업데이트의 효율성을 보여준다. **즉, 적응 행렬의 정보는 대부분 상위 몇 개의 특이벡터에 집중되기 때문에, 높은 랭크를 사용할 필요 없이 작은 r 값만으로도 충분한 성능을 얻을 수 있다.**

D. Subspace Similarity between different random seeds.

Wq는 다른 실험에서도 공통된 구조를 가지므로 더 높은 랭크를 필요로 할 가능성이 크고, Wv는 비교적 낮은 랭크로도 충분한 정보 표현이 가능함이 확인됐다. 이에, **LoRA를 적용할 때 Wq의 랭크를 상대적으로 더 높게 설정하는 것이 성능 향상에 기여**할 수 있다.

E. How Does The Adaptation Matrix ΔW Compare To W?

논문에서는 ΔW와 W의 correlation에 대해 더 깊게 연구했는데, 이는 사전 학습된 language model을 adapting 하는데 큰 도움이 될 수 있다.

이를 위해 W를 W의 r-rank 부분공간에 투영하여 Frobenius norm 비교를 통해 W의 정보량을 분석했다. 구체적인 과정은 다음과 같다.

구체적인 과정:

1. W를 **r-차원 부분공간에 투영:**
   - **SVD(Singular Value Decomposition)** 를 사용하여 W를 분해:

$$W = U\Sigma V^T$$

   - 여기서:
     - U : 왼쪽 특이벡터 행렬 (Left Singular Vector Matrix)
     - V : 오른쪽 특이벡터 행렬 (Right Singular Vector Matrix)
     - Σ : 대각행렬 (Singular Value Matrix)
2. **투영된 행렬 생성:**
   - W**의 r-차원 부분공간에 투영:**

$$W' = U_r\Sigma_r V_r^T$$

   - 여기서 U_r와 V_r는 W의 **상위 r개의 특이벡터만 사용**하여 구성됨.
3. **Frobenius Norm 비교:**
   - 원본 행렬 W와 투영된 행렬 W' 간의 차이를 비교하기 위해 **Frobenius Norm** 사용:

$$||W' - W||_F$$

   - **비교 대상:**
     - **(1) 상위 r개의 특이벡터를 사용한 경우**
     - **(2) 무작위 행렬을 사용한 경우**

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

위 테이블을 통해, 논문에서는 △W와 W가 강한 상관관계를 가진다는 사실을 알 수 있는데, 이는 △W가 무작위 행렬이 아니라 W에 존재하는 특정 특징을 증폭하는 역할을 한다는 것을 의미한다. **즉, LoRA는 완전히 새로운 정보를 학습하는 것이 아니라 기존에 학습된 정보 중 특정 방향을 강조하는 것이다.**

또, △W는 W의 상위 특이벡터(top singular directions)를 반복하지 않는데, 즉 LoRA는 단순히 기존의 중요한 특성을 그대로 따르는 것이 아니라, 기존 학습된 특징 중 강조되지 않는 부분을 선택적으로 증폭한다는 것을 알 수 있다. 이는 LoRA가 특정 다운스트림 태스크에서 중요한 정보를 활성화하는 역할을 한다는 것을 시사한다.

또, 증폭 계수(amplification factor)가 매우 큰데, 예로 r=4일 때 증폭 계수는 21.5 (6.91/0.32) 에 달하기 때문에 아주 적은 수의 파라미터(r=4)만으로도 원래 모델의 중요한 특

징을 크게 증폭 가능하다. 하지만 r=64에서는 증폭 계수가 작아진다는 것을 확인해 볼 수 있다. 1.87 (3.57/1.90)

마지막으로, Wq의 상위 특이벡터 개수를 늘려가면서 상관관계 변화를 시각적으로 분석한 결과, **LoRA가 특정 방향을 증폭하는 경향이 있다는 것**을 알아냈다.

8. Conclusion and Future Work

LoRA는 거대한 언어 모델을 효과적으로 적응(adaptation)시키는 방법으로,

가. 추론 속도를 저하시키지 않고,
나. 입력 시퀀스 길이를 줄이지 않으며,
다. 모델 성능을 유지하면서도 효율적으로 학습할 수 있는 방법이다.

LoRA의 주요 장점으로는 다음과 같은데,

가. 하드웨어 비용 절감 : 기존의 완전한 미세 조정(fine-tuning) 방식보다 훨씬 적은 연산량과 저장 공간을 요구함
나. 빠른 태스크 전환 : 대부분의 모델 파라미터를 공유하기 때문에, 여러 태스크 간 전환이 빠름
다. Transformer 모델 뿐만 아니라 모든 신경망(Dense Layers 포함)에 적용 가능

남겨진 향후 연구 방향으로는 다음과 같은 것들이 있다.

가. LoRA와 다른 효율적 적응 방법의 조합
나. Fine-tuning과 LoRA 원리 연구
다. LoRA를 적용할 최적의 가중치 행렬 선택 방법 연구
라. W의 랭크(rank) 결핍 연구