

LORA

Low-Rank Adaptation Of LLM



FOM
Focus On Data Mining

INDEX

1.Introduction

2.Proposed Method

3.Conclusion

4.How To Use

01 | Introduction

Background

- BERT, GPT-3의 등장으로 집중되는 Transfer Learning, Fine-Tuning

- ✓ NLP의 특징 상 training data의 부족을 해결하기 위해 Fine-Tuning을 통한 Transfer Learning Model을 활용한 Task들이 많아지고 매우 좋은 성능을 보이고 있음
- ✓ Transfer Learning의 장점은 위처럼 성능이 좋고 다양한 Task에 활용이 가능하지만 Model의 크기가 너무 크고 무겁다는 단점이 존재함 (Parameter의 수가 너무 많음)
 - > "Full Fine-Tuning is prohibitively expensive"
- ✓ 이러한 문제점을 파악 후 Fine-Tuning을 일반화하기 위해 생기는 질문
 1. Do we need to fine-tune all the parameters?
 2. How expensive should the matrix updates be? (matrix rank에 대한 관점에서)
 - > Model의 rank를 조정해서(축소해서) fine-tuning을 경량화하자

01 | Introduction

Why LORA

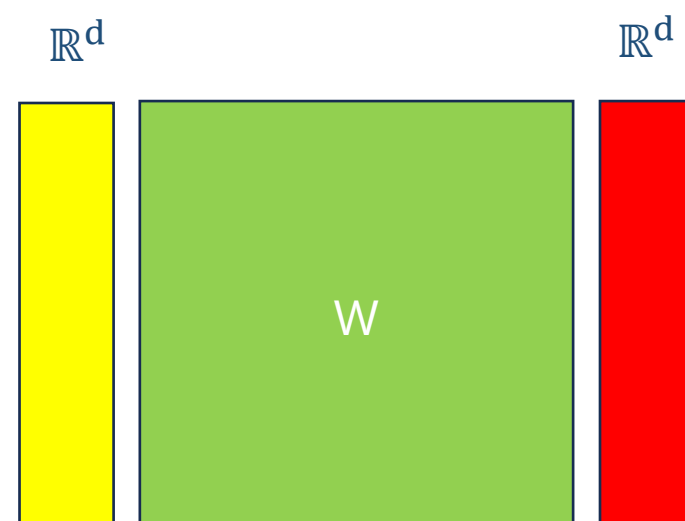
- LoRA – Low-Rank Adaptation Of LLM

- ✓ 뜻을 해석해보자면 **모델의 Rank를 조정(축소)**하여 모델의 사이즈를 줄이는 method
- ✓ "It **speeds up training** and **drastically reduces the size of model checkpoints** by training few parameters compared to the base model while **preserving the performance of the full fine-tuning**"
 - > LoRA inventor가 유튜브에서 직접 한 말
 - > 모델의 파라미터 수를 줄이고 체크포인트 수를 줄여 모델의 크기를 줄여 학습 속도를 증가시킴과 동시에 성능을 유지함과 동시에 더 향상 시킴 (LoRA의 이점)
 - > SO!! 어떻게 Model의 rank를 조정할 것이고 얼마나 줄일 것인가??

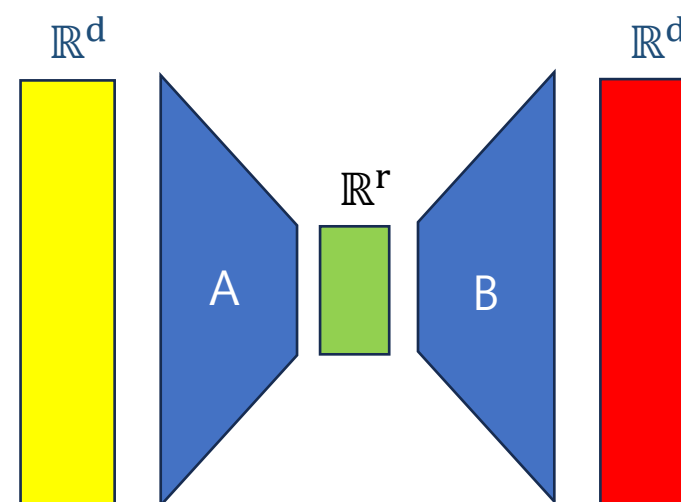
02 | Proposed Method

What is LoRA?

- LoRA는 Model의 weight matrix의 rank를 경량화 하는 method
 - ✓ LoRA는 linear transformation을 통해 Full rank matrix ($d \times d$, weight matrix)를 Low rank matrix ($r \times r$)로 축소 시킴과 동시에 여전히 output dimension을 \mathbb{R}^d 로 유지시킴



Full rank (rank = d)
 $d \times d$ parameters



Low rank (rank = $r \ll d$)
 $2 \times d \times r$ parameters

$A : d \times r$ matrix
 $B : r \times d$ matrix

여기서 !당연히! 떠올라야 하는 의문점

1. How to choose **the low rank "r"**
2. How to address **the learning deficiency** from reducing the rank?

02 | Proposed Method

What is LoRA?

- LoRA는 Model의 weight matrix의 rank를 경량화 하는 method

✓ LoRA는 linear transformation을 통해 Full rank matrix ($d \times d$, weight matrix)를 Low rank matrix ($r \times r$)로 축소 시킴과 동시에 여전히 output dimension을 \mathbb{R}^d 로 유지시킴

- 기존의 full fine-tuning objective function

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

- LoRA를 통해 경량화 한 adaptive fine-tuning objective function

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

이때, $|\Theta| \ll |\Phi|$ 임

➔ 이러한 가정을 통해 모델을 경량화 및 최적화 속도를 빠르게 함

02 | Proposed Method

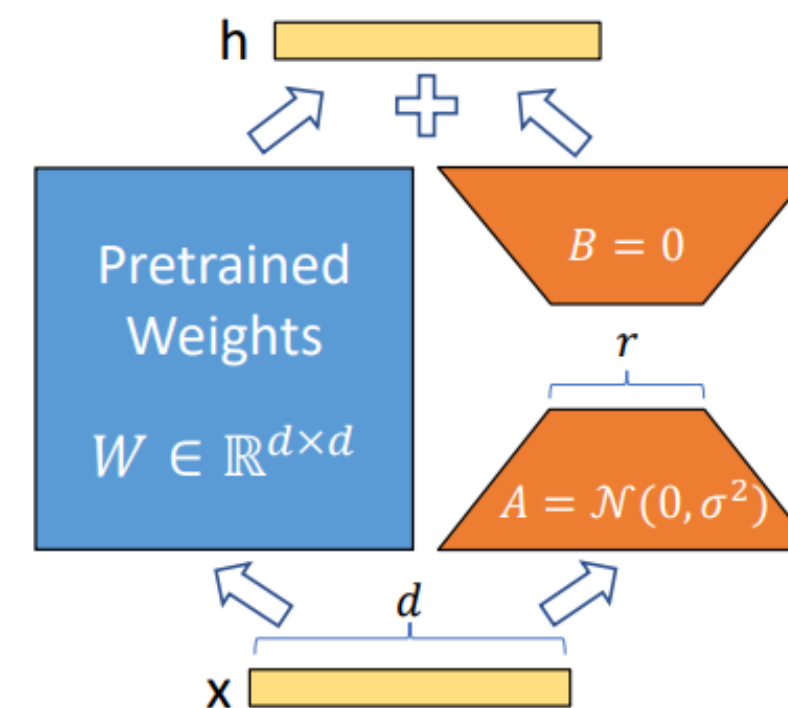
What is LoRA?

• LoRA에서의 h 수식

- ✓ Adaptive optimizer인 LoRA가 최종적으로 만들어내는 h를 살펴보자

$$h = W_0x + \Delta Wx = W_0x + BAx$$

- ✓ 이때, W_0 는 pre-trained weight matrix, ΔW 는 LoRA에서 연산되는 새로운 weight matrix (BA , where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times d}$, $BA \in \mathbb{R}^{d \times d}$)



02 | Proposed Method

How to choose the low rank “r”

- 앞에서 본 것처럼 r 의 크기는 매우 중요
 - ✓ ΔWx scaled by $\frac{\alpha}{r}$, where α is a constant in r
 - > Optimizing with Adam, tuning α is roughly the same as tuning the learning rate
simply set α to the first r (we try and do not tune it)
 - ✓ 즉, Adam의 lr 최적화 방식과 비슷하게 r 의 크기를 최적화 함

02 | Proposed Method

How to address the learning deficiency from reducing the rank?

- 다양한 LoRA Experiment

- ✓ RoBERTa_base, RoBERTa_large, DeBERTa_XXL, GPT-2 M, GPT-2 L, GPT-3을 여러가지 adapter tuning, FT(Fine-Tuning), BitFit 등 많은 방식들과 비교해봤지만 LoRA가 Trainable Parameters 수를 현저히 많이 줄임과 동시에 성능이 많이 저하되지 않고 오히려 더 좋은 성능을 보였음
- > LoRA는 rank reduction을 활용하여 경량화 하였지만, 경량화 과정에서 성능이 저하되지는 않았음

02 | Proposed Method

Benefits of using LoRA

- Reduction of checkpoints sizes

- ✓ LoRA leads to smaller checkpoints

- ✓ Ex) Full-finetuning : 175,000,000,000 trainable parameters, 1TB / checkpoint

- LoRA : 4,700,000 trainable parameters, 25MB / checkpoint

- Doesn't introduce any inference latency

- ✓ LoRA add additional weight to pretrained weights(Original parameters)

- ✓ Final Adapted Weight = Pretrained Weight(Original parameters) + Update Weight(Generated by LoRA, Reduction of the matrix (r x d))

- > At this moment, There is "NO additional latency" because of the LoRA's simplicity (+ by parallel)

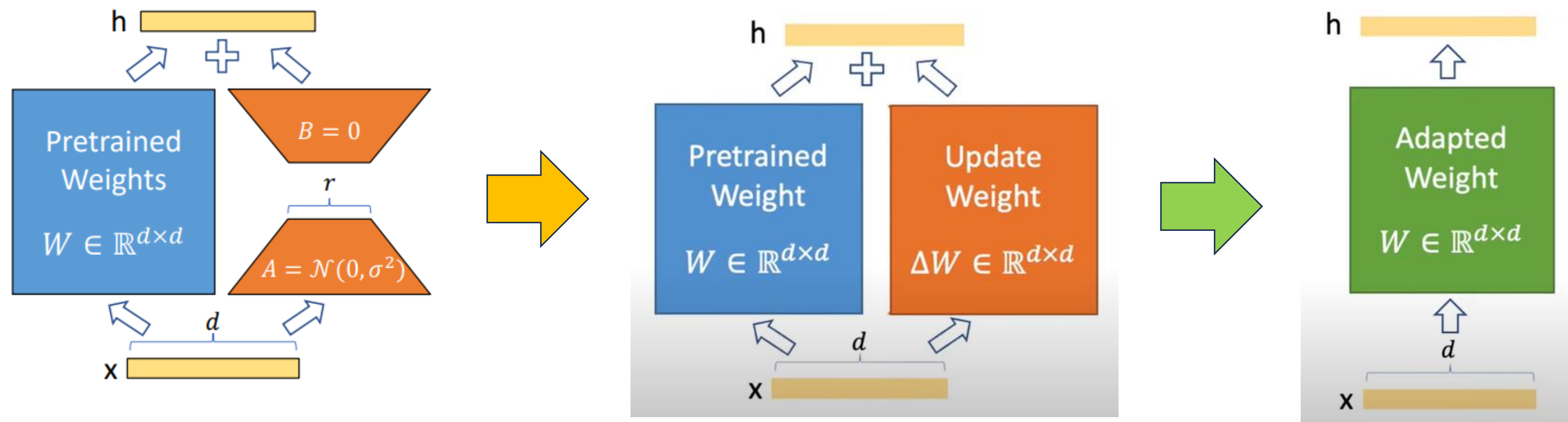
- > Simply just subtracting the updates, We can get original parameters

02 | Proposed Method

Benefits of using LoRA

• Adapted Weight

- ✓ Adapted Weight = Pretrained Weight + Update Weight
- ✓ Pretrained Weight = Original parameters, Update Weight = made by LoRA



LoRA를 정리하자면, Transfer Learning에서 Fully Fine-Tuning을 거치지 않고 Original parameters에 Update Weight을 더해줌으로써 Pretrained model을 Low rank Fine-Tuning하는 method

03 | Conclusion

Simplification of fine-tuning

- LoRA는 Transfer Learning & Fine-Tuning을 문제점인 너무 많은 파라미터 및 너무 큰 모델의 크기를 Low Rank Fine-Tuning으로 해결한 method
 - ✓ 최근의 NLP 동향은 The Bigger, The Better의 동향임을 고려하였을 때, Fine-Tuning의 경량화는 매우 중요하고 유익한 Task임을 알 수 있음
 - > LLM을 효율적으로 튜닝할 수 있음
 - ✓ 다른 Adapter류의 기법과 다르게 Additional latency가 발생하지 않음
 - ✓ LoRA는 대부분의 dense layer에 적용 가능함

04

How To Use

How to use LoRA in python

Reference : <https://github.com/fshnkarimi/Fine-tuning-an-LLM-using-LoRA>

- Installation

- ✓ LoRA installation

- Usage

- ✓ How to use LoRA

- Model Architecture

- ✓ Model Arch with LoRA

- Training

- ✓ How to fine-tuning with LoRA

04 | How To Use Installation_library

• Installation

- ✓ !pip install datasets
- !pip install transformers
- !pip install peft
- !pip install evaluate

```
from datasets import load_dataset, DatasetDict, Dataset
```

```
from transformers import (
```

```
    AutoTokenizer,
```

```
    AutoConfig,
```

```
    AutoModelForSequenceClassification,
```

```
    DataCollatorWithPadding,
```

```
    TrainingArguments,
```

```
    Trainer)
```

```
from peft import PeftModel, PeftConfig, get_peft_model, LoraConfig
```

```
import evaluate
```

```
import torch
```

```
import numpy as np
```

04 | How To Use Usage

Reference : <https://huggingface.co/docs/peft/quicktour>

- Usage

- ✓ Load your dataset
- ✓ Define the LoRA model architecture
 - > Define backbone model
 - ex) GPT- neo 1.3b, Roberta-base, ...
- ✓ Tokenize the dataset
- ✓ Train the LoRA model

04 | How To Use Usage

- Usage

- ✓ Train the LoRA model

First, What is **PEFT (Parameter-Efficient Fine-Tuning)**

-> PEFT : Reducing LLM Fine Tuning cost

-> To Use PeftModel we have to define PeftConfig that stores all the important parameters for building PeftModel

Second, Define **LoraConfig** (= PeftConfig)

1. task_type: the task to train for (ex, sequence-to-sequence language modeling)
2. inference_mode: whether you're using the model for inference or not
3. r: the dimension of the low-rank matrices
4. lora_alpha: the scaling factor for the low-rank matrices
5. lora_dropout: the dropout probability of the LoRA layers

04 | How To Use Usage

- LoRA Config in python

- ✓ EX)

```
from peft import LoraConfig, TaskType
peft_config = LoraConfig(task_type=TaskType.SEQ_2_SEQ_LM,
                        inference_mode=False,
                        r=8,
                        lora_alpha=32,
                        lora_dropout=0.1)
```

04 | How To Use

Usage

- Training Peft Model

- ✓ We can use Transformers Trainer, Accelerate, or any custom Pytorch training loop

Ex)

```
training_args = TrainingArguments(  
    output_dir="your-name/bigscience/mt0-large-lora", -> backbone model 디렉토리
```

```
    ----- 하이퍼파라미터 조정 -----  
    learning_rate=1e-3,  
    per_device_train_batch_size=32,  
    per_device_eval_batch_size=32,  
    num_train_epochs=2,  
    weight_decay=0.01,  
    evaluation_strategy="epoch",  
    save_strategy="epoch",  
    load_best_model_at_end=True,
```

```
)
```

04 | How To Use Usage

- Training Peft Model

- ✓ Pass final model, training arguments, dataset, tokenizer, and any other necessary component to Trainer

Ex)

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=tokenized_datasets["train"],  
    eval_dataset=tokenized_datasets["test"],  
    tokenizer=tokenizer,  
    data_collator=data_collator,  
    compute_metrics=compute_metrics,  
)  
  
trainer.train()
```

04 | How To Use Usage

- Save model

- ✓ `model.save_pretrained("output_dir")`

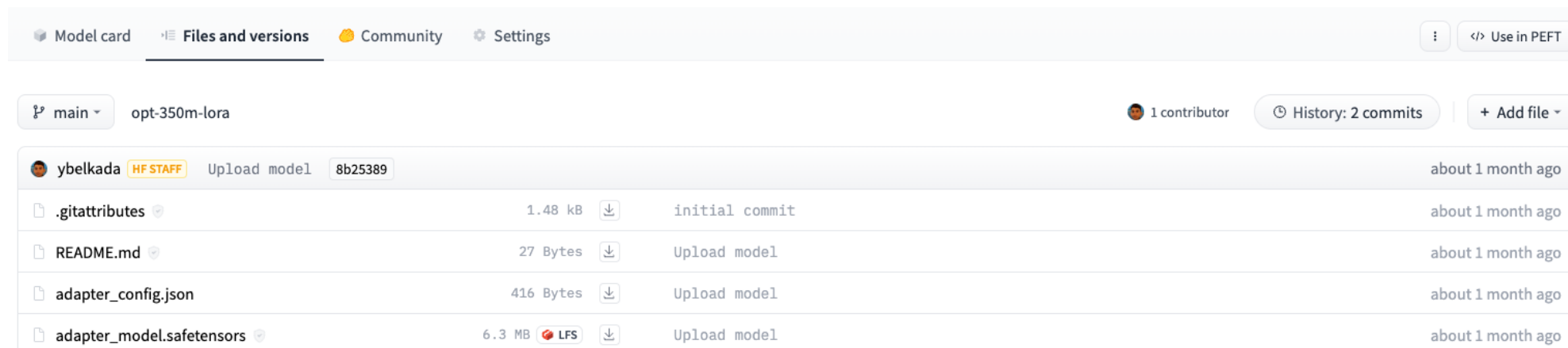
- > Fine-Tuning 된 Model 저장

- Save model to the Hub

- ✓ `from huggingface_hub import notebook_login`

`notebook_login()`

`model.push_to_hub("your-name/bigscience/mt0-large-lora")`



The screenshot shows the HuggingFace Hub interface for a repository named 'opt-350m-lora' by user 'ybelkada'. The repository has 1 contributor and 2 commits. The file list includes:

File Name	Size	Upload Method	Time
.gitattributes	1.48 kB	initial commit	about 1 month ago
README.md	27 Bytes	Upload model	about 1 month ago
adapter_config.json	416 Bytes	Upload model	about 1 month ago
adapter_model.safetensors	6.3 MB	Upload model	about 1 month ago

-> HuggingFace Hub에 업로드 된 Fine-Tuning된 Model



Q & A

감사합니다