LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

Abstract

자연어 처리(NLP)의 주요 패러다임 중 하나는 일반 도메인 데이터에 대한 대규모 pre-training과 특정 과제나 도메인에 대한 적응으로 구성된다. 더 큰 모델을 pre-train할수록, 모든 모델 파라미터를 재학습해야 하는 full fine-tuning은 점차 실현 가능성이 낮아진다. GPT-3 175B를 예로 들면, 각각 175B 파라미터를 갖는 fine-tuned model을 독립적으로 배포하는 것은 비용이 지나치게 크다. 이를 해결하기 위해 우리는 **Lo**w-**R**ank **A**daptation(LoRA)를 제안한다. 이는 사전 학습된 모델 가중치를 고정한 채, Transformer architecture의 각 레이어에 학습 가능한 rank decomposition matrix를 삽입하여 다운스트림 작업에서 학습해야 하는 파라미터 수를 크게 줄여준다. Adam으로 fine-tuning한 GPT-3 175B와 비교했을 때, LoRA는 학습해야 하는 파라미터 수를 최대 10,000배까지 줄이고, GPU 메모리 요구량을 3배 정도 절감한다. 또한 RoBERTa, DeBERTa, GPT-2, GPT-3 등에서 모델 품질 면에서 fine-tuning과 동등하거나 그 이상의 성능을 보이면서도, 더 적은 학습 파라미터와 더 높은 학습 처리량을 제공하며, adapter와 달리 추론 시 추가 지연이 없다. 아울러 언어 모델 적응에서의 rank-deficiency에 대한 실증적 연구를 통해 LoRA의 효과를 입증한다. 우리는 LoRA를 PyTorch 모델에 쉽게 통합할 수 있는 패키지를 공개하며, RoBERTa, DeBERTa, GPT-2에 대한 구현 및 모델 체크포인트를 `https://github.com/microsoft/LoRA`에서 제공한다.

*Equal contribution.

[0]V1과 비교했을 때, 본 초안에는 더 나은 baselines, GLUE 실험, 그리고 adapter latency에 대한 추가 내용이 포함되어 있다.

1 Introduction

자연어 처리(NLP)의 많은 응용 분야에서는 하나의 대규모 사전 학습 언어 모델을 여러 다운스트림 작업에 적응시키는 과정을 필요로 한다. 이러한 적응은 일반적으로 fine-tuning을 통해 이루어지며, 이는 사전 학습된 모델의 모든 파라미터를 업데이트한다. 그러나 fine-tuning의 주요 단점은 새로운 모델이 원본 모델과 동일한 수의 파라미터를 갖게 된다는 점이다. 갈수록 더 큰 모델들이 수개월 주기로 학습됨에 따라, GPT-2 (Radford et al., b)나 RoBERTa large (Liu et al., 2019)에서는 단순한 "불편함"에 지나지 않았던 문제가, 1,750억 개의 학습 가능 파라미터를 갖는 GPT-3 (Brown et al., 2020)에서는 실제 서비스 배포에 치명적인 도전 과제로 바뀌게 되었다.[1]

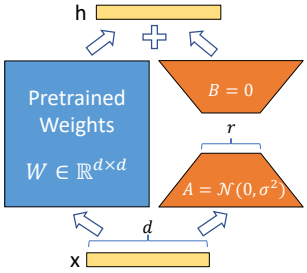

Figure 1: Our reparametrization. We only train A and B.

이 문제를 완화하기 위해, 일부 파라미터만 적응시키거나 새로운 작업을 위한 외부 모듈을 학습하는 방법이 많이 시도되었다. 이 방식은 사전 학습된 모델에 더해 작업별로 아주 적은 수의 파라미터만 저장하고 불러오면 되므로, 배포 시 운영 효율성을 크게 높일 수 있다. 그러나 기존 기술들은 모델의 깊이를 확장하여 추론 지연을 유발하거나 (Houlsby et al., 2019; Rebuffi et al., 2017), 모델이 활용할 수 있는 시퀀스 길이를 줄이는 (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021) 경우가 많다(Section 3). 더욱이 이러한 기법들은 자주 fine-tuning 방식과 동일한 수준의 성능을 달성하지 못하여, 효율성과 모델 품질 사이에서 트레이드오프를 발생시키기도 한다.

Li et al. (2018a); Aghajanyan et al. (2020)에서는 과다하게 파라미터화된 모델이 사실상 낮은 내재적 차원을 가진다는 사실을 보여주었고, 우리는 모델 적응 시 가중치 변화 역시 낮은 "내재적 랭크(intrinsic rank)"를 지닌다고 가정한다. 이를 바탕으로 제안하는 방법이 바로 **Lo**w-**R**ank **A**daptation (LoRA)이다. LoRA는 사전 학습된 가중치를 고정한 채, 적응 과정에서 달라지는 dense layer의 가중치를 랭크 분해(rank decomposition) 행렬로 간접 학습하도록 함으로써, Figure 1에 나타낸 것처럼 일부 dense layer만을 효과적으로 학습할 수 있게 한다. GPT-3 175B를 예로 들어 보면, 전체 랭크(d)가 최대 12,288에 이르더라도 필요한 랭크(r)가 1 또는 2 정도로 매우 작아도 충분함을 확인하였다. 이는 LoRA가 저장 및 연산 양 측면에서 모두 효율적(storage- and compute-efficient)임을 의미한다.

LoRA possesses several key advantages.

- 하나의 사전 학습 모델을 공유하여 다양한 작업을 위한 작은 규모의 LoRA 모듈들을 구축할 수 있다. 사전 학습 모델은 고정한 채, Figure 1의 A와 B 행렬만 교체함으로써 작업을 전환할 수 있으므로, 저장 공간 요구량과 작업 전환 시의 오버헤드를 크게 줄일 수 있다.
- LoRA는 대부분의 파라미터에 대해 기울기를 계산하거나 옵티마이저 상태를 유지할 필요가 없으므로, adaptive optimizer를 사용할 때 학습 효율을 높이고 하드웨어 진입 장벽을 최대 3배 낮춰준다. 대신 주입된, 훨씬 더 작은 랭크 분해 행렬들만 최적화하면 된다.
- 간단한 선형 구조 덕분에 배포 시 학습된 행렬들을 고정된 가중치와 병합할 수 있으며, 이로써 완전한 fine-tuning 모델과 비교했을 때도 추론 지연이 추가로 발생하지 않는다.
- LoRA는 기존의 여러 기법들과 상호 보완적(orthogonal)이므로, 예를 들어 prefix-tuning 등 다양한 방법들과 결합하여 사용할 수 있다. Appendix F에서는 이 예시를 제시하였다.

Terminologies and Conventions 우리는 Transformer architecture를 자주 언급하며, 그 차원에 대한 기존 용어 체계를 그대로 사용한다. Transformer 레이어의 입력 및 출력 차원의 크기를 d_{model}이라 부른다. 자체 어텐션(self-attention) 모듈 내에서 사용되는

[1]GPT-3 175B는 few-shot learning만으로도 무시할 수 없는 성능을 보이지만, Appendix A에서 알 수 있듯이 fine-tuning을 통해 성능이 크게 향상된다.

query/key/value/output projection 행렬들은 각각 W_q, W_k, W_v, W_o라 부른다. W 또는 W_0는 사전 학습된 가중치 행렬을 의미하며, ΔW는 적응 과정에서 누적되는 gradient update를 의미한다. LoRA 모듈의 랭크는 r로 표기한다. 우리는 (Vaswani et al., 2017; Brown et al., 2020)에서 제시한 관례를 따르며, 모델 최적화에는 Adam (Loshchilov & Hutter, 2019; Kingma & Ba, 2017)을 사용하고, Transformer의 MLP feedforward 차원 d_{ffn}은 $4 \times d_{model}$로 설정한다.

2 Problem Statement

비록 우리의 제안 방식은 특정 학습 목표에 구애받지 않지만, 본 논문에서는 언어 모델링을 주된 활용 사례로 삼는다. 아래에서는 언어 모델링 문제, 특히 작업별 프롬프트가 주어졌을 때 조건부 확률을 최대화하는 문제에 대해 간략히 설명한다.

가령, 우리는 사전 학습된 autoregressive language model $P_\Phi(y \mid x)$을 갖고 있다고 하자. 이때 Φ로 매개변수화된 해당 모델은 예를 들어 Transformer architecture (Vaswani et al., 2017)를 기반으로 한 GPT (Radford et al., b; Brown et al., 2020)와 같은 범용 멀티태스크 학습자(generic multi-task learner)가 될 수 있다. 이 사전 학습 모델을 요약(summarization), 기계 독해(MRC), 자연어에서 SQL로의 변환(NL2SQL) 등 다양한 다운스트림 조건부 텍스트 생성 과제에 적응시킨다고 가정해 보자. 각 다운스트림 과제는 컨텍스트-타깃 쌍의 학습 데이터셋 $\mathcal{Z} = \{(x_i, y_i)\}_{i=1,..,N}$을 가지며, 여기서 x_i와 y_i는 토큰 시퀀스로 구성된다. 예컨대 NL2SQL의 경우, x_i는 자연어 질의이고 y_i는 이에 대응되는 SQL 명령이며, 요약 작업에서는 x_i가 문서의 본문, y_i가 그 요약본이다.

full fine-tuning 과정에서는, 모델을 사전 학습된 가중치 Φ_0로 초기화한 뒤, 조건부 언어 모델링 목적을 최대화하도록 기울기를 반복적으로 추적하여 $\Phi_0 + \Delta\Phi$로 업데이트한다:

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

full fine-tuning의 핵심적인 문제점 중 하나는, 각 다운스트림 과제마다 원본 모델과 동일한 차원의 파라미터 집합 $\Delta\Phi$를 학습해야 한다는 것이다. 즉, $\Delta\Phi$의 차원 $|\Delta\Phi|$는 $|\Phi_0|$와 동일하다. 그 결과, 사전 학습 모델의 크기가 매우 큰 경우(예: GPT-3의 경우 $|\Phi_0| \approx 175$ Billion), 이렇게 fine-tuning된 다수의 독립된 모델 인스턴스를 저장 및 배포하는 것은 거의 불가능할 정도로 어려운 과제가 된다.

본 논문에서는 보다 적은 수의 파라미터로도 작업에 특화된 파라미터 증가분 $\Delta\Phi = \Delta\Phi(\Theta)$을 표현할 수 있는 방식, 즉 $|\Theta| \ll |\Phi_0|$를 만족하는 형태로 파라미터 증분을 부호화(encode)하는 방식을 채택한다. 이때 $\Delta\Phi$를 찾는 문제는 곧 Θ에 대한 최적화 문제로 바뀐다:

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi_0+\Delta\Phi(\Theta)}(y_t \mid x, y_{<t})\right) \tag{2}$$

이후 섹션에서는 계산 및 메모리 사용 면에서 효율적인 저랭크 표현(low-rank representation)을 사용하여 $\Delta\Phi$를 부호화할 것을 제안한다. 예를 들어 사전 학습 모델이 GPT-3 175B일 경우, 학습해야 하는 파라미터의 수 $|\Theta|$를 $|\Phi_0|$의 약 0.01% 수준까지 줄일 수 있다.

3 Aren't Existing Solutions Good Enough?

우리가 해결하고자 하는 문제는 전혀 새로운 것이 아니다. Transfer learning이 등장한 이래로, 수많은 연구가 모델 적응 시 필요한 파라미터와 연산량을 줄이기 위해 노력해 왔다. 잘 알려진 여러 연구에 대한 조사 내용은 Section 6를 참조하라. 언어 모델링을 예시로 들면, 효율적인 모델 적응과 관련해서 크게 두 가지 전략이 주목받는다. 하나는 어댑터 레이어(adapter layers)를 추가하는 방법 (Houlsby et al., 2019; Rebuffi et al., 2017; Pfeiffer et al., 2021; Rücklé et al., 2020)이며, 다른 하나는 입력 레이어의 activation 일부 형태를 최적화하는 방법이다 (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021). 그러나 두 전략 모두 대규모이면서 추론 지연(latency)에 민감한 실제 프로덕션 환경에서는 일정한 한계를 가진다.

Adapter Layers Introduce Inference Latency 어댑터에는 다양한 변형 기법이 존재한다. 여기서는 Houlsby et al. (2019)의 원형 설계(Transformer 블록 하나당 두 개의 어댑터 레이어가 있음)와, Lin et al. (2020)의 좀 더 최근 방식(블록당 하나의 어댑터 레이어를 두되 추가적인 LayerNorm (Ba et al., 2016)을 사용하는 방식)에 주목한다. 물론 어댑터 레이어의 일부를 pruning하거나 멀티태스크 환경을 활용하면 전체 지연 시간을 줄일 수는 있지만 (Rücklé et al., 2020; Pfeiffer et al., 2021), 어댑터 레이어에서 발생하는 추가 연산량을 직접적으로 없애는 방법은 없다. 어댑터 레이어가 작은 bottleneck dimension을 두어 종종 원본 모델 파라미터의 <1% 수준을 유지하기 때문에, 추가되는 FLOPs가 제한적이므로 큰 문제가 아닌 것처럼 보일 수 있다. 그러나 대규모 신경망은 하드웨어 병렬화를 통해 지연 시간을 줄이는데, 어댑터 레이어는 순차적으로 처리해야 하므로 이로 인한 지연이 발생한다. 이는 특히 배치 크기가 1처럼 매우 작은 온라인 추론 환경에서 중요한 차이를 만든다. 모델 병렬화가 없는 일반적인 시나리오, 예를 들어 단일 GPU 상에서 GPT-2 (Radford et al., b) medium을 추론하는 경우, 매우 작은 bottleneck dimension을 사용하더라도 어댑터 사용 시 지연이 유의미하게 증가함을 확인할 수 있다(Table 1).

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%)

Table 1: GPT-2 medium에서 단일 forward pass의 inference latency를 밀리초(ms) 단위로 측정하였으며, 100회의 시도에 대한 평균치를 제시한다. We use an NVIDIA Quadro RTX8000. "$|\Theta|$"는 adapter layers에서 학습 가능한 파라미터의 수를 의미한다. $\text{Adapter}^{\text{L}}$ 및 $\text{Adapter}^{\text{H}}$는 각각 Section 5.1에서 설명하는 adapter tuning의 두 가지 변형이다. 온라인 환경에서 시퀀스 길이가 짧은 경우, adapter layers가 유발하는 inference latency는 상당할 수 있다. 보다 자세한 연구 결과는 Appendix B에서 확인할 수 있다.

이 문제는 모델을 Shoeybi et al. (2020); Lepikhin et al. (2020)처럼 shard(샤딩)해야 할 때 더욱 악화된다. 추가적인 depth 때문에, 어댑터 파라미터를 중복하여 여러 번 저장하지 않는 한, 각 어댑터 레이어를 처리하는 과정에서 `AllReduce`나 `Broadcast`와 같은 동기화 GPU 연산이 더 많이 발생하기 때문이다.

Directly Optimizing the Prompt is Hard 이와 반대되는 접근으로서 prefix tuning (Li & Liang, 2021)은 다른 유형의 어려움에 직면한다. 우리는 prefix tuning이 최적화하기가 어렵고, trainable parameters의 크기에 따라 성능이 단조적으로 변하지 않는다는 사실을 관찰하였으며, 이는 원 논문에서 보고된 유사한 결과와 일치한다. 더 근본적으로는, 적응을 위해 시퀀스 길이의 일부를 할당하면 다운스트림 태스크를 처리할 수 있는 시퀀스 길이가 필연적으로 줄어들게 되므로, 이로 인해 prompt를 튜닝하는 방법이 다른 기법들보다 성능이 낮아지는 것으로 추정된다. 태스크 성능에 관한 연구는 Section 5에서 다루기로 한다.

4 Our Method

본 절에서는 LoRA의 간단한 설계와 그 실용적 장점을 설명한다. 이 절에서 제시하는 원리는 일반적인 딥러닝 모델의 모든 dense layer에 적용 가능하지만, 본 연구에서는 동기를 부여하는 사용 사례로서 Transformer language models의 특정 가중치에만 초점을 맞춘다.

4.1 Low-Rank-Parametrized Update Matrices

신경망에는 일반적으로 매트릭스 곱셈을 수행하는 여러 dense layer가 존재한다. 이때 해당 layer들의 가중치 행렬은 보통 full-rank를 갖는다. 특정 태스크에 적응할 때, Aghajanyan et al. (2020)에 따르면, 사전 학습된 language models에는 낮은 "intrinsic dimension"이 존재하며, 더 작은 부분공간으로의 임의 투사에도 불구하고 여전히 효율적으로 학습이 가능하다. 이에 착안하여, 우리는 적응 시점에서 가중치의 업데이트 또한 낮은 "intrinsic rank"를

가진다고 가정한다. 사전 학습된 가중치 행렬 $W_0 \in \mathbb{R}^{d\times k}$에 대해, 이를 $W_0 + \Delta W = W_0 + BA$ 형태의 low-rank 분해로 표현하여 업데이트를 제한한다. 여기서 $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$이며, $r \ll \min(d, k)$이다. 훈련 과정에서 W_0는 고정되어 gradient 업데이트를 받지 않으며, A와 B는 trainable parameters를 포함한다. W_0와 $\Delta W = BA$는 동일한 입력과 곱해진 뒤, 각각의 출력을 좌표별로 합산한다는 점에 유의하라. 즉, $h = W_0x$일 때, 우리의 수정된 forward pass는 다음과 같이 표현된다:

$$h = W_0x + \Delta Wx = W_0x + BAx \tag{3}$$

우리는 Figure 1에서 우리의 재매개변수화(reparametrization)를 도식화하였다. 우리는 A를 랜덤 Gaussian으로 초기화하고 B는 0으로 설정하였는데, 이로써 학습 초기에 $\Delta W = BA$는 0이 된다. 그 후, ΔWx에 대해 $\frac{\alpha}{r}$ 스케일을 적용하는데, 이때 α는 r의 상수이다. Adam을 사용할 때, 적절한 초기화 스케일을 설정한다면 α를 조정하는 것은 학습률을 조정하는 것과 거의 유사하다. 이에 따라, 우리는 단순히 처음 시도하는 r에 α를 맞추고, 이후로 이를 따로 조정하지 않는다. 이러한 스케일링 기법은 r이 달라져도 하이퍼파라미터를 재조정할 필요성을 줄여준다고 알려져 있다 (Yang & Hu, 2021).

A Generalization of Full Fine-tuning. 보다 일반적인 fine-tuning 형태는 사전 학습된 파라미터 중 일부를 훈련에 포함하는 것을 허용한다. LoRA는 한 걸음 더 나아가, 적응 시점에서 가중치 행렬에 적용되는 누적 gradient 업데이트가 반드시 full-rank일 필요는 없도록 한다. 이는 LoRA를 모든 가중치 행렬에 적용하고, 모든 bias[2]를 학습 대상으로 설정할 경우, LoRA의 rank r를 사전 학습된 가중치 행렬의 rank로 맞추어주면 full fine-tuning과 유사한 표현력을 확보할 수 있음을 의미한다. 즉, 학습 가능한 파라미터 수를 늘릴수록[3], LoRA를 학습하는 것은 원래 모델을 학습하는 것에 점차 가까워지게 되며, adapter 기반 기법들은 MLP로, 그리고 prefix 기반 기법들은 긴 입력 시퀀스를 처리할 수 없는 모델로 수렴한다.

No Additional Inference Latency. 운영 환경에 배포할 때, 우리는 $W = W_0 + BA$를 명시적으로 계산하고 저장한 뒤, 평소와 동일하게 inference를 수행할 수 있다. W_0와 BA 모두 $\mathbb{R}^{d\times k}$ 차원을 갖는다. 다른 다운스트림 태스크로 전환해야 할 때에는, W_0에서 BA를 빼고, 그 대신에 $B'A'$를 더하는 방식을 사용할 수 있다. 이는 매우 빠르며 메모리 사용량도 거의 증가하지 않는다. 무엇보다도, 이는 fine-tuned 모델과 비교했을 때 inference 시 추가 지연(latency)을 유발하지 않는다는 점을 보장한다.

4.2 Applying LoRA to Transformer

원칙적으로, 신경망 내 어떠한 부분집합의 가중치 행렬에도 LoRA를 적용하여 학습해야 할 파라미터 수를 줄일 수 있다. Transformer 구조에서는 self-attention 모듈에 네 개의 가중치 행렬(W_q, W_k, W_v, W_o)과 MLP 모듈에 두 개의 가중치 행렬이 존재한다. 출력 차원이 보통 attention head 단위로 분할되기는 하지만, 우리는 W_q (또는 W_k, W_v)를 $d_{model} \times d_{model}$ 차원의 단일 행렬로 취급한다. 본 연구에서는 단순성과 파라미터 효율성을 위해 **오직 attention 가중치만** 다운스트림 태스크에 맞추어 적응시키고, MLP 모듈은 동결하여(즉, 다운스트림 태스크 학습에서 제외하여) 사용한다.Transformer에서 서로 다른 종류의 attention 가중치 행렬을 적응시키는 효과에 대한 추가 연구는 Section 7.1에서 다루며, MLP layer, LayerNorm layer, 그리고 bias를 적응시키는 방안에 대한 경험적 검증은 후속 연구로 남긴다.

Practical Benefits and Limitations. 가장 큰 이점은 메모리와 스토리지 사용량의 감소이다. Adam으로 학습된 대형 Transformer의 경우, $r \ll d_{model}$이라면 동결된 파라미터들에 대한 optimizer state를 저장할 필요가 없어 VRAM 사용량을 최대 $2/3$까지 줄일 수 있다. GPT-3 175B 모델에서는 학습 시 VRAM 사용량이 1.2TB에서 350GB로 감소하였다. 또한 $r = 4$로 설정하고 query와 value 프로젝션 행렬만을 적응시키면, 체크포인트 크기가 약 10,000배(350GB에서 35MB로) 줄어든다[4]. 이로써 훨씬 적은 수의 GPU로도 학습을 수행할 수 있으며, I/O 병목 현상도 방지할 수 있다. 또 다른 이점으로는, 모든 파라미터를 교체하지 않고 LoRA 가중치만 갈아 끼우면 되므로, 배포된 환경에서 태스크 간 전환 비용이 크게 줄어든다는 점을 들 수 있다. 이는 사전 학습된 가중치가 VRAM에 상주한 상태에서, 필요에 따라 즉시 교체할 수 있는 다

[2]가중치에 비해 파라미터 수가 무시할 정도로 적다.

[3]난이도 높은 태스크에 적응할 때 필연적으로 발생한다.

[4]배포 시에는 여전히 350GB 모델이 필요하지만, 100개의 적응된 모델을 저장하려면 350GB + 35MB $\times$ 100 $\approx$ 354GB만 있으면 되고, 이는 100 $\times$ 350GB $\approx$ 35TB에 비해 훨씬 작다.

양한 맞춤형 모델을 생성하는 데에 용이하다. 또한 GPT-3 175B 전체를 fine-tuning하는 것과 비교했을 때, 학습 시 약 25%의 속도 향상이 관측되었는데,[5] 이는 대다수의 파라미터에 대해 gradient를 계산하지 않아도 되기 때문이다.

LoRA 역시 몇 가지 제약이 존재한다. 예컨대, 추가적인 추론 지연을 제거하기 위해 A와 B를 W에 흡수하려는 경우, 서로 다른 A, B를 사용하는 여러 태스크의 입력을 단일 forward pass로 일괄 처리(batch)하기가 간단하지 않다. 다만, 지연(latency)이 핵심이 아닌 시나리오에서는 가중치를 병합하지 않고 배치 내 샘플별로 동적으로 사용할 LoRA 모듈을 선택하는 것도 가능하다.

5 Empirical Experiments

우리는 LoRA를 RoBERTa (Liu et al., 2019), DeBERTa (He et al., 2021), 그리고 GPT-2 (Radford et al., b)에 적용하여 다운스트림 태스크 성능을 평가하고, 이후 이를 GPT-3 175B (Brown et al., 2020) 규모로 확장한다. 본 실험은 자연어 이해(NLU)부터 생성(NLG)에 이르는 광범위한 태스크를 포함한다. 구체적으로, RoBERTa와 DeBERTa 모델은 GLUE (Wang et al., 2019) 벤치마크 위에서 평가한다. GPT-2의 경우, Li & Liang (2021)에서 사용한 설정을 따라 직접 비교를 수행하고, 여기에 더해 GPT-3를 이용한 대규모 실험을 위해 WikiSQL (Zhong et al., 2017) (NL을 SQL 쿼리로 변환)과 SAMSum (Gliwa et al., 2019) (대화 요약) 태스크도 추가하였다. 사용한 데이터셋에 대한 자세한 설명은 Appendix C를 참조하라. 모든 실험은 NVIDIA Tesla V100에서 수행하였다.

5.1 Baselines

다양한 기존 연구와 폭넓게 비교하기 위해, 우리는 이들 연구에서 사용한 실험 설정을 최대한 동일하게 재현하고, 가능할 경우 이미 보고된 수치를 그대로 인용한다. 단, 이로 인해 일부 베이스라인은 특정 실험에만 등장할 수 있음을 밝힌다.

Fine-Tuning (FT)은 다운스트림 태스크 적응에 널리 쓰이는 방법이다. Fine-tuning 시, 모델은 사전 학습된 가중치와 bias로 초기화되고, 모든 모델 파라미터가 gradient 업데이트를 받는다.간단한 변형으로, 특정 레이어만 업데이트하고 나머지는 동결하는 방법이 있다. 우리는 Li & Liang (2021)에서 보고된 GPT-2 베이스라인 중 마지막 두 개 레이어만 적응시키는 기법(**FT$^{\text{Top2}}$**)을 포함한다.

Bias-only 혹은 BitFit은 모델의 다른 모든 파라미터를 동결한 채, bias 벡터만 학습하는 베이스라인이다. 이 베이스라인은 BitFit (Zaken et al., 2021)에서도 동시대에 연구되었다.

Prefix-embedding tuning (PreEmbed)은 입력 토큰 사이에 특별한 토큰을 삽입하는 기법이다. 이 특별한 토큰들은 모델의 기존 vocabulary에는 없는 trainable word embedding을 갖는다. 이 토큰들을 어디에 배치하느냐가 성능에 영향을 미칠 수 있는데, 본 연구에서는 Li & Liang (2021)의 논의에 따라 prefix(문장 앞)와 infix(문장 내) 방식에 초점을 맞춘다. l_p (prefix 토큰 수), l_i (infix 토큰 수)로 두 방식을 나타낼 때, 학습해야 하는 파라미터 수는 $|\Theta| = d_{model} \times (l_p + l_i)$와 같다.

Prefix-layer tuning (PreLayer)는 prefix-embedding tuning을 확장한 기법이다. 특수 토큰의 word embedding(또는 임베딩 레이어 이후의 activation)만 학습하는 대신, 각 Transformer 레이어 이후의 activation을 학습한다. 즉, 이전 레이어에서 계산된 activation을 통째로 trainable한 값으로 대체하는 방식이다. 이에 따른 학습 파라미터 수는 $|\Theta| = L \times d_{model} \times (l_p + l_i)$이며, 여기서 L은 Transformer 레이어의 수를 의미한다.

Adapter tuning은 Houlsby et al. (2019)에서 제안된 기법으로, self-attention 모듈(및 MLP 모듈)과 그 이후의 residual connection 사이에 adapter layer를 삽입한다. adapter layer에는 활성함수로 구분된 두 개의 fully connected 레이어와 bias가 포함된다. 우리는 이 원래 설계를 **Adapter$^{\text{H}}$**라고 칭한다. 최근에는 Lin et al. (2020)이 더욱 효율적인 방식으로, MLP 모듈 이후와 LayerNorm 이후에만 adapter layer를 적용하는 기법을 제안하였다. 우리는 이를 **Adapter$^{\text{L}}$**라 칭한다. 이는 Pfeiffer et al. (2021)에서 제안된 또 다른 설계(**Adapter$^{\text{P}}$**)와도 매우 유사하다. 더 나아가, 일부 adapter 레이어를 생략하여 효율성을 높이는 AdapterDrop (Rücklé

[5] GPT-3 175B 전체를 fine-tuning할 경우 V100 GPU당 학습 처리량은 초당 32.5 토큰이며, 같은 모델 병렬화 환경에서 LoRA를 사용할 때는 V100 GPU당 초당 43.1 토큰의 처리량을 달성하였다.

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**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, 그리고 $\text{DeBERTa}_{\text{XXL}}$를 GLUE 벤치마크에서 다양한 적응 방법으로 비교한 결과이다. MNLI에 대해서는 overall (matched와 mismatched) accuracy, CoLA는 Matthew's correlation, STS-B는 Pearson correlation, 그리고 다른 태스크들은 accuracy를 보고한다. 모든 지표에서 값이 클수록 성능이 좋음을 의미한다. *는 기존 연구에서 보고된 수치를 나타낸다. †는 공정한 비교를 위해 Houlsby et al. (2019)와 유사한 설정을 사용한 실험을 의미한다.

et al., 2020)(**$\text{Adapter}^{\text{D}}$**)도 포함한다. 가급적 여러 기존 연구에서 보고된 수치를 그대로 인용하여 비교 대상을 최대화하고자 하며, 이 경우 해당 행의 첫 번째 열에 별표(*)를 표시하였다. 모든 경우, 학습해야 하는 파라미터 수는 $\hat{L}_{Adpt}$개의 adapter 레이어와 $\hat{L}_{LN}$개의 trainable LayerNorm에 대해

$$|\Theta| = \hat{L}_{Adpt} \times \left(2 \times d_{model} \times r + r + d_{model}\right) \; + \; 2 \times \hat{L}_{LN} \times d_{model}$$

로 계산된다(예: $\text{Adapter}^{\text{L}}$에서 $\hat{L}_{Adpt}$는 적용된 adapter 레이어 수, $\hat{L}_{LN}$은 학습 대상 LayerNorm 수).

LoRA는 기존 가중치 행렬에 병렬로 rank 분해 행렬의 쌍을 학습 파라미터로 추가한다. Section 4.2에서 언급한 바와 같이, 대부분의 실험에서는 단순화를 위해 W_q와 W_v에만 LoRA를 적용한다. 학습해야 하는 파라미터 수는 r과 원래 가중치의 크기에 따라 결정되며, 구체적으로 $|\Theta| = 2 \times \hat{L}_{LoRA} \times d_{model} \times r$이다. 여기서 $\hat{L}_{LoRA}$는 LoRA가 적용된 가중치 행렬의 수다.

5.2 RoBERTa base/large

RoBERTa (Liu et al., 2019)는 BERT (Devlin et al., 2019a)에서 제안된 사전 학습 방식을 최적화하여, 많은 수의 추가 학습 파라미터를 도입하지 않고도 모델의 태스크 성능을 향상시켰다. 최근 수년간 GLUE 벤치마크 (Wang et al., 2019) 등의 NLP 리더보드에서는 이보다 훨씬 규모가 큰 모델들이 두각을 나타냈지만, RoBERTa는 여전히 그 크기에 비해 충분히 경쟁력 있고 실무적으로도 인기 있는 사전 학습 모델로 자리매김하고 있다. 본 연구에서는 HuggingFace Transformers 라이브러리 (Wolf et al., 2020)에서 제공하는 RoBERTa base (125M)와 RoBERTa large(355M)를 가져와, GLUE 벤치마크의 다양한 태스크에 대해 여러 파라미터 효율적(adaptation) 기법의 성능을 평가한다. 또한, Houlsby et al. (2019) 및 Pfeiffer et al. (2021)의 설정을 가능한 한 동일하게 재현하였다. Adapters와 비교할 때 LoRA의 성능을 공정하게 평가하기 위해 중요한 두 가지 변경 사항을 적용하였다. 첫째, 모든 태스크에서 동일한 배치 크기를 사용하고, adapter 베이스라인과 동일하게 시퀀스 길이를 128로 설정하였다. 둘째, MRPC, RTE, STS-B를 학습 시킬 때, fine-tuning 베이스라인에서처럼 이미 MNLI에 적응된 모델이 아닌, 사전 학습 상태의 모델로부터 초기화를 진행하였다. 이와 같이 Houlsby et al. (2019)의 더 제한된 설정을 따른 실험 결과에는 † 기호를 표시한다. 이 결과는 Table 2 (상위 세 섹션)에 제시하였다. 하이퍼파라미터에 관한 세부 사항은 Section D.1를 참고하라.

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) 및 large (L) 모델에 대해, E2E NLG Challenge에서 서로 다른 적응 기법을 적용한 결과이다. 모든 지표에서 값이 높을수록 성능이 우수함을 나타낸다. LoRA는 유사하거나 더 적은 학습 파라미터를 사용하면서도 여러 베이스라인을 능가하는 성능을 보인다. 본 연구에서 수행한 실험에는 신뢰 구간을 표기하였으며, *는 기존 연구에서 보고된 수치를 의미한다.

5.3 DeBERTa XXL

DeBERTa (He et al., 2021)는 훨씬 더 대규모로 학습된, 최근에 제안된 BERT 변형 모델이며, GLUE (Wang et al., 2019)와 SuperGLUE (Wang et al., 2020)와 같은 벤치마크에서도 매우 경쟁력 있는 성능을 보인다. 우리는 LoRA가 GLUE에서 완전히 fine-tuning된 DeBERTa XXL (1.5B)의 성능에 여전히 근접할 수 있는지 평가하였다. 해당 결과는 Table 2 (하단 섹션)에 제시하였다. 하이퍼파라미터에 대한 자세한 설명은 Section D.2를 참조하라.

5.4 GPT-2 medium/large

앞선 NLU 실험을 통해 LoRA가 full fine-tuning에 근접하는 경쟁력 있는 대안이 될 수 있음을 확인하였다. 이번에는 GPT-2 medium, large (Radford et al., b)와 같은 NLG 모델에서도 LoRA의 효과가 여전히 유지되는지 살펴보고자 한다. 직접적인 비교를 위해, 우리는 가능한 한 Li & Liang (2021)와 유사한 실험 설정을 유지하였다. 지면 관계상, 본문에서는 E2E NLG Challenge 결과(Table 3)만 제시한다. WebNLG (Gardent et al., 2017) 및 DART (Nan et al., 2020) 실험 결과는 **??**에서 확인할 수 있다. 하이퍼파라미터 목록은 Section D.3를 참고하라.

5.5 Scaling up to GPT-3 175B

LoRA의 성능을 한계치까지 시험하기 위해, 우리는 파라미터가 1750억 개인 GPT-3 모델로 규모를 확장하였다. 학습 비용이 매우 높으므로, 각각의 항목별 오차 범위 대신, 주어진 태스크에 대해 랜덤 시드로 인한 일반적인 표준편차만을 보고한다. 사용된 하이퍼파라미터에 대한 자세한 사항은 Section D.4를 참조하라.

Table 4에서 볼 수 있듯이, LoRA는 세 가지 데이터셋 모두에서 fine-tuning 베이스라인의 성능에 근접하거나 이를 상회하였다. 흥미롭게도, Figure 2에서 나타나듯, 모든 기법이 학습해야 할 파라미터 수가 많아질수록 성능이 단조롭게 향상되는 것은 아니다. 예컨대 prefix-embedding tuning에서는 256개 이상의 특수 토큰을, prefix-layer tuning에서는 32개 이상의 특수 토큰을 사용했을 때 성능이 오히려 상당 폭 하락하였다. 이는 Li & Liang (2021)에서도 보고된 유사한 관찰과 일치하며, 이 현상의 심층적인 원인은 본 연구 범위를 넘어가지만, 특수 토큰이 많아질수록 입력 분포가 사전 학습 데이터 분포와 더 크게 달라지기 때문이라고 추정된다. 한편, 적은 양의 데이터가 주어졌을 때 다양한 적응 기법이 어떻게 성능을 보이는지에 대한 연구는 Section F.2에서 별도로 다루었다.

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

Table 4: GPT-3 175B에 대해 다양한 적응 기법을 적용한 결과이다. WikiSQL은 논리 형태에 대한 검증 정확도(Validation Accuracy)를, MultiNLI-matched는 검증 정확도를, SAMSum은 Rouge-1/2/L 지표를 보고한다. LoRA는 full fine-tuning을 비롯한 기존 기법들보다 우수한 성능을 보인다. WikiSQL 결과는 약 $\pm 0.5\%$, MNLI-m은 약 $\pm 0.1\%$, SAMSum 결과는 세 가지 지표 각각에 대해 약 ± 0.2, ± 0.2, ± 0.1 정도의 변동 폭을 보였다.

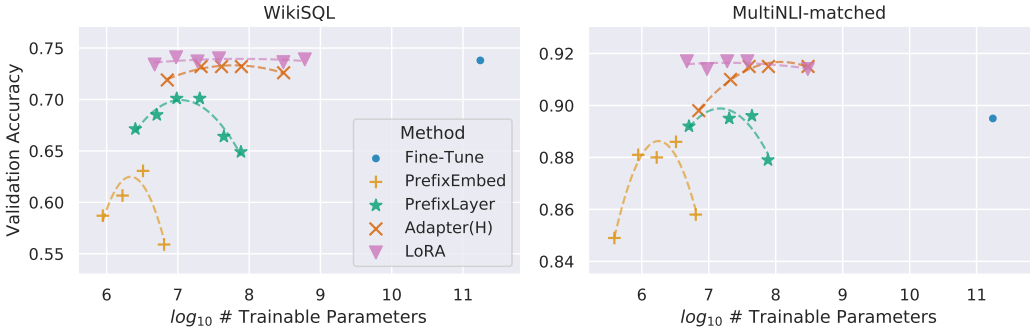

Figure 2: GPT-3 175B에 대해 WikiSQL과 MNLI-matched에서의 검증 정확도(Validation Accuracy)를 학습해야 할 파라미터 수에 따라 비교한 결과이다. LoRA는 보다 우수한 확장성과 태스크 성능을 보인다. 플롯된 데이터 포인트에 대한 자세한 설명은 Section F.1를 참조하라.

6 Related Works

Transformer Language Models. Transformer (Vaswani et al., 2017)는 self-attention을 대대적으로 활용하는 sequence-to-sequence 구조이다. Radford et al. (a)는 Transformer 디코더를 여러 층으로 쌓아올려 이를 자동회귀(autoregressive) 방식의 language modeling에 적용하였다. 이후로 Transformer 기반 language model은 NLP 전반을 지배하며, 다양한 태스크에서 최신(SOTA) 성능을 달성해 왔다. BERT (Devlin et al., 2019b)와 GPT-2 (Radford et al., b) 이후, 방대한 양의 텍스트로 학습된 대형 Transformer language model을 일반 도메인 데이터로 먼저 사전 학습한 뒤, 태스크별 데이터로 추가 학습(fine-tuning)하는 새로운 패러다임이 등장하였다. 이는 특정 태스크용 데이터를 바로 학습하는 것보다 큰 성능 향상을 가져온다. 또한, Transformer의 크기를 키우면 성능이 꾸준히 향상되기 때문에, 대규모 모델을 연구하는 방향이 계속 진행되어 왔다. GPT-3 (Brown et al., 2020)는 현재까지 단일 Transformer 언어 모델로서는 가장 큰 규모인 1750억 개의 파라미터를 갖고 있다.

Prompt Engineering and Fine-Tuning. GPT-3 175B는 적은 수의 추가 학습 예시만으로도 모델의 동작을 부분적으로 바꿀 수 있으나, 이 성능은 입력 프롬프트(prompt)에 크게 의존한다 (Brown et al., 2020). 이는 모델이 특정 태스크에서 최대 성능을 내도록 프롬프트를 작성하고 형식을 구성하는 데 많은 시행착오가 필요함을 의미하며, 이를 일반적으로 prompt engineering 또는 prompt hacking이라고 부른다. Fine-tuning은 일반 도메인에서 사전 학습된 모델을 특정 태스크에 맞춰 재학습시키는 과정이다 Devlin et al. (2019b); Radford et al. (a). 모델 파라미터의 일부만 학습하는 변형 기법도 제안되어 왔지만 Devlin et al. (2019b); Collobert & Weston (2008), 실무에서는 대부분 다운스트림 성능을 극대화하기 위해 전체

파라미터를 재학습시키는 방식을 채택한다. 그러나 GPT-3 175B는 체크포인트 자체가 매우 크며, 사전 학습 시와 동일한 수준의 메모리 용량이 필요하므로 일반적인 방식으로 fine-tuning 하기에 높은 하드웨어 장벽이 존재한다.

Parameter-Efficient Adaptation. 여러 연구에서 기존 레이어들 사이에 adapter 레이어를 삽입하여 학습 파라미터를 효율적으로 운용하는 방법을 제안하였다 (Houlsby et al., 2019; Rebuffi et al., 2017; Lin et al., 2020). 우리의 방법 또한 유사한 병목 구조(bottleneck structure)를 사용하여 가중치 업데이트에 대한 low-rank 제약을 부여한다. 핵심적인 기능적 차이는, 학습된 가중치를 추론 시점에서 기존 가중치에 병합(merge)할 수 있어 추론 지연(latency)을 유발하지 않는다는 점이며, 이는 adapter 레이어들과는 다른 특징이다(Section 3). 최근에는 compacter (Mahabadi et al., 2021)라는 adapter 기법이 제안되었는데, 이는 Kronecker 곱과 특정 가중치 공유(scheme)를 활용하여 adapter 레이어를 효과적으로 파라미터화한다. 유사하게, LoRA를 다른 텐서 곱 기반 기법과 결합한다면 파라미터 효율성을 더욱 향상시킬 가능성이 있으며, 이는 향후 연구 과제로 남겨 둔다. 한편, fine-tuning 대신 입력 word embedding 자체를 최적화하여, 프롬프트를 연속적이고 미분 가능한 방식으로 일반화하는 연구도 최근 많이 진행되고 있다 (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021). 본 논문에서는 그 중 Li & Liang (2021)와 비교 실험을 수행하였다. 하지만 이 방향의 연구는 대체로 프롬프트에 많은 특수 토큰을 추가하여 확장하는 방식에 의존하는데, positional embedding을 학습하는 상황에서는 이러한 특수 토큰들이 실제 태스크 토큰이 사용할 수 있는 시퀀스 길이를 잠식하게 된다.

Low-Rank Structures in Deep Learning. 딥러닝에서 저랭크(low-rank) 구조는 매우 흔하게 나타난다. 다수의 머신러닝 문제는 내재적으로 저랭크 구조를 갖고 있는 것으로 알려져 있다 (Li et al., 2016; Cai et al., 2010; Li et al., 2018b; Grasedyck et al., 2013). 더욱이, 파라미터가 매우 많아 과적합(over-parameterized)된 신경망에서 특히, 학습 후 저랭크 성질을 가지게 되는 경우가 많다는 사실도 보고되어 있다 (Oymak et al., 2019). 일부 선행 연구에서는 원래 신경망을 학습할 때부터 저랭크 제약을 명시적으로 부여하기도 한다 (Sainath et al., 2013; Povey et al., 2018; Zhang et al., 2014; Jaderberg et al., 2014; Zhao et al., 2016; Khodak et al., 2021; Denil et al., 2014). 그러나 우리의 인지 범위 내에서는, 이러한 연구들 중 어떠한 것도 다운스트림 태스크에 적응하기 위해 동결된 모델에 저랭크 업데이트를 적용하는 방법은 다루지 않는다. 이론 연구 분야에서, 기반 개념(underlying concept class)에 특정 저랭크 구조가 존재할 경우, 신경망이 해당 구조의 (유한 폭) 뉴럴 탄젠트 커널(neural tangent kernel) (Allen-Zhu et al., 2019; Li & Liang, 2018)을 포함한 다른 고전적 학습 기법들보다 더 나은 성능을 보인다는 사실이 알려져 있다 (Ghorbani et al., 2020; Allen-Zhu & Li, 2019; Allen-Zhu & Li, 2020a). Allen-Zhu & Li (2020b)에서는 저랭크 기반 적응 기법이 적대적 학습(adversarial training)에서도 유용할 수 있음을 이론적으로 시사한다. 종합하자면, 본 논문에서 제안하는 저랭크 방식의 적응 업데이트는 여러 선행 연구로부터 충분한 동기를 부여받았다고 할 수 있다.

7 Understanding the Low-Rank Updates

LoRA가 실증적으로 보여준 장점을 기반으로, 우리는 다운스트림 태스크로부터 학습된 저랭크 적응의 속성을 더욱 명확히 설명하고자 한다. 저랭크 구조는 하드웨어적 진입 장벽을 낮추어 병렬로 다수의 실험을 수행할 수 있게 해줄 뿐 아니라, 업데이트 가중치와 사전 학습된 가중치 간의 상관관계를 이해하기에도 유리하다. 본 연구에서는 특히 GPT-3 175B에 초점을 맞추었는데, 이 모델은 학습해야 할 파라미터 수를 최대 10,000배까지 줄이면서도 태스크 성능에 거의 영향을 주지 않는 결과를 달성하였다.

우리는 일련의 실험적 연구를 통해 다음과 같은 질문에 답하고자 한다: 1) 파라미터 예산(parameter budget)이 제한된 상황에서, 사전 학습된 Transformer의 어떤 가중치 행렬 부분집합을 적응시켜야 다운스트림 성능이 최대화되는가? 2) "최적" 적응 행렬인 ΔW는 정말로 저랭크(rank-deficient)인가? 그렇다면, 실제 활용 시 어떤 랭크(rank) 수준을 사용하는 것이 바람직한가? 3) ΔW와 W 사이에는 어떤 연관성이 있는가? ΔW는 W와 높은 상관관계를 갖는가? 또한 ΔW의 크기는 W 대비 어느 정도인가?

우리의 (2)번, (3)번 질문에 대한 답변은, 사전 학습된 언어 모델을 다운스트림 태스크에 활용하는 근본적인 원리를 이해하는 데 중요한 통찰을 제공한다고 믿는다. 이는 NLP에서 매우 핵심적인 주제이다.

7.1 Which Weight Matrices in Transformer Should We Apply LoRA to?

주어진 제한된 파라미터 예산 안에서, 다운스트림 태스크에서 최적의 성능을 얻기 위해서는 Transformer의 어떤 유형의 가중치에 LoRA를 적용해야 할까? Section 4.2에서 언급했듯이, 우리는 self-attention 모듈에 있는 가중치 행렬들에만 초점을 맞춘다. GPT-3 175B 모델에 대해 파라미터 예산을 1,800만 개(이를 FP16 형태로 저장할 경우 대략 35MB)에 맞추었으며, 이는 96개 모든 레이어에서 한 가지 유형의 attention 가중치만 적응시킬 경우 $r = 8$, 두 가지 유형을 적응시킬 경우 $r = 4$에 해당한다. 결과는 Table 5에 제시하였다.

	# 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 ($\pm 0.5\%$)	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**

Table 5: GPT-3에서 다양한 유형의 attention 가중치에 LoRA를 적용하되, 동일한 학습 파라미터 수를 가정했을 때 WikiSQL과 MultiNLI에서 측정한 검증 정확도를 나타낸다. W_q와 W_v를 동시에 적응시키는 경우 전반적으로 가장 우수한 성능을 보인다. 무작위 시드에 따른 표준편차는 데이터셋별로 일관되었으며, 그 값을 첫 번째 열에 기재하였다.

유의할 점은 모든 파라미터를 ΔW_q 혹은 ΔW_k에만 할당할 경우 성능이 크게 저하되며, W_q와 W_v를 동시에 적응시키는 방식이 가장 좋은 결과를 낸다는 점이다. 이는 rank가 4에 불과해도 ΔW가 충분한 정보를 포착할 수 있으며, 결과적으로 단 하나의 가중치 유형에 더 큰 rank를 할당하기보다는 여러 가중치 행렬을 적응시키는 편이 유리함을 시사한다.

7.2 What is the Optimal Rank r for LoRA?

이번에는 모델 성능에 대한 rank r의 영향을 살펴본다. 비교를 위해, 우리는 $\{W_q, W_v\}$, $\{W_q, W_k, W_v, W_c\}$, 그리고 W_q만 적응시키는 경우를 고려한다.

	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

Table 6: WikiSQL과 MultiNLI에서 서로 다른 rank r로 학습했을 때의 검증 정확도이다. 흥미롭게도, W_q와 W_v를 동시에 적응시키는 경우, rank가 1처럼 매우 작아도 두 데이터셋에서 충분히 성능을 유지한다. 반면 W_q만 학습할 경우에는 더 큰 r가 필요하다. 유사한 실험을 GPT-2에서도 진행하였으며, 이는 Section H.2에서 확인할 수 있다.

Table 6에서 확인할 수 있듯, 놀랍게도 LoRA는 매우 작은 r 값(특히 W_q만 학습할 때보다 W_q와 W_v를 동시에 학습할 때)이더라도 이미 상당히 경쟁력 있는 성능을 보인다. 이는 업데이트 행렬 ΔW가 지니는 "내재적 랭크(intrinsic rank)"가 실제로 매우 작을 수 있음을 시사한다.[6] 이를 좀 더 뒷받침하기 위해, 우리는 서로 다른 r 값 및 무작위 시드로 학습된 서브스페이스(subspace)가 얼마나 유사한지 확인하였다. 즉, r를 증가시키더라도 보다 "의미 있는" 서브스페이스를 추가로 포착하는 것이 아니라면, 낮은 랭크의 적응 행렬로도 충분함을 시사한다.

Subspace similarity between different r. 동일한 사전 학습된 모델을 사용하여, $r = 8$ 및 $r = 64$로 각각 학습된 적응 행렬 $A_{r=8}$, $A_{r=64}$를 고려하자. 이 행렬들에 대해 특이값 분해

[6] 하지만 모든 태스크나 데이터셋에서 작은 r가 늘 효과적이라고 단정 지을 수는 없다. 예컨대, 다운스트림 태스크가 사전 학습에 사용된 언어와 전혀 다른 언어라면, 전체 모델을 재학습하는(즉, $r = d_{model}$로 LoRA를 적용하는) 방식이 작은 r를 사용하는 LoRA보다 월등한 성능을 낼 수 있다.

(SVD)를 수행하여, 우측 특이벡터에 해당하는 유니터리 행렬 $U_{A_{r=8}}$와 $U_{A_{r=64}}$를 얻는다.[7] 이제, $U_{A_{r=8}}$에서 상위 i개의 특이벡터($1 \le i \le 8$)가 span하는 서브스페이스가, $U_{A_{r=64}}$에서 상위 j개의 특이벡터($1 \le j \le 64$)가 span하는 서브스페이스에 어느 정도로 포함되는지를 측정하고자 한다. 이를 위해 Grassmann 거리 기반의 정규화된 서브스페이스 유사도를 정의한다(자세한 내용은 Appendix G 참조):

$$\phi(A_{r=8}, A_{r=64}, i, j) = \frac{\|U^{i\top}_{A_{r=8}} U^{j}_{A_{r=64}}\|_F^2}{\min(i, j)} \in [0, 1], \tag{4}$$

여기서 $U^{i}_{A_{r=8}}$는 $U_{A_{r=8}}$에서 상위 i개의 특이벡터에 대응하는 열(column)을 의미한다.

$\phi(\cdot)$의 값은 $[0, 1]$ 범위를 갖는데, 1이면 두 서브스페이스가 완전히 겹치는 것이고, 0이면 완전히 분리됨을 의미한다. Figure 3는 i, j 값을 변화시키며 ϕ가 어떻게 달라지는지를 보여준다. 지면 제약으로 인해 96개 레이어 중 48번째 레이어만을 예시로 보이지만, Section H.1에서 확인할 수 있듯 동일한 결론이 다른 레이어에도 유효하다.

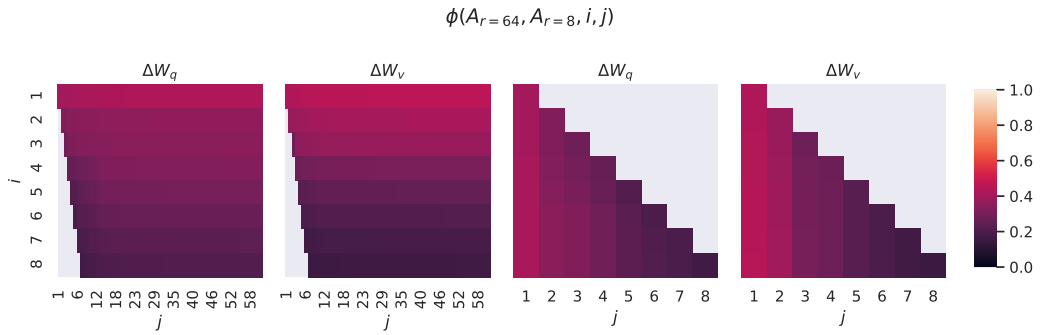

Figure 3: GPT-3에서 $r = 8$ 및 $r = 64$로 학습된 ΔW_q, ΔW_v의 A 행렬에 대해, 두 행렬의 열벡터 서브스페이스 간 유사도를 나타낸 그래프이다. 오른쪽 두 그래프는 왼쪽 두 그래프의 왼쪽 아래 삼각형 부분을 확대하였다. $r = 8$의 최상위 방향(top directions)은 $r = 64$의 서브스페이스에 포함되고, 그 반대도 마찬가지이다.

우리는 Figure 3에서 중요한 관찰을 얻을 수 있다.

> Directions corresponding to the top singular vector overlap significantly between $A_{r=8}$ and $A_{r=64}$, while others do not. Specifically, ΔW_v (resp. ΔW_q) of $A_{r=8}$ and ΔW_v (resp. ΔW_q) of $A_{r=64}$ share a subspace of dimension 1 with normalized similarity > 0.5, providing an explanation of why $r = 1$ performs quite well in our downstream tasks for GPT-3.

Since both $A_{r=8}$ and $A_{r=64}$ are learned using the same pre-trained model, Figure 3 indicates that the top singular-vector directions of $A_{r=8}$ and $A_{r=64}$ are the most useful, while other directions potentially contain mostly random noises accumulated during training. Hence, the adaptation matrix can indeed have a very low rank.

Subspace similarity between different random seeds. We further confirm this by plotting the normalized subspace similarity between two randomly seeded runs with $r = 64$, shown in Figure 4. ΔW_q appears to have a higher "intrinsic rank" than ΔW_v, since more common singular value directions are learned by both runs for ΔW_q, which is in line with our empirical observation in Table 6. As a comparison, we also plot two random Gaussian matrices, which do not share any common singular value directions with each other.

7.3 How Does the Adaptation Matrix ΔW Compare to W?

이제 ΔW와 원래 가중치 행렬 W 간의 관계를 좀 더 깊이 살펴본다. 구체적으로, ΔW가 W와 강하게 상관되어 있는가(즉, 수학적으로는 ΔW가 W의 상위 특이벡터 방향들에 주로 포함되

[7] 유사한 분석을 B에도 적용하여 왼쪽 특이벡터에 해당하는 유니터리 행렬을 비교할 수도 있으나, 본 실험에서는 A 행렬만을 대상으로 삼는다.

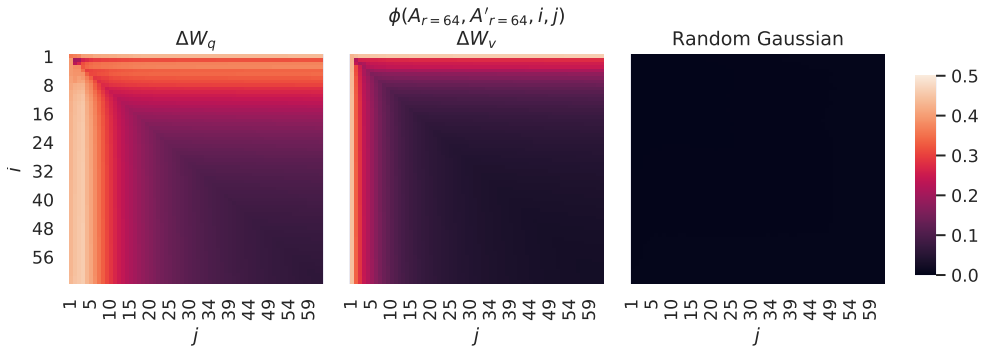

Figure 4: **Left and Middle:** Normalized subspace similarity between the column vectors of $A_{r=64}$ from two random seeds, for both ΔW_q and ΔW_v in the 48-th layer. **Right:** the same heat-map between the column vectors of two random Gaussian matrices. See Section H.1 for other layers.

어 있는지), 또한 ΔW의 "크기"가 W의 해당 방향들과 비교했을 때 어느 정도인지 궁금하다. 이는 사전 학습 언어 모델을 다운스트림 태스크로 어떻게 적응시키는지에 대한 작동 원리를 이해하는 데 도움이 된다.

이 질문들에 답하기 위해, 우리는 W를 ΔW가 형성하는 r차원 서브스페이스에 사영(projection)하는 방식을 사용한다. 우선, ΔW에 대해 특이값 분해(SVD)를 수행하여 왼쪽/오른쪽 특이벡터 행렬 U와 V를 구한 뒤, $U^\top W V^\top$를 계산한다. 그 다음, $\|U^\top W V^\top\|_F$와 $\|W\|_F$ (Frobenius 노름)을 비교하여, W가 어느 정도로 ΔW의 서브스페이스와 정렬되어 있는지를 평가한다. 비교 목적으로, W의 상위 r개 특이벡터 방향(특이값 분해 시 얻는 고유의 U, V)을 사용하거나 무작위 행렬을 사용하여 동일한 과정을 반복하고, 그 결과인 $\|U^\top W V^\top\|_F$도 함께 측정한다.

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

Table 7: GPT-3의 48번째 레이어에서 W_q를 대상으로, U와 V가 (1) ΔW_q, (2) W_q, 또는 (3) 무작위 행렬의 상위 r개 특이벡터 방향일 때, $U^\top W_q V^\top$의 Frobenius 노름을 비교한 결과이다.

우리는 Table 7에서 여러 가지 결론을 도출할 수 있다. 첫째, ΔW는 무작위 행렬과 비교할 때 W와 더 강한 상관관계를 가진다. 이는 ΔW가 이미 W 안에 존재하는 특정 특징(feature)을 더욱 증폭(amplify)하고 있음을 의미한다. 둘째, ΔW는 W의 상위 특이벡터 방향을 단순히 반복하기보다는, W가 크게 강조하지 않는 방향을 선택적으로 증폭하고 있음을 알 수 있다. 셋째, 증폭 계수(amplification factor)는 상당히 크다. 예컨대 $r = 4$인 경우 $21.5 \approx 6.91/0.32$로 나타난다. $r = 64$에서 증폭 계수가 더 작게 나타나는 이유에 대해서는 Section H.4를 참고하라. 또한 Section H.3에서는 W_q의 상위 특이벡터 방향을 점차 늘려가며, 그에 따라 상관관계가 어떻게 변하는지 시각적으로 보여준다. 결과적으로, 저랭크 적응 행렬은 사전 학습 과정에서 학습되었지만 일반 도메인 모델에서는 크게 부각되지 않았던 중요한 특징들을 다운스트림 태스크에 맞추어 증폭하는 역할을 하고 있음을 시사한다.

8 Conclusion and Future Work

거대한 언어 모델을 완전히 fine-tuning하는 것은 필요한 하드웨어 측면에서나, 서로 다른 태스크마다 독립된 인스턴스를 호스팅해야 하는 스토리지/스위칭 비용 측면에서 모두 지나치게 큰 부담을 야기한다. 이에 우리는 LoRA를 제안하였다. LoRA는 추론(Inference) 시 지연(latency)을 추가로 유발하거나 입력 시퀀스 길이를 축소하지 않으면서, 높은 모델 품질을 유지하도록 설계된 효율적인 적응 전략이다. 더욱이, 모델 파라미터의 대다수를 공유함으로써, 서비스 환경에서 태스크 전환 시 빠르게 모델을 교체(switching)할 수 있도록 지원한다. 본

논문에서는 Transformer 언어 모델을 중심으로 살폈으나, 제안하는 원리는 dense layer를 갖는 모든 신경망 구조에 일반적으로 적용할 수 있다.

앞으로의 연구 방향은 다양하다. 1) LoRA는 다른 파라미터 효율적 적응 기법과도 결합하여 상호보완적인 성능 향상을 가져올 수 있을 것이다. 2) 아직 fine-tuning이나 LoRA의 작동 메커니즘은 완전히 명확하지 않다. 구체적으로, 사전 학습 과정에서 학습된 특징(feature)이 어떻게 다운스트림 태스크를 잘 수행하도록 변형되는가에 대한 근본적인 이해가 필요하다. 우리는 LoRA가 전체 파라미터를 학습하는 fine-tuning보다 이러한 문제를 더 다루기 쉽도록 만들 수 있다고 본다. 3) 본 논문에서는 LoRA를 적용할 가중치 행렬을 선택할 때 휴리스틱에 크게 의존하였는데, 이를 더 원칙적으로 결정할 수 있는 방법이 있을지도 연구 과제로 남아 있다. 4) 마지막으로, ΔW의 랭크가 매우 작다는 점은 W 자체도 저랭크일 수 있음을 시사하며, 이는 향후 연구에 새로운 시사점을 제공할 수 있다.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. arXiv:2012.13255 [cs], December 2020. URL http://arxiv.org/abs/2012.13255.

Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Efficiently, Going Beyond Kernels? In NeurIPS, 2019. Full version available at http://arxiv.org/abs/1905.10337.

Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. arXiv preprint arXiv:2001.04413, 2020a.

Zeyuan Allen-Zhu and Yuanzhi Li. Feature purification: How adversarial training performs robust deep learning. arXiv preprint arXiv:2005.10190, 2020b.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In ICML, 2019. Full version available at http://arxiv.org/abs/1811.03962.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs], July 2020. URL http://arxiv.org/abs/2005.14165.

Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. SIAM Journal on optimization, 20(4):1956–1982, 2010.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), 2017. doi: 10.18653/v1/s17-2001. URL http://dx.doi.org/10.18653/v1/S17-2001.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, ICML '08, pp. 160–167, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019a.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs], May 2019b. URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005), 2005. URL https://aclanthology.org/I05-5002.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg challenge: Generating text from rdf data. In Proceedings of the 10th International Conference on Natural Language Generation, pp. 124–133, 2017.

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. When do neural networks outperform kernel methods? arXiv preprint arXiv:2006.13409, 2020.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. CoRR, abs/1911.12237, 2019. URL http://arxiv.org/abs/1911.12237.

Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. GAMM-Mitteilungen, 36(1):53–78, 2013.

Jihun Ham and Daniel D. Lee. Grassmann discriminant analysis: a unifying view on subspace-based learning. In ICML, pp. 376–383, 2008. URL https://doi.org/10.1145/1390156.1390204.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. WARP: Word-level Adversarial ReProgramming. arXiv:2101.00121 [cs], December 2020. URL http://arxiv.org/abs/2101.00121. arXiv: 2101.00121.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention, 2021.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. arXiv:1902.00751 [cs, stat], June 2019. URL http://arxiv.org/abs/1902.00751.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866, 2014.

Mikhail Khodak, Neil Tenenholtz, Lester Mackey, and Nicolò Fusi. Initialization and regularization of factorized neural layers, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020.

Brian Lester, Rami Al-Rfou, and Noah Constant. The Power of Scale for Parameter-Efficient Prompt Tuning. arXiv:2104.08691 [cs], April 2021. URL http://arxiv.org/abs/2104.08691. arXiv: 2104.08691.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the Intrinsic Dimension of Objective Landscapes. arXiv:1804.08838 [cs, stat], April 2018a. URL http://arxiv.org/abs/1804.08838. arXiv: 1804.08838.

Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. arXiv:2101.00190 [cs], January 2021. URL http://arxiv.org/abs/2101.00190.

Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In Advances in Neural Information Processing Systems, 2018.

Yuanzhi Li, Yingyu Liang, and Andrej Risteski. Recovery guarantee of weighted low-rank approximation via alternating minimization. In International Conference on Machine Learning, pp. 2358–2367. PMLR, 2016.

Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In Conference On Learning Theory, pp. 2–47. PMLR, 2018b.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. Exploring versatile generative language model via parameter-efficient transfer learning. In Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 441–459, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.41. URL https://aclanthology.org/2020.findings-emnlp.41.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. GPT Understands, Too. arXiv:2103.10385 [cs], March 2021. URL http://arxiv.org/abs/2103.10385. arXiv: 2103.10385.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers, 2021.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, et al. Dart: Open-domain structured data record to text generation. arXiv preprint arXiv:2007.02871, 2020.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The e2e dataset: New challenges for end-to-end generation. arXiv preprint arXiv:1706.09254, 2017.

Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. arXiv preprint arXiv:1906.05392, 2019.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning, 2021.

Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. Semi-orthogonal low-rank matrix factorization for deep neural networks. In Interspeech, pp. 3743–3747, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. pp. 12, a.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. pp. 24, b.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. CoRR, abs/1806.03822, 2018. URL http://arxiv.org/abs/1806.03822.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. arXiv:1705.08045 [cs, stat], November 2017. URL http://arxiv.org/abs/1705.08045. arXiv: 1705.08045.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. Adapterdrop: On the efficiency of adapters in transformers, 2020.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In 2013 IEEE international conference on acoustics, speech and signal processing, pp. 6655–6659. IEEE, 2013.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1170.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 6000–6010, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems, 2020.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. arXiv preprint arXiv:1805.12471, 2018.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pp. 1112–1122, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1101. URL https://www.aclweb.org/anthology/N18-1101.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Greg Yang and Edward J. Hu. Feature Learning in Infinite-Width Neural Networks. arXiv:2011.14522 [cond-mat], May 2021. URL http://arxiv.org/abs/2011.14522. arXiv: 2011.14522.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2021.

Yu Zhang, Ekapol Chuangsuwanich, and James Glass. Extracting deep neural network bottleneck features using low-rank matrix factorization. In 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp. 185–189. IEEE, 2014.

Yong Zhao, Jinyu Li, and Yifan Gong. Low-rank plus diagonal adaptation for deep neural networks. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5005–5009. IEEE, 2016.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. CoRR, abs/1709.00103, 2017. URL http://arxiv.org/abs/1709.00103.

A Large Language Models Still Need Parameter Updates

Few-shot learning, 또는 prompt engineering 방식은 학습 샘플이 매우 적을 때 상당한 이점을 제공한다. 그러나 실제로는, 성능이 중요한 응용 분야를 위해 수천 개 이상의 학습 예시를 마련하는 것이 종종 가능하다. Table 8에서 알 수 있듯이, 데이터셋의 크기가 대소에 관계없이, few-shot 학습 대비 fine-tuning이 모델 성능을 크게 개선한다. RTE에 대한 GPT-3 few-shot 결과는 GPT-3 논문 (Brown et al., 2020)에서 인용하였으며, MNLI-matched 태스크의 경우 클래스별로 시범(demonstration)을 두 개씩, 총 여섯 개의 in-context 예시를 사용하였다.

Method	MNLI-m (Val. Acc./%)	RTE (Val. Acc./%)
GPT-3 Few-Shot	40.6	69.0
GPT-3 Fine-Tuned	89.5	85.4

Table 8: GPT-3 (Brown et al., 2020)에서 fine-tuning이 few-shot 학습 대비 훨씬 우수한 성능을 보이는 결과를 나타낸다.

B Inference Latency Introduced by Adapter Layers

Adapter layer는 사전 학습된 모델에 순차적(sequential)으로 추가되는 외부 모듈인 반면, 본 논문에서 제안하는 LoRA는 병렬(parallel)로 추가되는 외부 모듈로 볼 수 있다. 따라서 adapter layer를 사용하는 경우, 기본(base) 모델 계산에 더해 adapter layer 계산도 추가로 수행되어야 하므로, 필연적으로 추론(inference) 지연(latency)이 발생한다. 물론 Rücklé et al. (2020)에서 지적한 바와 같이, 모델의 배치 크기(batch size)나 시퀀스 길이(sequence length)가 충분히 커 하드웨어 병렬성을 완전히 활용할 수 있다면, adapter layer가 유발하는 지연은 어느 정도 완화될 수 있다. 우리는 유사한 방식으로 GPT-2 medium에서 latency 실험을 진행하여 이들의 관찰을 재확인하였고, 동시에 온라인 추론 환경처럼 배치 크기가 작을 경우에는 추가 지연이 상당할 수 있음을 밝힌다.

우리는 NVIDIA Quadro RTX8000에서 단일 forward pass를 100회 반복하여 평균을 구함으로써 latency를 측정하였다. 입력 배치 크기, 시퀀스 길이, 그리고 adapter의 bottleneck 차원 r을 변경하면서 실험을 진행하였다. adapter 설계로는 Houlsby et al. (2019)의 오리지널 설계를 Adapter$^{\mathrm{H}}$, 그리고 최근 제안된 더 효율적인 변형 (Lin et al., 2020)을 Adapter$^{\mathrm{L}}$라 명명하여 비교하였다. 각 설계에 대한 자세한 사항은 Section 5.1를 참고하라. Figure 5에서는 adapter가 없는($r = 0$) 기준 대비, adapter 적용으로 인해 발생한 지연 증가율(

C Dataset Details

GLUE Benchmark 는 자연어 이해(NLU) 태스크를 폭넓게 아우르는 데이터셋 모음이다. MNLI (inference, Williams et al., 2018), SST-2(감정 분석, Socher et al., 2013), MRPC(의미적 유사 문장 판단, Dolan & Brockett, 2005), CoLA(언어적 타당성, Warstadt et al., 2018), QNLI (inference, Rajpurkar et al., 2018), QQP[8](질의응답), RTE(inference), STS-B(텍스트 유사

[8] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

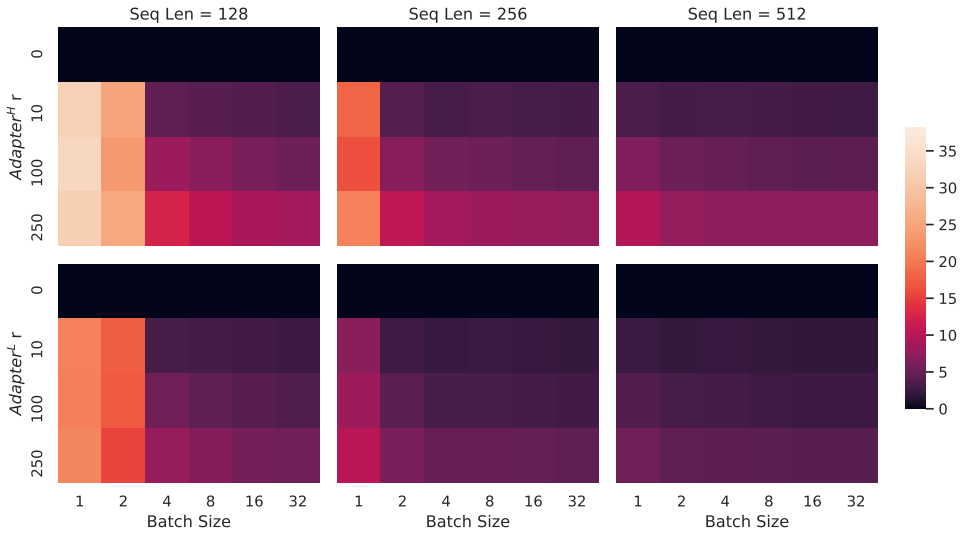

Figure 5: adapter를 사용하지 않은($r = 0$) 기준 대비 추론 latency가 얼마나 느려졌는지를 백분율로 나타낸 결과이다. 상단 행은 Adapter$^{\text{H}}$, 하단 행은 Adapter$^{\text{L}}$에 대한 결과를 보여준다. 배치 크기와 시퀀스 길이가 클수록 지연이 감소하지만, 온라인 환경에서 시퀀스 길이가 짧은 경우 지연이 30% 이상까지 높아질 수 있다. 시각적 구분을 위해 컬러맵을 조정하였다.

도, Cer et al., 2017) 등을 포함한다. 이처럼 광범위한 커버리지를 갖추고 있어, GLUE 벤치마크는 RoBERTa나 DeBERTa와 같은 NLU 모델을 평가하는 데 표준으로 사용된다. 개별 데이터셋은 각각 서로 다른 오픈 라이선스 형태로 공개되어 있다.

WikiSQL 은 Zhong et al. (2017)에서 제안되었으며, 총 $56,355/8,421$의 학습/검증 예시를 포함한다. 자연어 질의와 테이블 스키마(table schema)를 입력으로 받아, 이에 해당하는 SQL 쿼리를 생성하는 과제이다. 본 실험에서는 입력(context)을 {table schema, query}로, 출력을 $y = \{\text{SQL}\}$로 인코딩하였다. 이 데이터셋은 BSD 3-Clause 라이선스로 공개되어 있다.

SAMSum 은 Gliwa et al. (2019)에서 소개되었으며, $14,732/819$의 학습/테스트 예시가 포함되어 있다. 두 사람 간의 대화(conversation)와, 전문 언어학자가 작성한 추상적 요약(linguist-generated abstractive summaries)으로 구성된다. 우리 실험에서는 말풍선(utterance)을 "\n"으로 연결하고, 마지막에 "\n\n"을 추가하여 입력(context)을 구성하였으며, 출력은 $y = \{\text{summary}\}$이다. 이 데이터셋은 비상업용 라이선스인 Creative Commons BY-NC-ND 4.0 하에 공개되어 있다.

E2E NLG Challenge 는 Novikova et al. (2017)에서 처음 제안된 데이터셋으로, 데이터 기반(end-to-end) 자연어 생성 시스템 학습을 위해 널리 사용된다. 식당 도메인에서 추출된 약 4.2만 건의 학습 예시, 4,600건의 검증 예시, 4,600건의 테스트 예시로 구성되어 있다. 각 입력 테이블은 다수의 참조 문장을 가질 수 있으며, 입력 (x, y)는 슬롯-값(slot-value) 쌍의 시퀀스와 이와 대응되는 자연어 문장으로 이루어져 있다. 이 데이터셋은 Creative Commons BY-NC-SA 4.0 라이선스로 공개되어 있다.

DART 는 Nan et al. (2020)에서 제시된 오픈 도메인 데이터-투-텍스트(data-to-text) 데이터셋이다. DART의 입력은 ENTITY | RELATION | ENTITY 형태의 삼중항(triple) 시퀀스로 구조화된다. 총 약 8.2만 건의 예시를 포함하고 있어, E2E에 비해 훨씬 대규모이며 복잡한 데이터-투-텍스트 태스크이다. 이 데이터셋은 MIT 라이선스로 공개되어 있다.

WebNLG 는 또 다른 데이터-투-텍스트 평가용 데이터셋으로, Gardent et al. (2017)에서 소개되었다. 총 약 2.2만 건의 예시가 14개의 다른 카테고리로 구성되는데, 이 가운데 9개는 학습 중에 등장하며, 나머지 5개 카테고리는 테스트에만 등장한다. 따라서 평가 시 보통 학습 중에 등장했던 카테고리(S), 등장하지 않았던 카테고리(U), 그리고 전체(A)로 구분하여 결과를 제시한다. 각 입력 예시는 SUBJECT | PROPERTY | OBJECT 형태의 삼중항 시퀀스로 표현된다. 이 데이터셋은 Creative Commons BY-NC-SA 4.0 라이선스로 공개되어 있다.

D Hyperparameters Used in Experiments

D.1 RoBERTa

본 연구에서는 AdamW 옵티마이저와 선형 학습률(learning rate) 감소 스케줄(linear decay)을 사용하여 학습을 진행하였다. LoRA에 대해서는 학습률, 학습 에폭(epoch) 수, 배치 크기를 탐색하였다. Liu et al. (2019)의 방법을 참조하여, MRPC, RTE, STS-B를 학습할 때는 일반적인 초기화 대신, MNLI에서 최적화된 체크포인트를 사용하여 LoRA 모듈을 초기화하였다. 단, 사전 학습된 모델 자체는 모든 태스크에서 동결(frozen) 상태를 유지한다. 최종 보고된 성능은 무작위 시드 5개에 대한 결과의 중앙값이며, 각 실험(run)에서는 최상의 에폭 결과를 사용하였다.

Houlsby et al. (2019) 및 Pfeiffer et al. (2021)와의 공정한 비교를 위해, 본 실험에서는 모델 시퀀스 길이를 128로 제한하고, 모든 태스크에서 동일한 배치 크기를 사용하였다. 중요한 점은, MRPC, RTE, STS-B에 적응시키는 경우, MNLI에 이미 적응된 모델이 아닌, 사전 학습된 RoBERTa large 모델을 시작점으로 사용하였다는 것이다. 이와 같이 제약된 설정을 따른 실험 결과에는 † 표기를 부여하였다. 본 연구에서 사용한 하이퍼파라미터는 Table 9를 참고하라.

D.2 DeBERTa

We again train using AdamW with a linear learning rate decay schedule. Following He et al. (2021), we tune learning rate, dropout probability, warm-up steps, and batch size. We use the same model sequence length used by (He et al., 2021) to keep our comparison fair. Following He et al. (2021), we initialize the LoRA modules to our best MNLI checkpoint when adapting to MRPC, RTE, and STS-B, instead of the usual initialization; the pre-trained model stays frozen for all tasks. We report the median over 5 random seeds; the result for each run is taken from the best epoch. See the hyperparameters used in our runs in Table 10.

D.3 GPT-2

본 연구에서는 모든 GPT-2 모델을 AdamW (Loshchilov & Hutter, 2017) 옵티마이저와 선형 학습률 스케줄을 사용하여 총 5에폭(epoch) 동안 학습하였다. 배치 크기, 학습률, 그리고 빔 검색(beam search)에서의 빔 크기는 Li & Liang (2021)에 기술된 값을 그대로 사용하였으며, 이에 맞추어 LoRA의 하이퍼파라미터도 함께 탐색하였다. 결과는 무작위 시드 3개에 대해 측정된 평균값을 보고하며, 각 실험(run)에서는 최적의 에폭에서의 성능을□□한다. LoRA를 GPT-2에 적용할 때 사용한 하이퍼파라미터는 Table 11에 나열하였다. 다른 베이스라인 기법에 대한 하이퍼파라미터 설정은 Li & Liang (2021)를 참조하라.

D.4 GPT-3

GPT-3 관련 모든 실험에서, 우리는 AdamW (Loshchilov & Hutter, 2017) 옵티마이저를 사용하여 배치 크기 128, 가중치 감쇠(weight decay) 계수 0.1로 총 2에폭(epoch)을 학습하였다. 시퀀스 길이는 WikiSQL (Zhong et al., 2017)에서 384, MNLI (Williams et al., 2018)에서 768, SAMSum (Gliwa et al., 2019)에서는 2048로 설정하였다. 모든 기법과 데이터셋 조합에 대해 학습률(learning rate)을 탐색하였으며, 추가 하이퍼파라미터 관련 세부 사항은 Section D.4를 참고하라.

Prefix-embedding tuning의 경우, $l_p = 256$과 $l_i = 8$이 최적값으로 나타났으며, 이때 학습되는 파라미터 수는 총 약 320만 개($3.2M$)이다. Prefix-layer tuning은 $l_p = 8$, $l_i = 8$을 사용하여 약 2,020만 개($20.2M$)의 파라미터를 학습하였고, 전체적으로 가장 우수한 성능을 보이는 구성을 확보하였다.

LoRA는 두 가지 파라미터 예산을 제시하는데, 470만 개($4.7M$)와 3,770만 개($37.7M$)이다. 전자의 경우 $r_q = r_v = 1$이거나 $r_v = 2$ 등으로 설정하며, 후자의 경우 $r_q = r_v = 8$ 또는 $r_q = r_k = r_v = r_o = 2$와 같이 설정한다. 각 실험(run)에서의 최적 검증 성능을 보고하였으며, GPT-3 실험에 사용된 학습 하이퍼파라미터는 Table 12에 요약되어 있다.

Method	Dataset	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
	Optimizer	AdamW							
	Warmup Ratio	0.06							
	LR Schedule	Linear							
RoBERTa base LoRA	Batch Size	16	16	16	32	32	16	32	16
	# Epochs	30	60	30	80	25	25	80	40
	Learning Rate	5E-04	5E-04	4E-04	4E-04	4E-04	5E-04	5E-04	4E-04
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	8							
	Max Seq. Len.	512							
RoBERTa large LoRA	Batch Size	4	4	4	4	4	4	8	8
	# Epochs	10	10	20	20	10	20	20	30
	Learning Rate	3E-04	4E-04	3E-04	2E-04	2E-04	3E-04	4E-04	2E-04
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	16							
	Max Seq. Len.	128	128	512	128	512	512	512	512
RoBERTa large LoRA†	Batch Size	4							
	# Epochs	10	10	20	20	10	20	20	10
	Learning Rate	3E-04	4E-04	3E-04	2E-04	2E-04	3E-04	4E-04	2E-04
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	16							
	Max Seq. Len.	128							
RoBERTa large Adpt^{P} (3M)†	Batch Size	32							
	# Epochs	10	20	20	20	10	20	20	20
	Learning Rate	3E-05	3E-05	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	Bottleneck r	64							
	Max Seq. Len.	128							
RoBERTa large Adpt^{P} (0.8M)†	Batch Size	32							
	# Epochs	5	20	20	20	10	20	20	20
	Learning Rate	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	Bottleneck r	16							
	Max Seq. Len.	128							
RoBERTa large Adpt^{H} (6M)†	Batch Size	32							
	# Epochs	10	5	10	10	5	20	20	10
	Learning Rate	3E-05	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	Bottleneck r	64							
	Max Seq. Len.	128							
RoBERTa large Adpt^{H} (0.8M)†	Batch Size	32							
	# Epochs	10	5	10	10	5	20	20	10
	Learning Rate	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	Bottleneck r	8							
	Max Seq. Len.	128							

Table 9: GLUE 벤치마크에서 RoBERTa를 학습하기 위해 사용한 하이퍼파라미터를 정리하였다.

E Combining LoRA with Prefix Tuning

LoRA는 기존 prefix 기반 기법과 자연스럽게 결합할 수 있다. 본 절에서는 LoRA와 prefix-tuning 변형 기법들을 결합하여 WikiSQL과 MNLI에서 성능을 평가한다.

LoRA+PrefixEmbed (LoRA+PE)는 LoRA에 prefix-embedding tuning을 결합한 방식으로, $l_p + l_i$개의 특수 토큰을 삽입하고, 이 토큰들의 임베딩을 학습 파라미터로 취급한다. prefix-embedding tuning에 대한 자세한 내용은 Section 5.1를 참고하라.

LoRA+PrefixLayer (LoRA+PL)는 LoRA에 prefix-layer tuning을 결합한 방식이다. 이 역시 $l_p + l_i$개의 특수 토큰을 삽입하지만, 해당 토큰의 hidden representation이 자연스럽게 진화하도록 두는 대신, 매번 Transformer 블록을 통과할 때마다 입력과 무관한 벡터로 이를 교체한다.

Method	Dataset	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
	Optimizer	AdamW							
	Warmup Ratio	0.1							
	LR Schedule	Linear							
DeBERTa XXL LoRA	Batch Size	8	8	32	4	6	8	4	4
	# Epochs	5	16	30	10	8	11	11	10
	Learning Rate	1E-04	6E-05	2E-04	1E-04	1E-04	1E-04	2E-04	2E-04
	Weight Decay	0	0.01	0.01	0	0.01	0.01	0.01	0.1
	CLS Dropout	0.15	0	0	0.1	0.1	0.2	0.2	0.2
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	8							
	Max Seq. Len.	256	128	128	64	512	320	320	128

Table 10: DeBERTa XXL 모델을 GLUE 벤치마크 태스크들에 적용할 때 사용한 하이퍼파라미터를 요약하였다.

Dataset	E2E	WebNLG	DART
	Training		
Optimizer	AdamW		
Weight Decay	0.01	0.01	0.0
Dropout Prob	0.1	0.1	0.0
Batch Size	8		
# Epoch	5		
Warmup Steps	500		
Learning Rate Schedule	Linear		
Label Smooth	0.1	0.1	0.0
Learning Rate	0.0002		
Adaptation	$r_q = r_v = 4$		
LoRA α	32		
	Inference		
Beam Size	10		
Length Penalty	0.9	0.8	0.8
no repeat ngram size	4		

Table 11: E2E, WebNLG, DART 데이터셋에서 GPT-2 LoRA를 적용할 때 사용한 하이퍼파라미터를 요약하였다.

결과적으로, 임베딩뿐 아니라 이후 Transformer 블록의 activation도 학습 파라미터가 된다. prefix-layer tuning에 대한 자세한 내용은 Section 5.1를 참고하라.

Table 15에서 WikiSQL과 MultiNLI에 대한 LoRA+PE, LoRA+PL의 평가 결과를 제시하였다. 먼저, LoRA+PE는 WikiSQL에서 LoRA 단독 및 prefix-embedding tuning 단독보다 유의미하게 우수한 성능을 보이는데, 이는 LoRA가 prefix-embedding tuning과 어느 정도 상호보완적임을 시사한다. 반면 MultiNLI의 경우, LoRA+PE 조합이 LoRA 단독보다 나은 성능을 내지 못하는데, 이는 이미 LoRA만으로도 인간 수준에 근접하는 성능을 달성하고 있기 때문일 것으로 추정된다. 또한 LoRA+PL은 더 많은 학습 파라미터를 활용함에도 불구하고 LoRA 단독보다 다소 낮은 성능을 보였다. 이는 prefix-layer tuning이 학습률(learning rate)의 설정에 매우 민감하며, 이로 인해 LoRA 가중치의 최적화가 LoRA+PL 내에서 더 복잡해진 데 기인한 것으로 추정된다.

F Combining LoRA with Prefix Tuning

LoRA는 기존 prefix 기반 기법과 자연스럽게 결합할 수 있다. 본 절에서는 LoRA와 prefix-tuning 변형 기법들을 결합하여 WikiSQL과 MNLI에서 성능을 평가한다.

Hyperparameters	Fine-Tune	PreEmbed	PreLayer	BitFit	Adapter^{H}	LoRA
Optimizer	AdamW					
Batch Size	128					
# Epoch	2					
Warmup Tokens	250,000					
LR Schedule	Linear					
Learning Rate	5.00E-06	5.00E-04	1.00E-04	1.6E-03	1.00E-04	2.00E-04

Table 12: GPT-3 모델의 다양한 적응 기법에 대해 사용한 학습 하이퍼파라미터를 요약하였다. 학습률(learning rate)을 탐색한 뒤, 나머지 하이퍼파라미터는 모든 데이터셋에서 동일하게 적용하였다.

LoRA+PrefixEmbed (LoRA+PE)는 LoRA에 prefix-embedding tuning을 결합한 방식으로, $l_p + l_i$개의 특수 토큰을 삽입하고, 이 토큰들의 임베딩을 학습 파라미터로 취급한다. prefix-embedding tuning에 대한 자세한 내용은 Section 5.1를 참고하라.

LoRA+PrefixLayer (LoRA+PL)는 LoRA에 prefix-layer tuning을 결합한 방식이다. 이 역시 $l_p + l_i$개의 특수 토큰을 삽입하지만, 해당 토큰의 hidden representation이 자연스럽게 진화하도록 두는 대신, 매번 Transformer 블록을 통과할 때마다 입력과 무관한 벡터로 이를 교체한다. 결과적으로, 임베딩뿐 아니라 이후 Transformer 블록의 activation도 학습 파라미터가 된다. prefix-layer tuning에 대한 자세한 내용은 Section 5.1를 참고하라.

Table 15에서 WikiSQL과 MultiNLI에 대한 LoRA+PE, LoRA+PL의 평가 결과를 제시하였다. 먼저, LoRA+PE는 WikiSQL에서 LoRA 단독 및 prefix-embedding tuning 단독보다 유의미하게 우수한 성능을 보이는데, 이는 LoRA가 prefix-embedding tuning과 어느 정도 상호보완적임을 시사한다. 반면 MultiNLI의 경우, LoRA+PE 조합이 LoRA 단독보다 나은 성능을 내지 못하는데, 이는 이미 LoRA만으로도 인간 수준에 근접하는 성능을 달성하고 있기 때문일 것으로 추정된다. 또한 LoRA+PL은 더 많은 학습 파라미터를 활용함에도 불구하고 LoRA 단독보다 다소 낮은 성능을 보였다. 이는 prefix-layer tuning이 학습률(learning rate)의 설정에 매우 민감하며, 이로 인해 LoRA 가중치의 최적화가 LoRA+PL 내에서 더 복잡해진 데 기인한 것으로 추정된다.

Method	# Trainable Parameters	DART BLEU↑	DART MET↑	DART TER↓
GPT-2 Medium				
Fine-Tune	354M	46.2	**0.39**	**0.46**
Adapter^{L}	0.37M	42.4	0.36	0.48
Adapter^{L}	11M	45.2	0.38	**0.46**
FT^{Top2}	24M	41.0	0.34	0.56
PrefLayer	0.35M	46.4	0.38	**0.46**
LoRA	0.35M	**47.1**$_{\pm.2}$	**0.39**	**0.46**
GPT-2 Large				
Fine-Tune	774M	47.0	**0.39**	0.46
Adapter^{L}	0.88M	45.7$_{\pm.1}$	0.38	0.46
Adapter^{L}	23M	47.1$_{\pm.1}$	**0.39**	**0.45**
PrefLayer	0.77M	46.7	0.38	**0.45**
LoRA	0.77M	**47.5**$_{\pm.1}$	**0.39**	**0.45**

Table 13: DART 데이터셋에서 다양한 적응 기법을 적용한 GPT-2 모델의 결과이다. MET와 TER 지표에 대한 분산은 모든 적응 기법에서 0.01 미만으로 관측되었다.

Method	WebNLG								
	BLEU↑			MET↑			TER↓		
	U	S	A	U	S	A	U	S	A
	GPT-2 Medium								
Fine-Tune (354M)	27.7	**64.2**	46.5	.30	**.45**	.38	.76	**.33**	.53
Adapter^{L} (0.37M)	45.1	54.5	50.2	.36	.39	.38	.46	.40	.43
Adapter^{L} (11M)	**48.3**	60.4	54.9	**.38**	.43	**.41**	**.45**	.35	**.39**
FT^{Top2} (24M)	18.9	53.6	36.0	.23	.38	.31	.99	.49	.72
Prefix (0.35M)	45.6	62.9	55.1	**.38**	.44	**.41**	.49	.35	.40
LoRA (0.35M)	46.7$_{\pm.4}$	62.1$_{\pm.2}$	**55.3**$_{\pm.2}$	**.38**	.44	**.41**	.46	**.33**	**.39**
	GPT-2 Large								
Fine-Tune (774M)	43.1	65.3	55.5	.38	**.46**	.42	.53	.33	.42
Adapter^{L} (0.88M)	**49.8**$_{\pm.0}$	61.1$_{\pm.0}$	56.0$_{\pm.0}$	.38	.43	.41	**.44**	.35	.39
Adapter^{L} (23M)	49.2$_{\pm.1}$	64.7$_{\pm.2}$	**57.7**$_{\pm.1}$	**.39**	**.46**	**.43**	.46	.33	.39
Prefix (0.77M)	47.7	63.4	56.3	**.39**	.45	.42	.48	.34	.40
LoRA (0.77M)	48.4$_{\pm.3}$	64.0$_{\pm.3}$	57.0$_{\pm.1}$	**.39**	.45	.42	.45	**.32**	**.38**

Table 14: WebNLG 데이터셋에서 다양한 적응 기법을 적용한 GPT-2 모델의 결과이다. 모든 실험에서 MET와 TER 지표의 분산은 0.01 미만으로 관측되었다. "U"는 학습에서 등장하지 않은 카테고리(unseen), "S"는 학습 중 등장한 카테고리(seen), "A"는 테스트셋에 포함된 모든 카테고리를 의미한다.

F.1 Additional Experiments on GPT-3

Table 15는 GPT-3 모델에 대해 다양한 적응 기법을 적용한 추가 실험 결과를 보여준다. 본 실험에서는 성능과 학습해야 하는 파라미터 수 사이에 존재하는 트레이드오프를 중점적으로 살펴보았다.

F.2 Low-Data Regime

적은 양의 데이터가 주어졌을 때(이하, low-data regime)의 적응 기법 성능을 평가하기 위해, 우리는 MNLI 전체 학습 세트에서 무작위로 100개, 1,000개, 10,000개의 학습 예시를 각각 샘플링하여 MNLI-n 태스크를 구성하였다. Table 16에서는 MNLI-n에서의 다양한 적응 기법 성능을 제시한다. 흥미롭게도, MNLI-100 데이터셋에서 PrefixEmbed와 PrefixLayer 기법은 매우 낮은 성능을 보이는데, 특히 PrefixEmbed의 정확도는 무작위 추정(33.3%)보다 약간 높은 37.6%에 그쳤다. PrefixLayer는 PrefixEmbed보다 성능이 다소 높긴 하지만, MNLI-100에서 Fine-Tune이나 LoRA와 비교하면 여전히 크게 뒤처진다. 학습 예시 수가 증가함에 따라 prefix 기반 기법들과 LoRA/Fine-tuning 간의 성능 차이가 줄어들기는 하지만, 이는 GPT-3 환경에서 low-data 태스크에는 prefix 기반 접근이 부적합할 수 있음을 시사한다. LoRA는 MNLI-100과 MNLI-Full에서 Fine-Tune보다 나은 성능을 보였으며, 무작위 시드로 인한 편차(± 0.3)를 감안했을 때 MNLI-1k, MNLI-10k에서도 유사한 결과를 얻었다.

MNLI-n에서 서로 다른 적응 기법을 학습하기 위해 사용한 하이퍼파라미터는 Table 17에 요약되어 있다. MNLI-100 데이터셋의 PrefixLayer 학습 시, 더 큰 학습률을 사용하면 학습 손실이 줄어들지 않아, 작은 학습률을 사용하였다.

G Measuring Similarity Between Subspaces

본 논문에서는 두 개의 열 벡터가 직교 정규(orthonormal)인 행렬 $U_A^i \in \mathbb{R}^{d\times i}$와 $U_B^j \in \mathbb{R}^{d\times j}$ 간 서브스페이스 유사도를 측정하기 위해,

$$\phi(A, B, i, j) = \psi(U_A^i, U_B^j) = \frac{\|U_A^{i\top} U_B^j\|_F^2}{\min\{i, j\}}$$

를 사용한다. 여기서 U_A^i, U_B^j는 각각 행렬 A, B의 왼쪽 특이벡터 행렬(특이값 분해 시 얻는 유니터리 행렬)에서 상위 i 또는 j개의 열을 추출한 것이다. 이 유사도 지표는 서브스페이스

Method	Hyperparameters	# Trainable Parameters	WikiSQL	MNLI-m
Fine-Tune	-	175B	73.8	89.5
PrefixEmbed	$l_p = 32, l_i = 8$	0.4 M	55.9	84.9
	$l_p = 64, l_i = 8$	0.9 M	58.7	88.1
	$l_p = 128, l_i = 8$	1.7 M	60.6	88.0
	$l_p = 256, l_i = 8$	3.2 M	63.1	88.6
	$l_p = 512, l_i = 8$	6.4 M	55.9	85.8
PrefixLayer	$l_p = 2, l_i = 2$	5.1 M	68.5	89.2
	$l_p = 8, l_i = 0$	10.1 M	69.8	88.2
	$l_p = 8, l_i = 8$	20.2 M	70.1	89.5
	$l_p = 32, l_i = 4$	44.1 M	66.4	89.6
	$l_p = 64, l_i = 0$	76.1 M	64.9	87.9
$\text{Adapter}^{\text{H}}$	$r = 1$	7.1 M	71.9	89.8
	$r = 4$	21.2 M	73.2	91.0
	$r = 8$	40.1 M	73.2	91.5
	$r = 16$	77.9 M	73.2	91.5
	$r = 64$	304.4 M	72.6	91.5
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
LoRA+PL	$r_q = r_v = 8, l_p = 8, l_i = 4$	52.8 M	72.9	90.2

Table 15: WikiSQL과 MNLI에서 다양한 적응 기법을 대상으로 하이퍼파라미터를 분석한 결과이다. trainable parameter 수가 증가함에 따라 prefix-embedding tuning (PrefixEmbed)과 prefix-layer tuning (PrefixLayer)은 성능이 저하되는 반면, LoRA는 성능이 안정적으로 유지된다. 모든 결과는 validation accuracy를 기준으로 평가하였다.

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**

Table 16: GPT-3 175B를 이용해 MNLI 데이터셋의 부분집합(MNLI-n)에서 다양한 기법의 검증 정확도를 비교하였다. MNLI-n은 n개의 학습 예시를 포함하는 부분집합을 의미하며, 검증은 전체 검증 세트로 수행하였다. LoRA는 Fine-tuning을 비롯한 다른 기법 대비 높은 샘플 효율성을 보인다.

사이의 거리를 측정하는 표준 사영 거리(Projection Metric)의 역(reverse)에 해당한다 Ham & Lee (2008).

Hyperparameters	Adaptation	MNLI-100	MNLI-1k	MNLI-10K	MNLI-392K
Optimizer	-	AdamW			
Warmup Tokens	-	250,000			
LR Schedule	-	Linear			
Batch Size	-	20	20	100	128
# Epoch	-	40	40	4	2
Learning Rate	FineTune	5.00E-6			
	PrefixEmbed	2.00E-04	2.00E-04	4.00E-04	5.00E-04
	PrefixLayer	5.00E-05	5.00E-05	5.00E-05	1.00E-04
	LoRA	2.00E-4			
Adaptation-Specific	PrefixEmbed l_p	16	32	64	256
	PrefixEmbed l_i	8			
	PrefixTune	$l_p = l_i = 8$			
	LoRA	$r_q = r_v = 8$			

Table 17: MNLI(m)-n에서 GPT-3 모델에 다양한 적응 기법을 적용하기 위해 사용한 하이퍼파라미터를 정리하였다.

구체적으로, $U_A^{i\top} U_B^j$의 특이값을 $\sigma_1, \sigma_2, \cdots, \sigma_p$라 할 때, $p = \min\{i, j\}$이다. 표준 사영 거리(Projection Metric)는 다음과 같이 정의된다 Ham & Lee (2008).

$$d(U_A^i, U_B^j) = \sqrt{p - \sum_{k=1}^{p} \sigma_k^2} \quad \in [0, \sqrt{p}],$$

반면 우리 논문에서의 유사도는

$$\phi(A, B, i, j) = \psi(U_A^i, U_B^j) = \frac{\sum_{k=1}^{p} \sigma_k^2}{p} = \frac{1}{p}\Big(1 - d(U_A^i, U_B^j)^2\Big)$$

로 정의된다.

이 유사도 지표는 U_A^i와 U_B^j가 동일한 열(column) 공간을 공유하는 경우 $\phi(A, B, i, j) = 1$이 되며, 두 서브스페이스가 완전히 직교하면 $\phi(A, B, i, j) = 0$이 된다. 그 외의 경우에는 $0 < \phi(A, B, i, j) < 1$ 범위를 갖는다.

H Additional Experiments on Low-Rank Matrices

이 절에서는 저랭크 업데이트 행렬에 대한 추가 실험 결과를 제시한다.

H.1 Correlation between LoRA Modules

Figure 6와 Figure 7는, 본문 Figure 3와 Figure 4에서 제시한 결과가 다른 레이어에도 일관되게 확장됨을 보여준다.

H.2 Effect of r on GPT-2

우리는 Section 7.2에서 다룬 r의 효과에 대한 실험을 GPT-2 모델에서도 재현하였다. E2E NLG Challenge 데이터셋을 예시로 들어, 26,000 스텝 학습 후 서로 다른 r 값에서 달성한 검증 로스(validation loss)와 테스트 지표를 Table 18에 제시한다. GPT-2 Medium 모델에서는 사용하는 지표에 따라 최적의 rank가 4에서 16 사이로 나타났으며, 이는 GPT-3 175B와 유사한 경향을 보인다. 단, 모델 크기와 적응 시점에서의 최적 rank 간의 구체적인 상관관계는 여전히 열려 있는 연구 문제다.

H.3 Correlation between W and ΔW

Figure 8는 서로 다른 r 값에서 W와 ΔW 간 정규화된 서브스페이스 유사도를 보여준다.

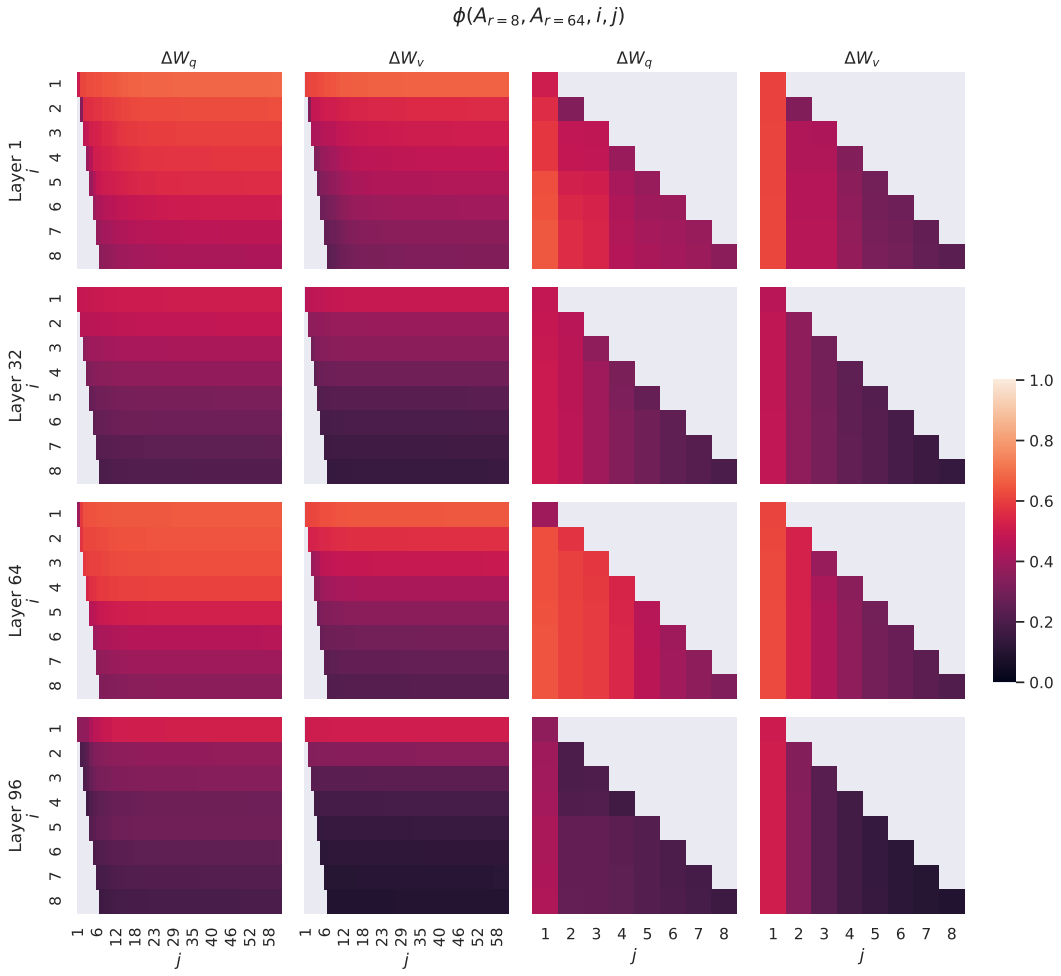

Figure 6: 96레이어 Transformer에서 1번째, 32번째, 64번째, 96번째 레이어의 ΔW_q와 ΔW_v에 대해, $A_{r=8}$과 $A_{r=64}$의 열 벡터 간 정규화된 서브스페이스 유사도를 비교한 결과이다.

주목할 점은, ΔW가 W의 상위 특이벡터 방향을 거의 포함하지 않는다는 것이다. 즉, ΔW의 상위 4개 특이벡터 방향과 W의 상위 10% 특이벡터 간 유사도가 0.2를 간신히 넘는 수준에 불과하다. 이는 ΔW가 기존의 W에서 강조되지 않은 특정 태스크 관련 "중요 방향"들을 학습한다는 근거로 볼 수 있다.

여기서 이어지는 흥미로운 질문은, 모델이 성공적으로 적응(adaptation)하기 위해서는 이와 같은 태스크 특화 방향들을 얼마나 "강하게" 증폭(amplify)해야 하는가 하는 것이다.

H.4 Amplification Factor

우리는 자연스럽게 특징 증폭 계수(feature amplification factor)를

$$\frac{\|\Delta W\|_F}{\|U^\top W V^\top\|_F}$$

로 정의할 수 있다. 여기서 U, V는 ΔW에 대한 특이값 분해(SVD)에서 얻은 왼쪽/오른쪽 특이벡터 행렬이다. (즉, $UU^\top WV^\top V$는 W를 ΔW가 생성하는 서브스페이스에 사영(projection)한 값에 해당한다.)

직관적으로, ΔW가 주로 태스크 특화(task-specific) 방향을 담고 있다면, 이 지표는 ΔW가 그러한 방향들을 얼마나 크게 증폭하는지를 측정한다. Section 7.3에서 보았듯, $r = 4$인 경우 이 증폭 계수는 최대 20에 달한다. 즉, 사전 학습 모델 W의 전체 특징 공간에서, 각 레이어마다 대체로 4개의 특징 방향이 존재하며, 다운스트림 태스크에서 보고한 정확도를 달성하기 위해

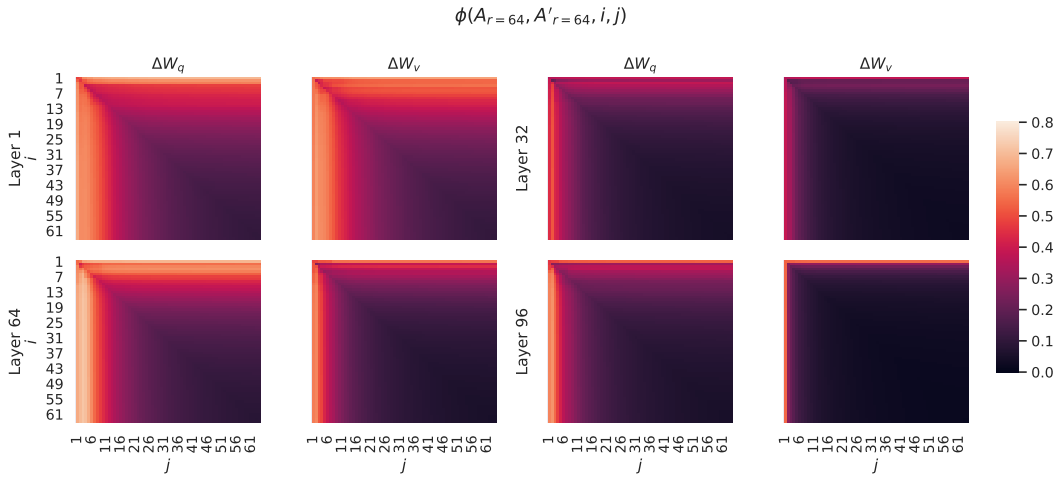

Figure 7: 96레이어 Transformer에서 1번째, 32번째, 64번째, 96번째 레이어의 ΔW_q와 ΔW_v에 대해, 무작위 시드가 다른 두 번의 학습에서 얻은 $A_{r=64}$ 열 벡터 간 정규화된 서브스페이스 유사도를 비교한 결과이다.

Rank r	val_loss	BLEU	NIST	METEOR	ROUGE_L	CIDEr
1	1.23	68.72	8.7215	0.4565	0.7052	2.4329
2	1.21	69.17	8.7413	0.4590	0.7052	2.4639
4	1.18	**70.38**	**8.8439**	**0.4689**	0.7186	**2.5349**
8	1.17	69.57	8.7457	0.4636	**0.7196**	2.5196
16	**1.16**	69.61	8.7483	0.4629	0.7177	2.4985
32	**1.16**	69.33	8.7736	0.4642	0.7105	2.5255
64	**1.16**	69.24	8.7174	0.4651	0.7180	2.5070
128	**1.16**	68.73	8.6718	0.4628	0.7127	2.5030
256	**1.16**	68.92	8.6982	0.4629	0.7128	2.5012
512	**1.16**	68.78	8.6857	0.4637	0.7128	2.5025
1024	1.17	69.37	8.7495	0.4659	0.7149	2.5090

Table 18: GPT-2 Medium 모델에 다양한 rank r 값을 적용하여 E2E NLG Challenge에서 LoRA를 학습했을 때의 검증 로스와 테스트 지표 결과이다. GPT-3에서는 $r = 1$로도 많은 태스크에서 충분한 성능을 보였지만, 여기서는 검증 로스 기준으로 $r = 16$, BLEU 기준으로 $r = 4$에서 최고 성능을 보인다. 이는 GPT-2 Medium의 적응에도 GPT-3 175B와 유사한 내재적 랭크(intrinsic rank)가 작용함을 시사한다. 다만, 일부 하이퍼파라미터가 $r = 4$ 값에 맞춰 튜닝되었음을 유의해야 하며, 이는 다른 r 값에서 최적이 아닐 수 있다.

서는 이들이 약 20배 정도 증폭되어야 한다는 의미이다. 그리고 다른 다운스트림 태스크별로는 전혀 다른 특징 방향들이 증폭될 것으로 예상된다.

그런데 $r = 64$인 경우, 이 증폭 계수는 약 2에 불과하다. 이는 즉, $r = 64$로 학습된 ΔW 안에 존재하는 대부분의 방향들은 크게 증폭되지 않음을 의미한다. 이는 전혀 놀라운 결과가 아니며, 사실상 "태스크 특화 방향(task-specific directions)"을 표현하기 위해, 즉 모델을 적응시키기 위해 필요한 내재적 랭크가 실제로 매우 낮다는 또 다른 근거라고 할 수 있다. 반면, $r = 4$ 버전의 ΔW에 포함된 방향들은 훨씬 더 큰 계수(약 20)로 증폭되는 셈이다.

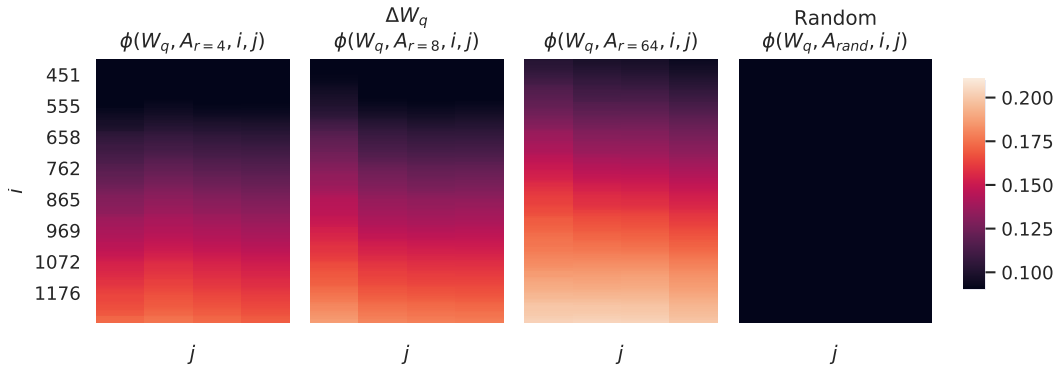

Figure 8: GPT-3에서 r 값을 달리하여 학습한 ΔW_q의 특이벡터 방향과, 원래 가중치 W_q의 특이벡터 방향을 비교한 정규화된 서브스페이스 유사도. 무작위 기준(random baseline)도 함께 나타내었다. ΔW는 W에서 강조되지 않은 "중요 방향"을 증폭하며, r가 커질수록 W가 이미 강조하던 방향들을 더 많이 포착하게 된다.