GEN AI 인텐시브 과정

강사 장철원

Section 0

코스소개

DAY1	DAY2	DAY3	DAY4	DAY5	DAY6	DAY7	DAY8
LLM Basic Concept	Transformers paper review	Transformers LangChain LangGraph			LLM service develop	Final Project	

Section 0

목차

❑ Full Fine Turning

❑ Adapter

❑ Adapter Fusion

❑ LoRA

GEN AI 인텐시브 과정

Section 1. Full Fine Tuning

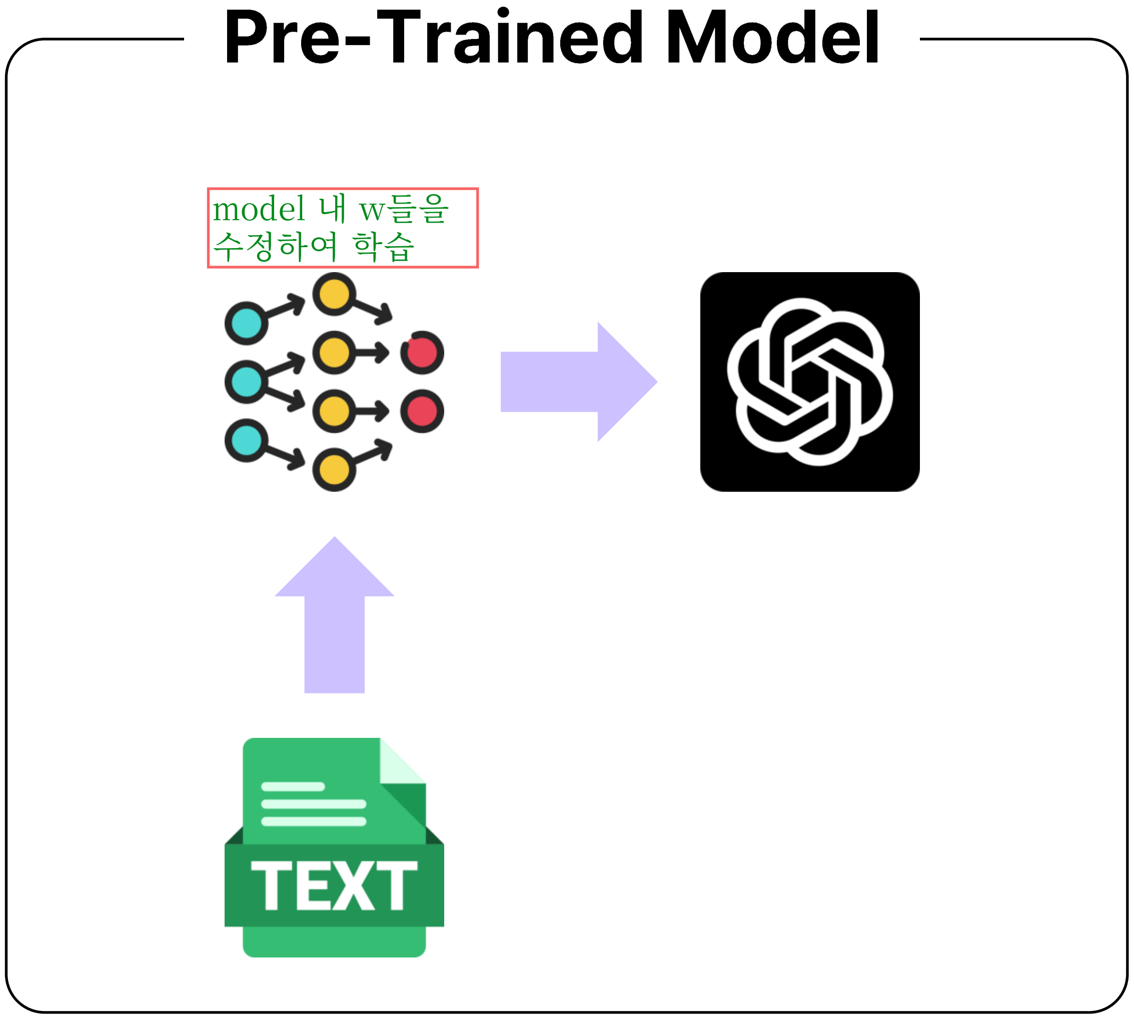
Section 1-1. Full Fine Tuning의 개념

Full Fine Tuning의 개념

- 파인 튜닝(Fine Tuning)
  - 사전 학습이 끝난 모델에 대해 또 다른 Task에 대해 데이터를 추가시켜 재학습 하는 과정
  - LLM을 구성하는 파라미터의 개수: 수억 ~ 수십억개

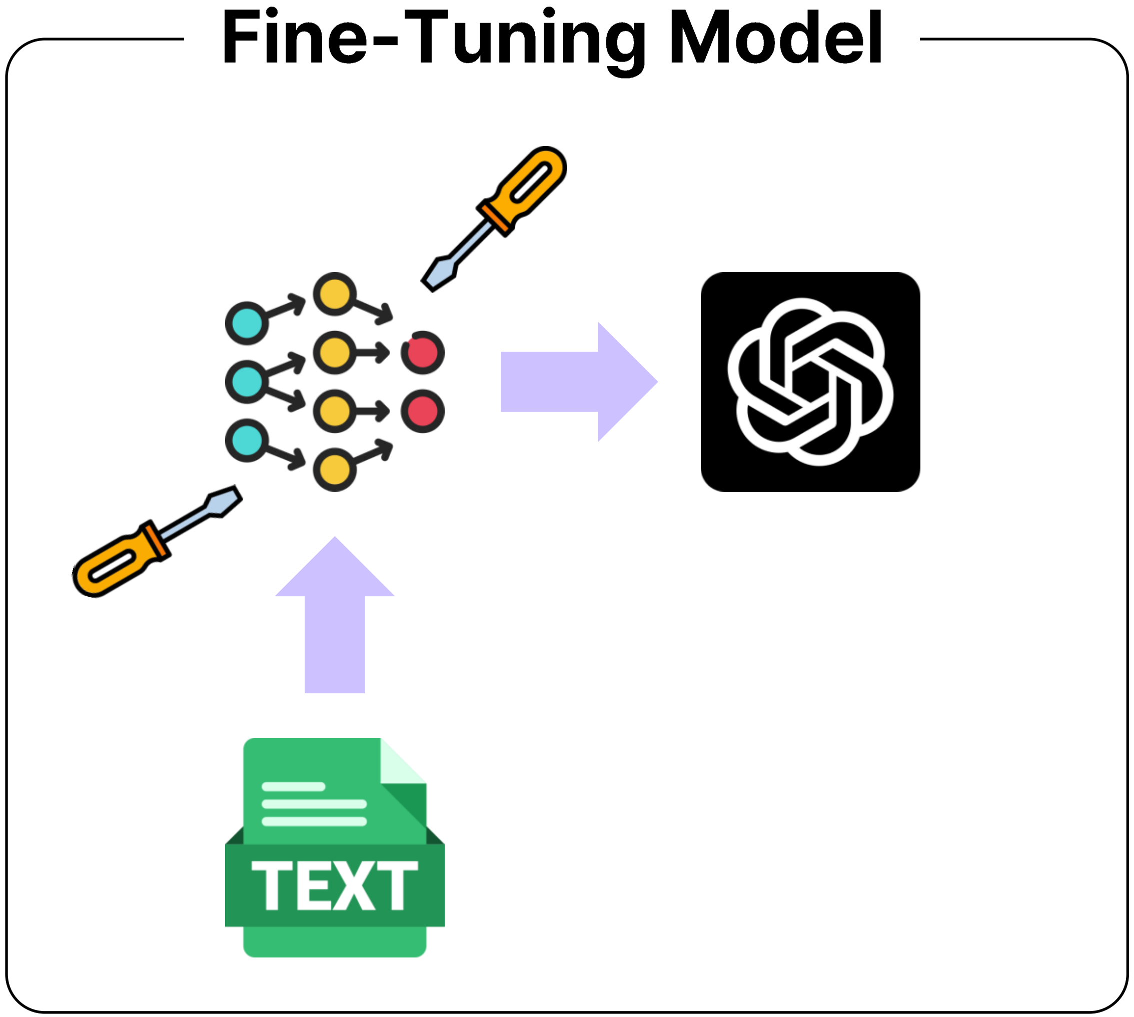
- 풀 파인 튜닝(Full Fine Tuning)
  - 모델을 구성하는 전체 파라미터를 모두 재학습 시키는 방법

Full Fine Tuning의 개념



Pre-Trained Model

model 내 w들을
수정하여 학습



Fine-Tuning Model

Full Fine Tuning의 한계

- 수 십억개의 파라미터를 모두 재학습 시켜야 하므로…
  - 학습 시간이 오래걸림
  - Task 개수만큼의 모델이 필요하므로 요구되는 저장 공간이 넓어짐
  - 효율성과 확장성이 떨어짐.

PEFT(Parameter Efficient Fine Tuning)

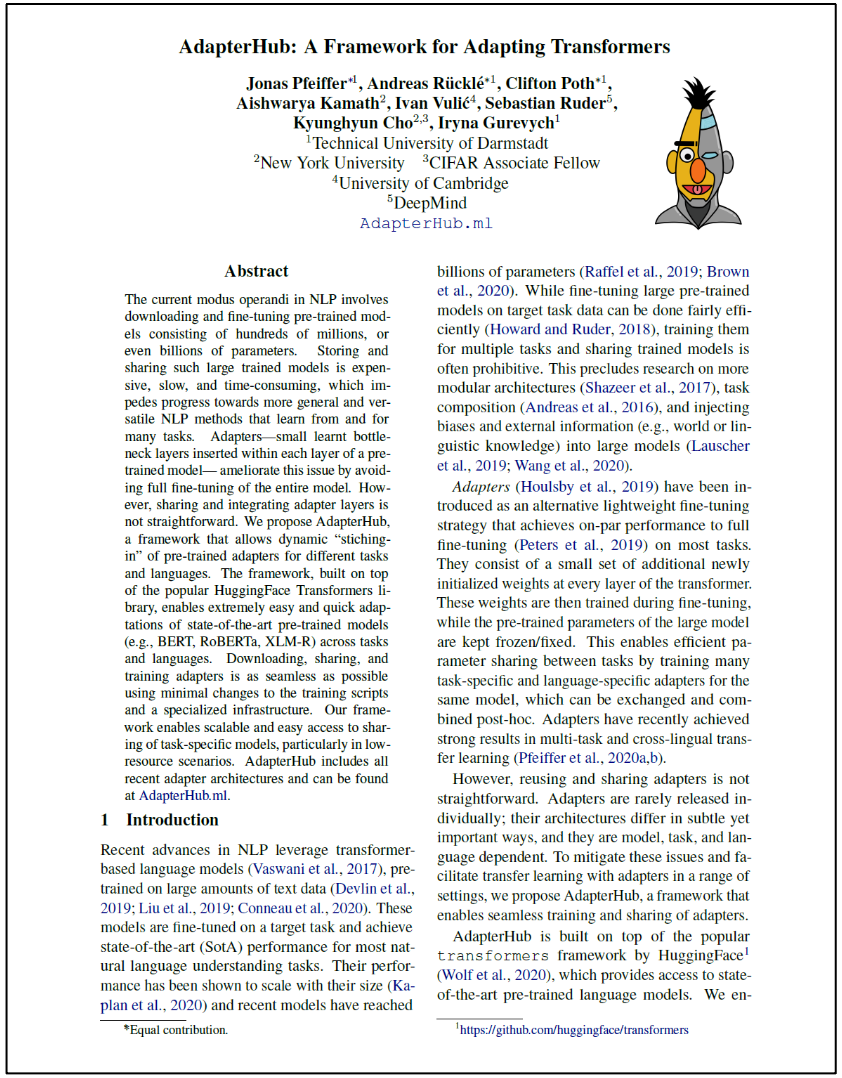
- 모델 전체를 다시 학습하지 않고, 일부만 수정하여 효율적인 학습 추구
    - 모델을 구성하는 파라미터의 대부분을 동결(freeze) 시키고,
    - 작은 학습 가능한 모듈을 추가함으로써 파인튜닝 하는 방법

GEN AI 인텐시브 과정

Section 2. Adapter

Section 2-1. Adapter의 개념

Adapter의 개념 - 기존 트랜스포머 구조에 탈부착 가능한 모듈



AdapterHub: A Framework for Adapting Transformers

Jonas Pfeiffer*[1], Andreas Rücklé*[1], Clifton Poth*[1],
Aishwarya Kamath[2], Ivan Vulić[4], Sebastian Ruder[5],
Kyunghyun Cho[2,3], Iryna Gurevych[1]
[1]Technical University of Darmstadt
[2]New York University [3]CIFAR Associate Fellow
[4]University of Cambridge
[5]DeepMind
AdapterHub.ml

Abstract

The current modus operandi in NLP involves downloading and fine-tuning pre-trained models consisting of hundreds of millions, or even billions of parameters. Storing and sharing such large trained models is expensive, slow, and time-consuming, which impedes progress towards more general and versatile NLP methods that learn from and for many tasks. Adapters—small learnt bottleneck layers inserted within each layer of a pre-trained model— ameliorate this issue by avoiding full fine-tuning of the entire model. However, sharing and integrating adapter layers is not straightforward. We propose AdapterHub, a framework that allows dynamic “stiching-in” of pre-trained adapters for different tasks and languages. The framework, built on top of the popular HuggingFace Transformers library, enables extremely easy and quick adaptations of state-of-the-art pre-trained models (e.g., BERT, RoBERTa, XLM-R) across tasks and languages. Downloading, sharing, and training adapters is as seamless as possible using minimal changes to the training scripts and a specialized infrastructure. Our framework enables scalable and easy access to sharing of task-specific models, particularly in low-resource scenarios. AdapterHub includes all recent adapter architectures and can be found at AdapterHub.ml.

1 Introduction

Recent advances in NLP leverage transformer-based language models (Vaswani et al., 2017), pre-trained on large amounts of text data (Devlin et al., 2019; Liu et al., 2019; Conneau et al., 2020). These models are fine-tuned on a target task and achieve state-of-the-art (SotA) performance for most natural language understanding tasks. Their performance has been shown to scale with their size (Kaplan et al., 2020) and recent models have reached billions of parameters (Raffel et al., 2019; Brown et al., 2020). While fine-tuning large pre-trained models on target task data can be done fairly efficiently (Howard and Ruder, 2018), training them for multiple tasks and sharing trained models is often prohibitive. This precludes research on more modular architectures (Shazeer et al., 2017), task composition (Andreas et al., 2016), and injecting biases and external information (e.g., world or linguistic knowledge) into large models (Lauscher et al., 2019; Wang et al., 2020).

Adapters (Houlsby et al., 2019) have been introduced as an alternative lightweight fine-tuning strategy that achieves on-par performance to full fine-tuning (Peters et al., 2019) on most tasks. They consist of a small set of additional newly initialized weights at every layer of the transformer. These weights are then trained during fine-tuning, while the pre-trained parameters of the large model are kept frozen/fixed. This enables efficient parameter sharing between tasks by training many task-specific and language-specific adapters for the same model, which can be exchanged and combined post-hoc. Adapters have recently achieved strong results in multi-task and cross-lingual transfer learning (Pfeiffer et al., 2020a,b).

However, reusing and sharing adapters is not straightforward. Adapters are rarely released individually; their architectures differ in subtle yet important ways, and they are model, task, and language dependent. To mitigate these issues and facilitate transfer learning with adapters in a range of settings, we propose AdapterHub, a framework that enables seamless training and sharing of adapters.

AdapterHub is built on top of the popular `transformers` framework by HuggingFace[1] (Wolf et al., 2020), which provides access to state-of-the-art pre-trained language models. We en-

*Equal contribution.

[1] https://github.com/huggingface/transformers

AdapterHub: A Framework for Adapting Transformers

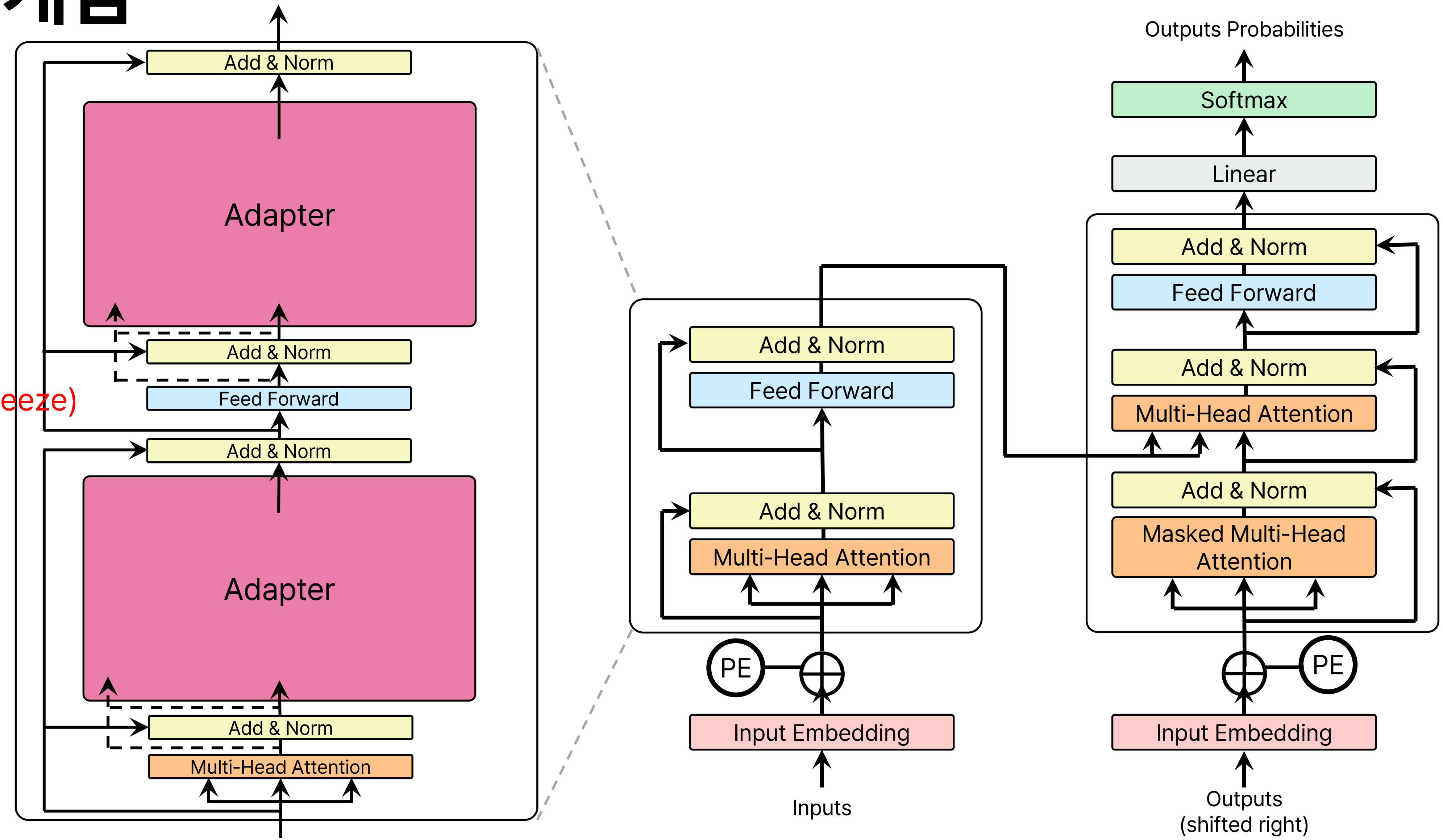
Jonas Pfeiffer et al, 2020

트랜스포머 인코더 모델인 BERT 기준으로 설명

Adapter의 개념

- LLM에 포함된 전체 파라미터를 튜닝하지 않아도, 사전 학습된 모델을 효율적으로 수정하거나 확장할 수 있는 방법
- 기존 트랜스포머 구조에 외부 모듈을 추가하는 방법
    - 어댑터만 학습하면 되므로 확장성, 모듈성이 뛰어남
    - 여러 어댑터를 조합해서 사용 가능(이후 AdapterFusion에서 설명)
- 기존 풀 파인튜닝의 단점을 보완하기 위해 등장한 방법

Adapter의 개념



전체 파라미터를
학습하는게 아니라
Adapter 내부의
파라미터만 학습

기존 파라미터들은 동결(freeze)

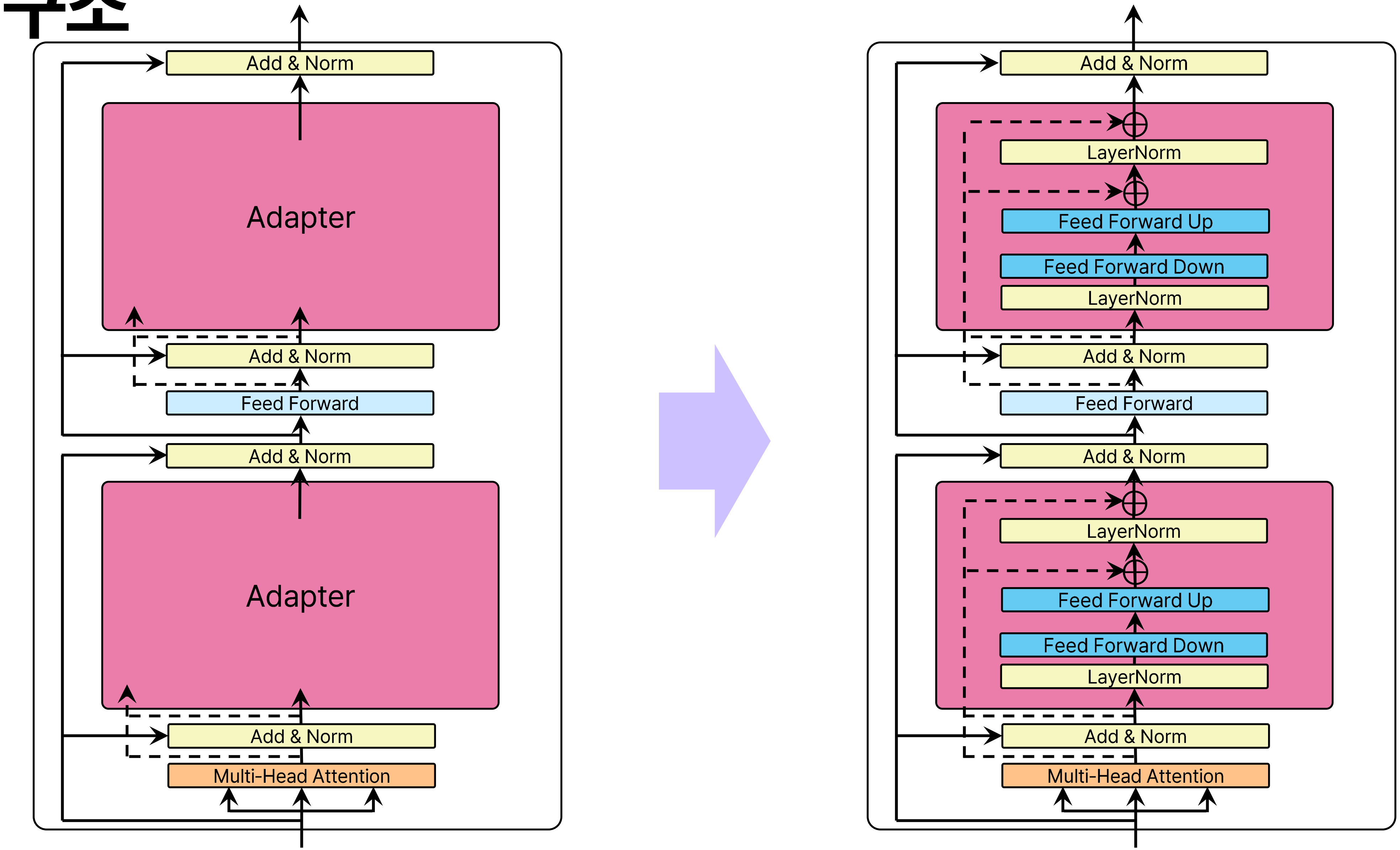
전체 파라미터를
학습하는게 아니라
Adapter 내부의
파라미터만 학습

GEN AI 인텐시브 과정

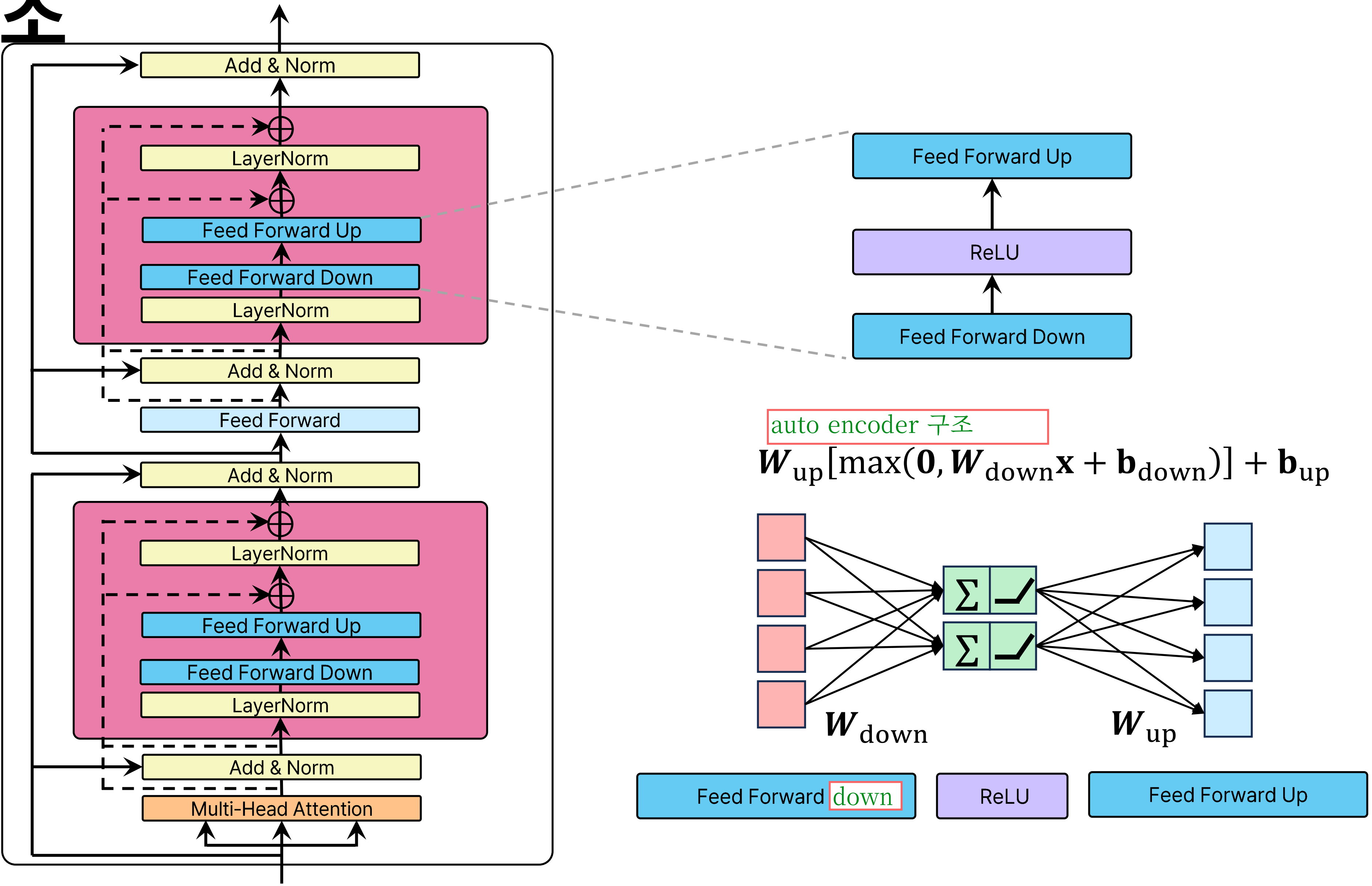
Section 2. Adapter

Section 2-2. Adapter의 구조

Adapter의 구조


Add & Norm
Adapter
Add & Norm
Feed Forward
Add & Norm
Adapter
Add & Norm
Multi-Head Attention
Add & Norm
LayerNorm
Feed Forward Up
Feed Forward Down
LayerNorm
Add & Norm
Feed Forward
Add & Norm
LayerNorm
Feed Forward Up
Feed Forward Down
LayerNorm
Add & Norm
Multi-Head Attention

Adapter의 구조



auto encoder 구조

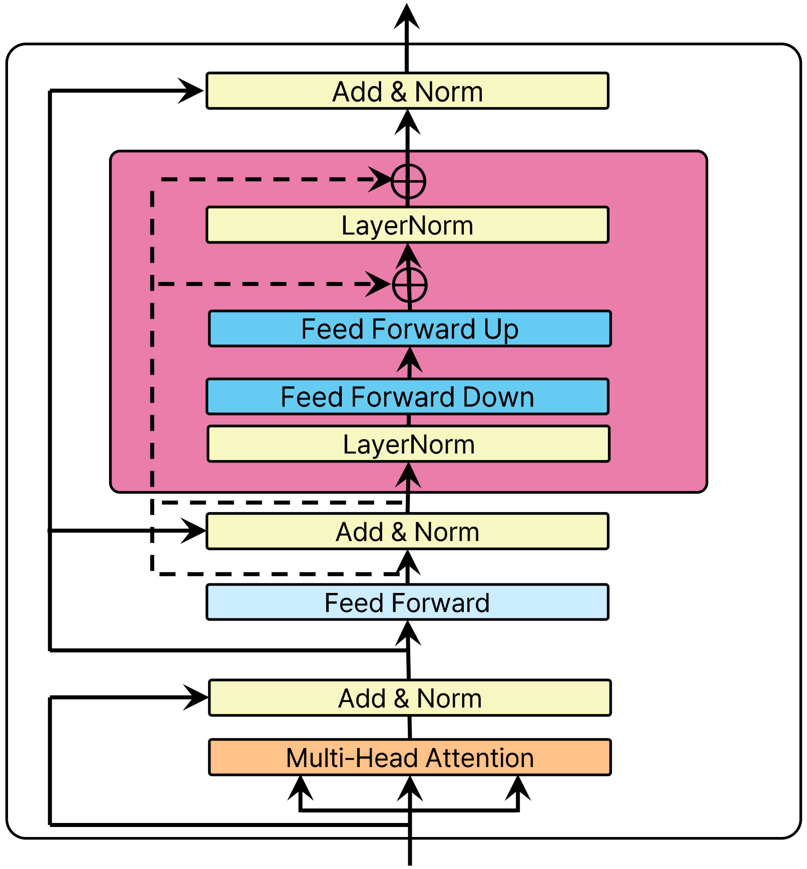
$$\boldsymbol{W}_{\text{up}}[\max(\mathbf{0}, \boldsymbol{W}_{\text{down}}\mathbf{x} + \mathbf{b}_{\text{down}})] + \mathbf{b}_{\text{up}}$$

GEN AI 인텐시브 과정

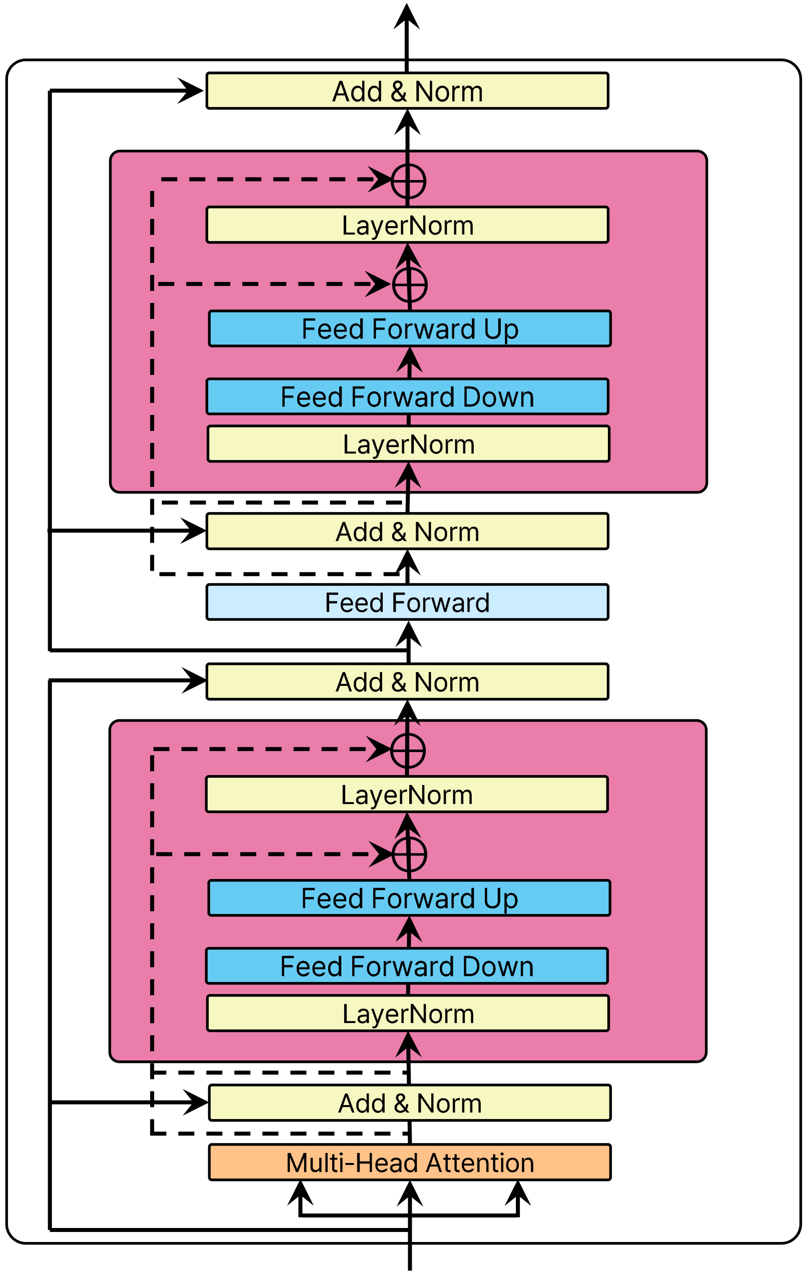
Section 2. Adapter

Section 2-3. Adapter의 종류

Adapter의 종류



Pfeiffer Architecture



Houlsby Architecture

Pfeiffer Architecture

- Adapter 삽입 위치에 따라 Pfeiffer Architecture와 Houlsby Architecture로 나뉨
- Pfeiffer Architecture
    - 하나의 Adapter만 삽입(Feed Forward 단계 이후)
    - 하나의 Adapter만 다루므로 구조가 단순하고 효율적
    - 학습해야할 파라미터가 적음
    - 작업이 단순하거나 리소스가 제한적일 때 효과적으로 사용 가능

Houlsby Architecture

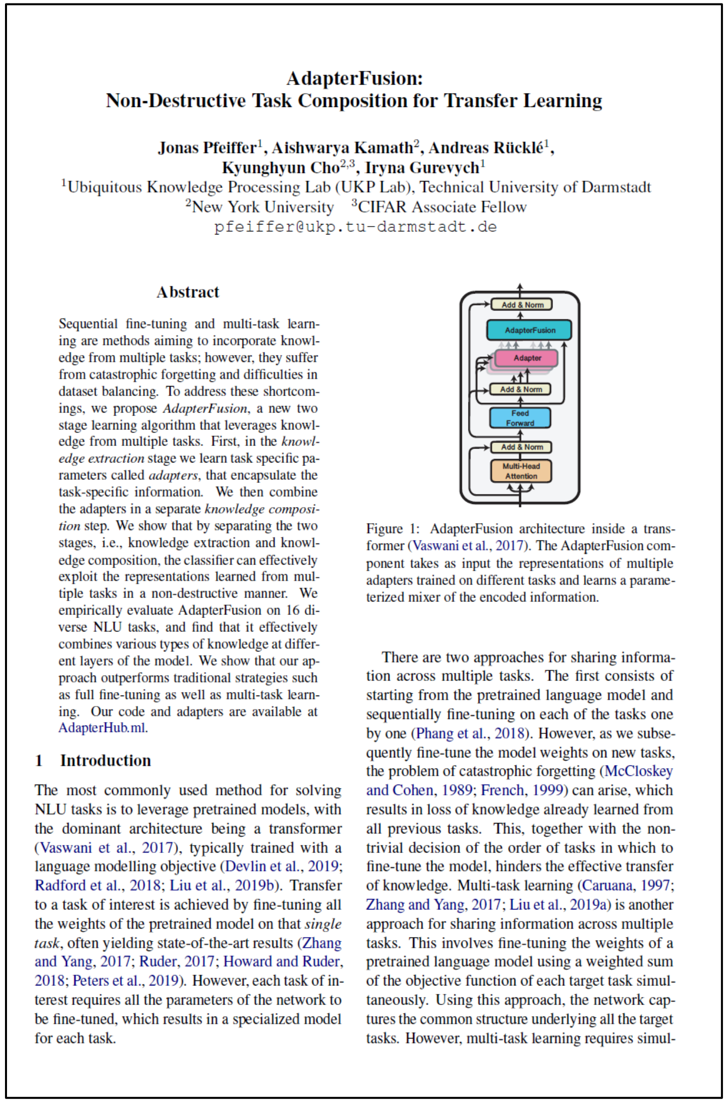
- Houlsby Architecture
    - 두 개의 Adapter만 삽입(Multi-head attention, Feed Forward 단계 이후)
    - Pfeiffer보다 구조가 복잡하고 학습해야할 파라미터가 많음
    - Adapter를 두 개 사용하므로 다양한 표현을 학습할 수 있어 성능이 더 좋은 경우가 많음
    - 복잡한 Task에 효과적

GEN AI 인텐시브 과정

Section 3. AdapterFusion

Section 3-1. AdapterFusion의 개념

AdapterFusion의 개념



AdapterFusion: Non-Destructive Task Composition for Transfer Learning

Jonas Pfeiffer[1], **Aishwarya Kamath**[2], **Andreas Rücklé**[1],
Kyunghyun Cho[2,3], **Iryna Gurevych**[1]
[1]Ubiquitous Knowledge Processing Lab (UKP Lab), Technical University of Darmstadt
[2]New York University [3]CIFAR Associate Fellow
pfeiffer@ukp.tu-darmstadt.de

Abstract

Sequential fine-tuning and multi-task learning are methods aiming to incorporate knowledge from multiple tasks; however, they suffer from catastrophic forgetting and difficulties in dataset balancing. To address these shortcomings, we propose *AdapterFusion*, a new two stage learning algorithm that leverages knowledge from multiple tasks. First, in the *knowledge extraction* stage we learn task specific parameters called *adapters*, that encapsulate the task-specific information. We then combine the adapters in a separate *knowledge composition* step. We show that by separating the two stages, i.e., knowledge extraction and knowledge composition, the classifier can effectively exploit the representations learned from multiple tasks in a non-destructive manner. We empirically evaluate AdapterFusion on 16 diverse NLU tasks, and find that it effectively combines various types of knowledge at different layers of the model. We show that our approach outperforms traditional strategies such as full fine-tuning as well as multi-task learning. Our code and adapters are available at AdapterHub.ml.

1 Introduction

The most commonly used method for solving NLU tasks is to leverage pretrained models, with the dominant architecture being a transformer (Vaswani et al., 2017), typically trained with a language modelling objective (Devlin et al., 2019; Radford et al., 2018; Liu et al., 2019b). Transfer to a task of interest is achieved by fine-tuning all the weights of the pretrained model on that *single task*, often yielding state-of-the-art results (Zhang and Yang, 2017; Ruder, 2017; Howard and Ruder, 2018; Peters et al., 2019). However, each task of interest requires all the parameters of the network to be fine-tuned, which results in a specialized model for each task.

Figure 1: AdapterFusion architecture inside a transformer (Vaswani et al., 2017). The AdapterFusion component takes as input the representations of multiple adapters trained on different tasks and learns a parameterized mixer of the encoded information.

There are two approaches for sharing information across multiple tasks. The first consists of starting from the pretrained language model and sequentially fine-tuning on each of the tasks one by one (Phang et al., 2018). However, as we subsequently fine-tune the model weights on new tasks, the problem of catastrophic forgetting (McCloskey and Cohen, 1989; French, 1999) can arise, which results in loss of knowledge already learned from all previous tasks. This, together with the non-trivial decision of the order of tasks in which to fine-tune the model, hinders the effective transfer of knowledge. Multi-task learning (Caruana, 1997; Zhang and Yang, 2017; Liu et al., 2019a) is another approach for sharing information across multiple tasks. This involves fine-tuning the weights of a pretrained language model using a weighted sum of the objective function of each target task simultaneously. Using this approach, the network captures the common structure underlying all the target tasks. However, multi-task learning requires simul-

AdapterFusion:
Non-Destructive Task Composition for Transfer Learning

Jonas Pfeiffer et al, 2020

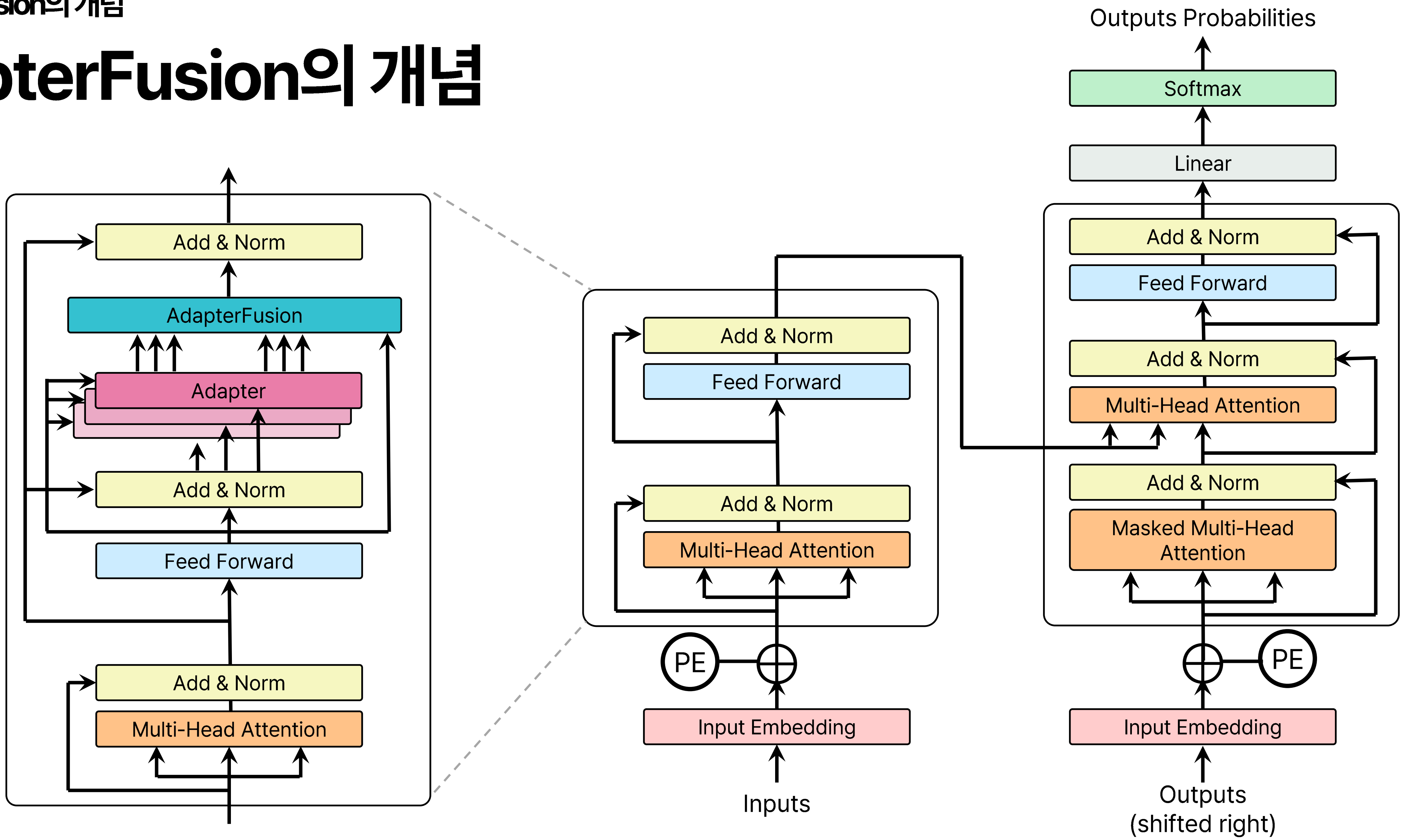
트랜스포머 인코더 모델인 BERT 기준으로 설명

AdapterFusion의 개념

- 여러 개의 Adapter들을 동시에 활용하는 방법
- 지도학습의 앙상블 학습(ensemble learning)과 비슷한 원리
- 각각의 Adapter들은 이미 사전 학습되어 있는 상황이고, 더이상 학습하지 않고 Freeze 상태 유지

- 이후 AdapterFusion 레이어만 새롭게 추가하고 AdapterFusion 레이어만 학습함
    - AdapterFusion 레이어에는 이전에 각 Adapter들의 출력값을 조합하는 파라미터로 구성되어 있음

AdapterFusion의 개념

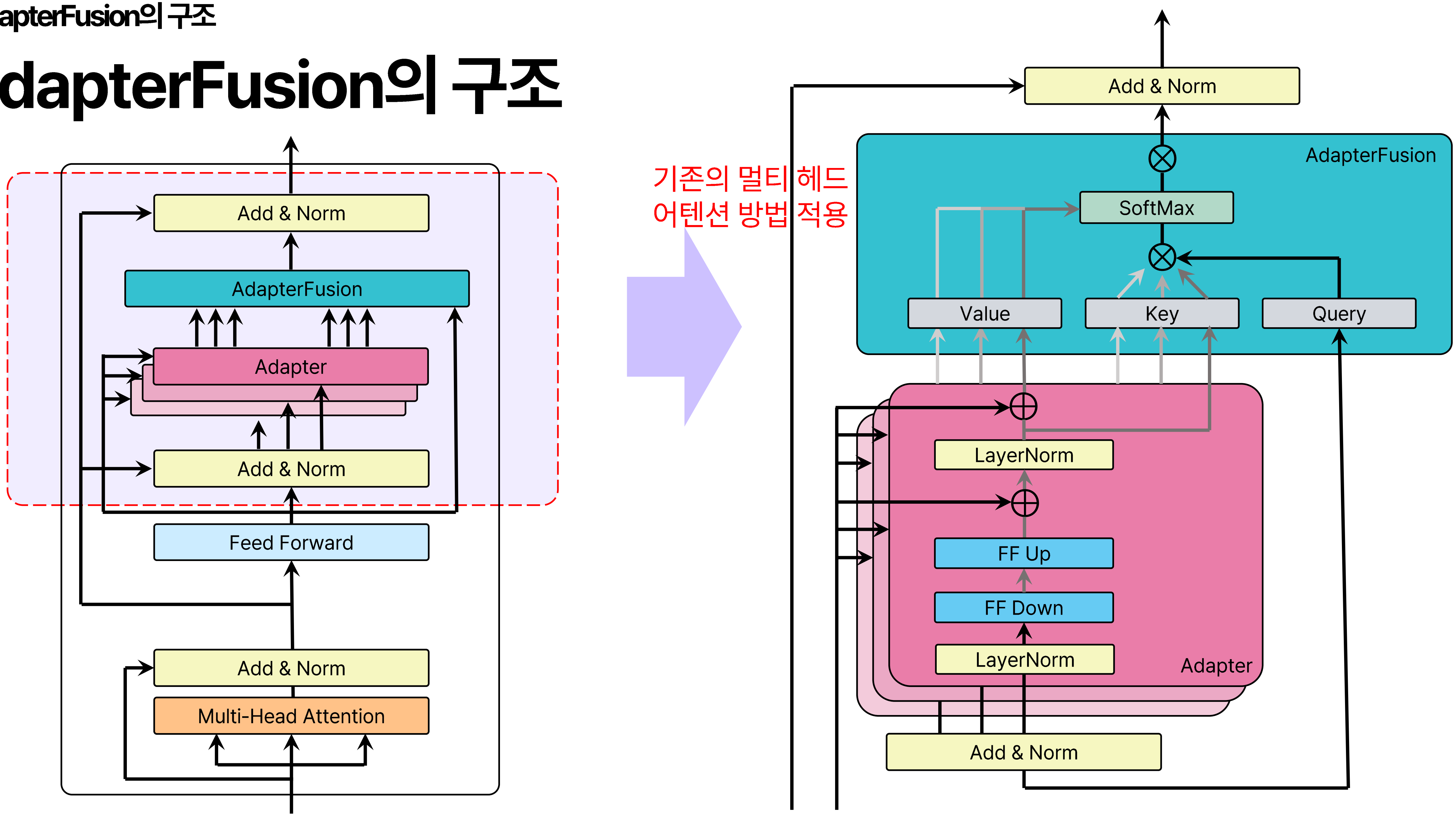


GEN AI 인텐시브 과정

Section 3. AdapterFusion

Section 3-2. AdapterFusion의 구조

AdapterFusion의 구조



기존의 멀티 헤드
어텐션 방법 적용

GEN AI 인텐시브 과정

Section 4. LoRA

Section 4-1. LoRA의 개념

LoRA의 개념

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

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **L**ow-**R**ank **A**daptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at https://github.com/microsoft/LoRA.

1 Introduction

Many applications in natural language processing rely on adapting *one* large-scale, pre-trained language model to *multiple* downstream applications. Such adaptation is usually done via *fine-tuning*, which updates all the parameters of the pre-trained model. The major downside of fine-tuning is that the new model contains as many parameters as in the original model. As larger models are trained every few months, this changes from a mere "inconvenience" for GPT-2 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a critical deployment challenge for GPT-3 (Brown et al., 2020) with 175 billion trainable parameters.[1]

Many sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to store and load a small number of task-specific parameters in addition to the pre-trained model for each task, greatly boosting the operational efficiency when deployed. However, existing techniques

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

*Equal contribution.

[0] Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.

[1] While GPT-3 175B achieves non-trivial performance with few-shot learning, fine-tuning boosts its performance significantly as shown in Appendix A.

1

LoRA:
Low-Rank Adaptation of Large Language Models

Edward Hu et al, 2021

Rank : 의미 있는 feature 개수

전체성능	성능	가격
6	3	5
4	2	8
⋮	⋮	⋮

Rank : 2
(전체 성능 = 성능 *2)로 선형 종속관계이다.

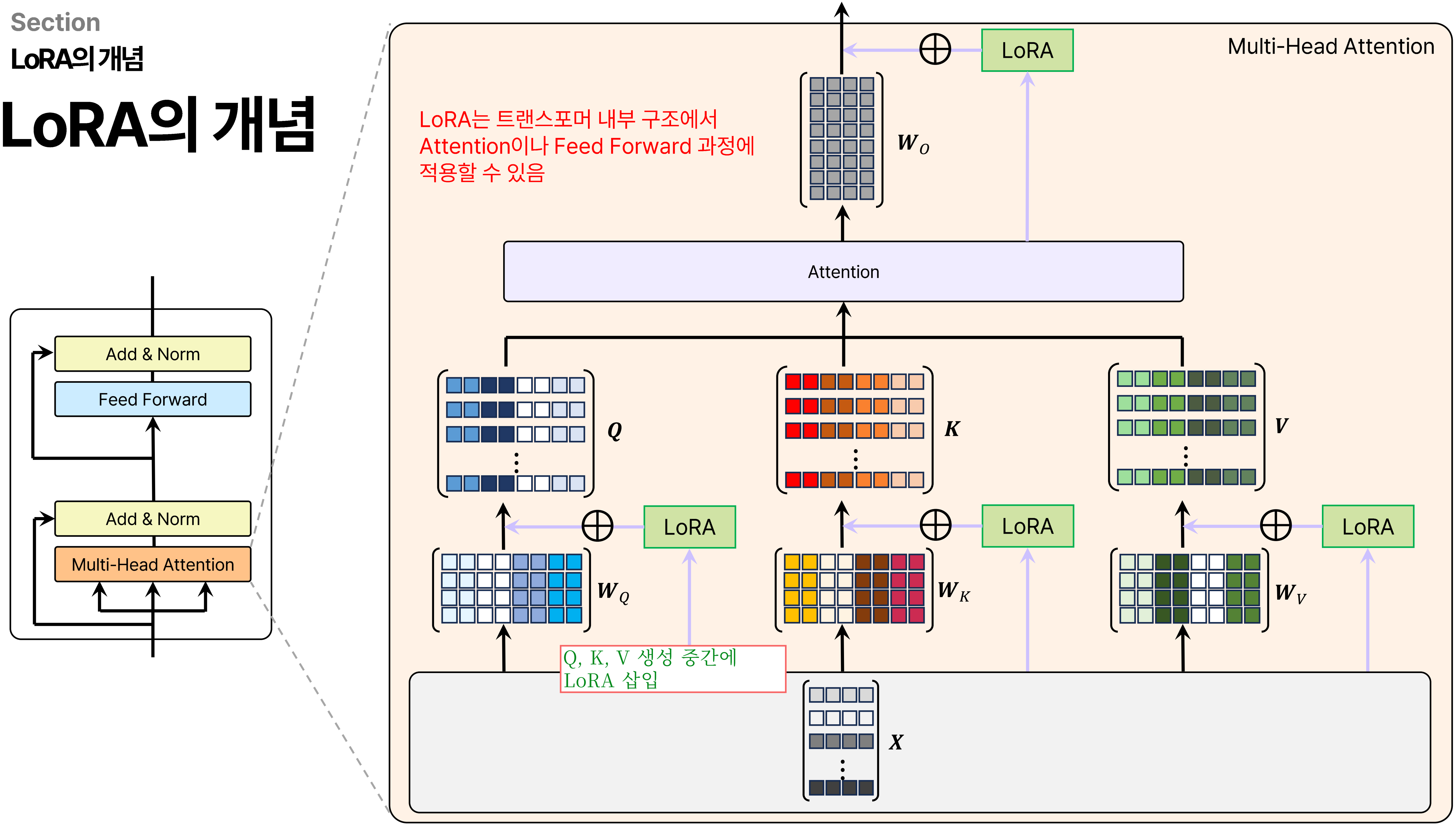
Rank 가 낮은 경우 - > 무의미한 정보일 수 있다.

LoRA의 개념

- 등장 배경
  - 풀 파인 튜닝 방식처럼 전체 파라미터를 재학습 시키기도 싫고...
  - 앞서 배운 Adapter처럼 외부 모듈을 트랜스포머 내부에 삽입하기도 싫고...

- LoRA는 기존 모델의 내부 파라미터를 수정하는 방식
  - 풀 파인 튜닝 방식처럼 모든 파라미터를 학습시키는 것이 아니라,
  - 기존 파라미터에 추가되는 보정 행렬만 학습
  - 보정 행렬의 Rank가 작아서 학습해야할 파라미터 개수 감소

LoRA의 개념



LoRA는 트랜스포머 내부 구조에서
Attention이나 Feed Forward 과정에
적용할 수 있음

Q, K, V 생성 중간에
LoRA 삽입

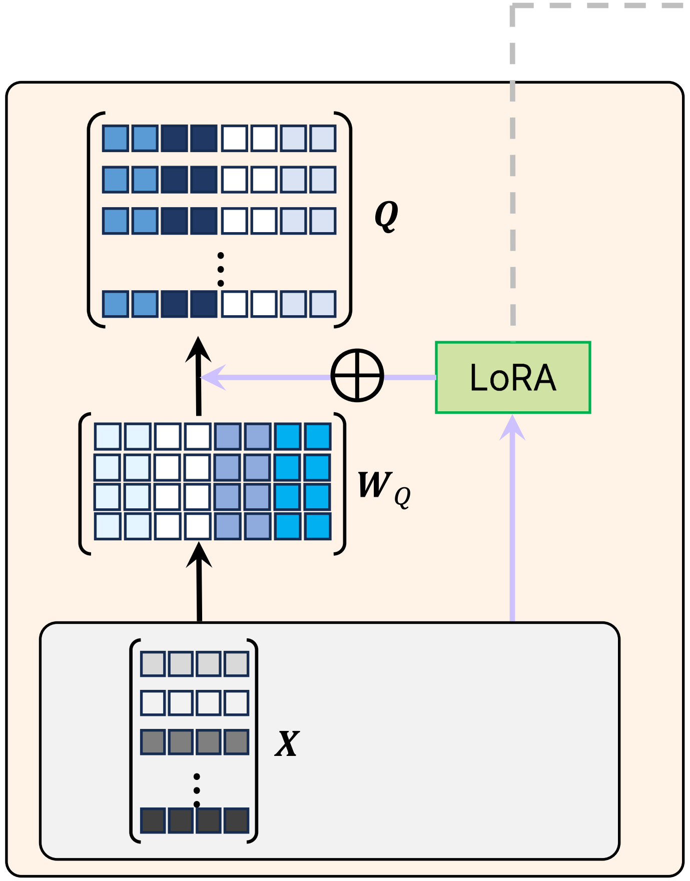
GEN AI 인텐시브 과정

Section 4. LoRA

Section 4-2. LoRA의 구조

가질수 있는 최대 rank 를 가지면 - > full rank

LoRA의 구조

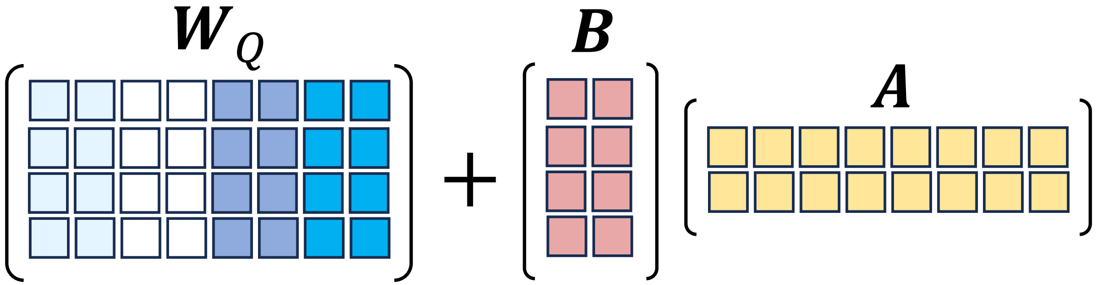


$$X(W_Q + \Delta W) = Q$$

$n \times d$ $d \times (h \cdot d_h)$ $d \times (h \cdot d_h)$

$$W_Q + BA \quad , r \ll \min(d, h \cdot d_h)$$

rank



$d \times (h \cdot d_h)$ $d \times r$ $r \times (h \cdot d_h)$

행렬 BA의 Rank는 r

감사합니다.

Q & A