간단 논세

# LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

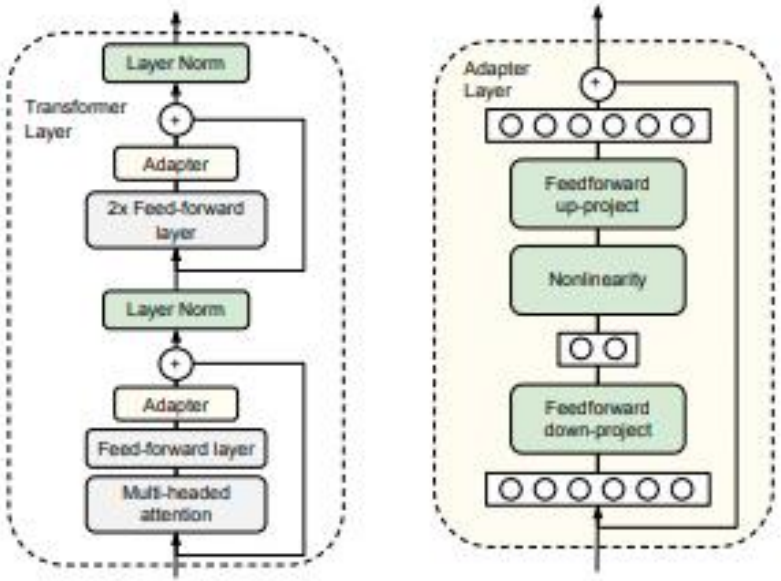
Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).

2024.09.27 유하영

# Background

다운스트림 태스크는 일반적으로 사전 학습된 모델의
모든 파라미터를 업데이트하는 fine-tuning을 통해 수행된다.

추가되는 어댑터에 대해서만 **학습**
-> 어댑터 레이어들은 latency가 증가한다




prompt 최적화

adaptation을 위해 시퀀스 길이의 일부를 사용하면 다운스트림 task를 처리하는 데 사용할 수 있는 시퀀스 길이가 필연적으로 줄어들어 다른 방법에 비해 프롬프트 튜닝 성능이 떨어진다.

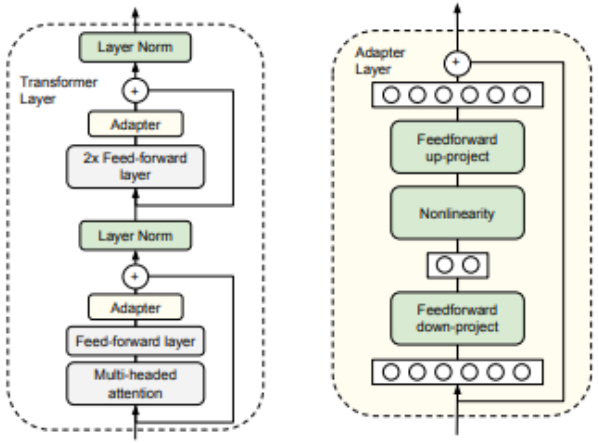
# LLM Tuning Methods

**Prompt Engineering** ← **PEFT** (Parameter Efficient Fine-Tuning) ← **RAG** (Retrieval Augmented Generation) ← **Full Fine-Tuning**

Cost ~ Complexity ~ Quality

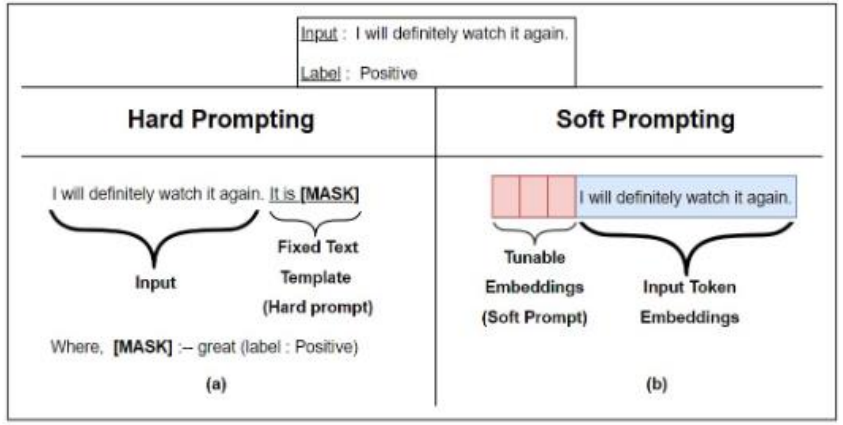
## 1. Adapter Layers

pre-trained model 사이사이에 **feed-forward networks**(학습 가능한 작은 신경망 층)를 **삽입**




## 2. Prompt Tuning

**프롬프트의 일부를 학습 가능한 매개변수**로 변환하여 훈련할 수 있도록 한 것 (입력 프롬프트에 대하여)




Hard Prompt vs. Soft Prompt (Senadeera & Ive, 2022)

## 3. LoRA / QLoRA (Low-Rank Adaptation)

pre-trained model의 원래 가중치는 유지하면서 학습 가능한 저차원 행렬인 lank decomposition 행렬을 삽입하여 소수의 파라미터만 조정하는 기법

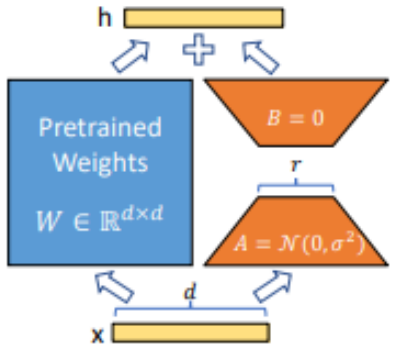



Figure 1: Our reparametrization. We only train $A$ and $B$.

https://velog.io/@sobit/LLM-%EB%8F%84%EB%A9%94%EC%9D%B8-%EC%A0%81%EC%9A%A9-%EC%96%B4%EB%96%A4-%EB%B0%A9%EB%B2%95%EC%9D%B4-%EC%A2%8B%EC%9D%84%EA%B9%8C-Full-Fine-Tuning-PEFT-Prompt-Engineering-RAG / https://www.linkedin.com/pulse/full-fine-tuning-peft-prompt-engineering-rag-which-one-right-you/

# LoRA

## LoRA
## :Low-Rank Adaptation

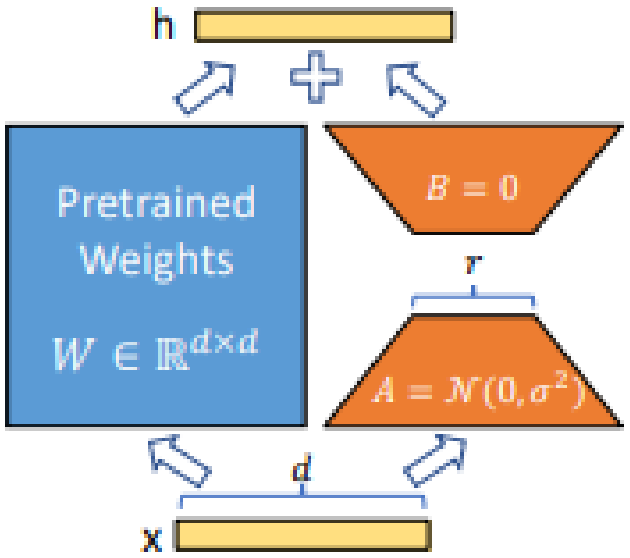



Figure 1: Our reparametrization. We only train $A$ and $B$.

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

사전 학습된 가중치를 고정된 상태로 유지하면서
추가적인 Low-Rank matrix를 학습하여
신경망의 일부 레이어를 간접적으로 학습

- 전체 fine-tuning을 일반화 할 수 있다.
  가중치 행렬에 대한 누적 기울기 업데이트가 필요하지 않다.
- 다른 다운스트림 task로 전환해야 하는 경우 $BA$를 뺀 다음
  다른 $BA$을 추가하여 $W$를 복구할 수 있다.

# Applying LoRA to Transformer

학습 가능한 파라미터의 수를 줄이기 위해
신경망에서 가중치 행렬의 모든 부분집합에 LoRA를 적용할 수 있다.

| | # of Trainable Parameters = 18M | | | | | | |
|---|---|---|---|---|---|---|---|
| Weight Type | $W_q$ | $W_k$ | $W_v$ | $W_o$ | $W_q, W_k$ | $W_q, W_v$ | $W_q, W_k, W_v, W_o$ |
| Rank $r$ | 8 | 8 | 8 | 8 | 4 | 4 | 2 |
| WikiSQL ($\pm 0.5\%$) | 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** |

LoRA 기법을 사용해 적은 수의 파라미터로도 높은 성능을 유지할 수 있음을 시사

# Result

| Model & Method | # Trainable Parameters | MNLI | SST-2 | MRPC | CoLA | QNLI | QQP | RTE | STS-B | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| $\text{RoB}_{\text{base}}$ (FT)* | 125.0M | **87.6** | 94.8 | 90.2 | **63.6** | 92.8 | **91.9** | 78.7 | 91.2 | 86.4 |
| $\text{RoB}_{\text{base}}$ (BitFit)* | 0.1M | 84.7 | 93.7 | **92.7** | 62.0 | 91.8 | 84.0 | 81.5 | 90.8 | 85.2 |
| $\text{RoB}_{\text{base}}$ ($\text{Adpt}^{\text{D}}$)* | 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 |
| $\text{RoB}_{\text{base}}$ ($\text{Adpt}^{\text{D}}$)* | 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 |
| $\text{RoB}_{\text{base}}$ (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** |
| $\text{RoB}_{\text{large}}$ ($\text{Adpt}^{\text{P}}$)† | 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 |
| $\text{RoB}_{\text{large}}$ ($\text{Adpt}^{\text{P}}$)† | 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 |
| $\text{RoB}_{\text{large}}$ ($\text{Adpt}^{\text{H}}$)† | 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 |
| $\text{RoB}_{\text{large}}$ ($\text{Adpt}^{\text{H}}$)† | 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** |
| $\text{DeB}_{\text{XXL}}$ (FT)* | 1500.0M | 91.8 | **97.2** | 92.0 | 72.0 | **96.0** | 92.7 | 93.9 | 92.9 | 91.1 |
| $\text{DeB}_{\text{XXL}}$ (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** |

| 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 ($\text{FT}^{\text{Top2}}$)* | 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 | $\mathbf{70.4}_{\pm.1}$ | $\mathbf{8.89}_{\pm.02}$ | $\mathbf{46.8}_{\pm.2}$ | $\mathbf{72.0}_{\pm.2}$ | $2.47_{\pm.02}$ |

# 이점

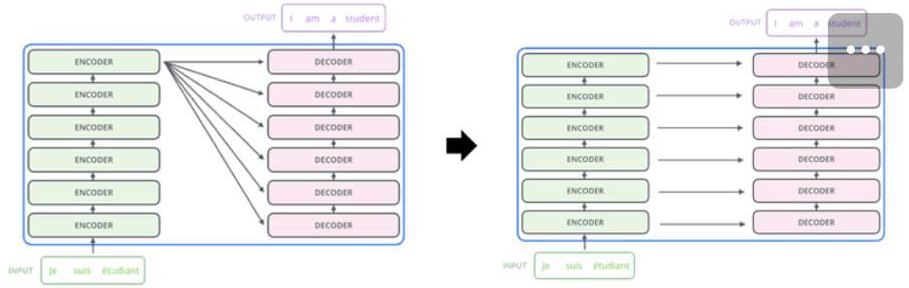
- 메모리와 스토리지 사용량 감소
- 모든 파라미터가 아닌 LoRA 가중치만 교환함으로써 훨씬 저렴한 비용으로 배포 중에 task 사이를 전환

# 8주차

# P-Transformer 모델 구현

## 8주차: P-Transformer 모델 구현

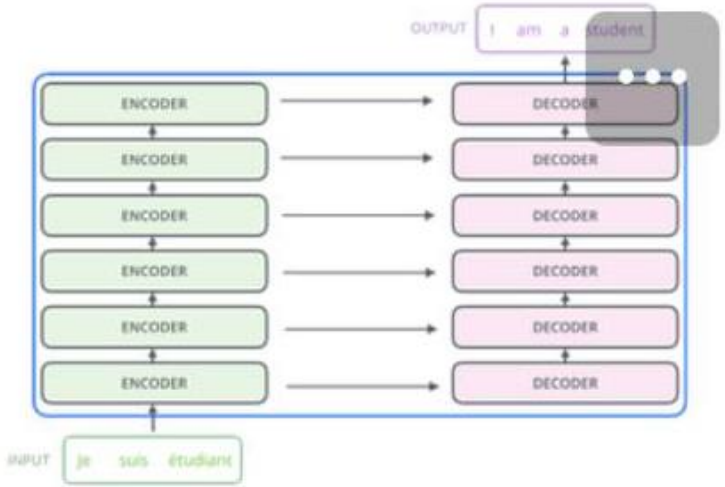
- 목적: Transformer모델을 원하는 방식대로 변형해서 쓸 수 있도록 구현 연습
- 내용:
  - Original Transformer를 P-Transformer로 변형




(왼쪽: Original Transformer; 오른쪽: Parallel Transformer (P-Transformer)

  - 5주차와 동일한 데이터셋에 대해서도 BLEU-4 Score 0.15 이상 목표

# P-Transformer




**인코더와 디코더의 병렬학습 진행**

인코더->디코더 -> 인코더->디코더 ..
순으로
레이어들이 순차적으로 통과되도록
구성

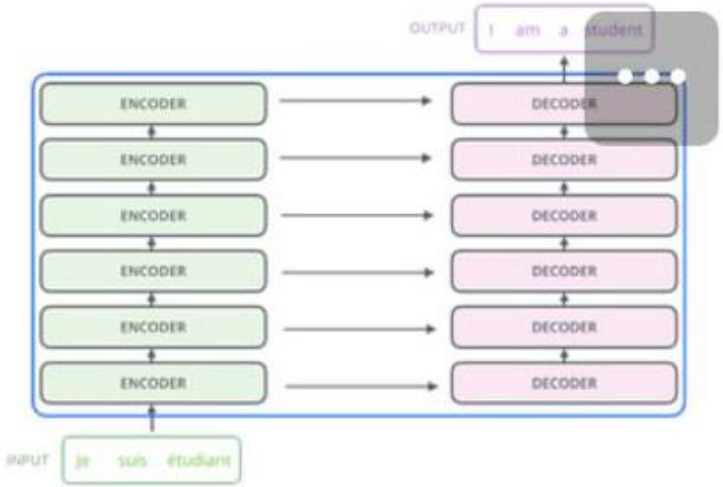
기존 모델

P-Transformer

Teacher Forcing
추가

# P-Transformer

기존 모델




```python
def forward(self, src, trg, src_mask, tgt_mask, src_padding_mask, tgt_padding_mask, memory_key_padding_mask):
    src_emb = self.positional_encoding(self.src_tok_emb(src))
    tgt_emb = self.positional_encoding(self.tgt_tok_emb(trg))

    #Encoder
    memory = src_emb
    for layer in self.enc_layers:
        memory = layer(memory, src_mask, src_padding_mask)
    #Decoder
    output = tgt_emb
    for layer in self.dec_layers:
        output = layer(output, memory, tgt_mask, None, tgt_padding_mask, memory_key_padding_mask)

    return self.generator(output)
```

**인코더와 디코더의 병렬학습 진행**

인코더->디코더 -> 인코더->디코더 .. 순으로
레이어들이 순차적으로 통과되도록 구성

P-Transformer

```python
def forward(self, src, trg, src_mask, tgt_mask, src_padding_mask, tgt_padding_mask, memory_key_padding_mask):
    src_emb = self.positional_encoding(self.src_tok_emb(src))
    tgt_emb = self.positional_encoding(self.tgt_tok_emb(trg))

    memory = src_emb
    output = tgt_emb

    #인코더->디코더 순으로 레이어들을 순차적으로 통과
    for i in range(self.num_layers):
        #인코더
        memory = self.enc_layers[i](memory, src_mask, src_padding_mask)

        #디코더
        output = self.dec_layers[i](output, memory, tgt_mask, None, tgt_padding_mask, memory_key_padding_mask)

    return self.generator(output)
```

# P-Transformer

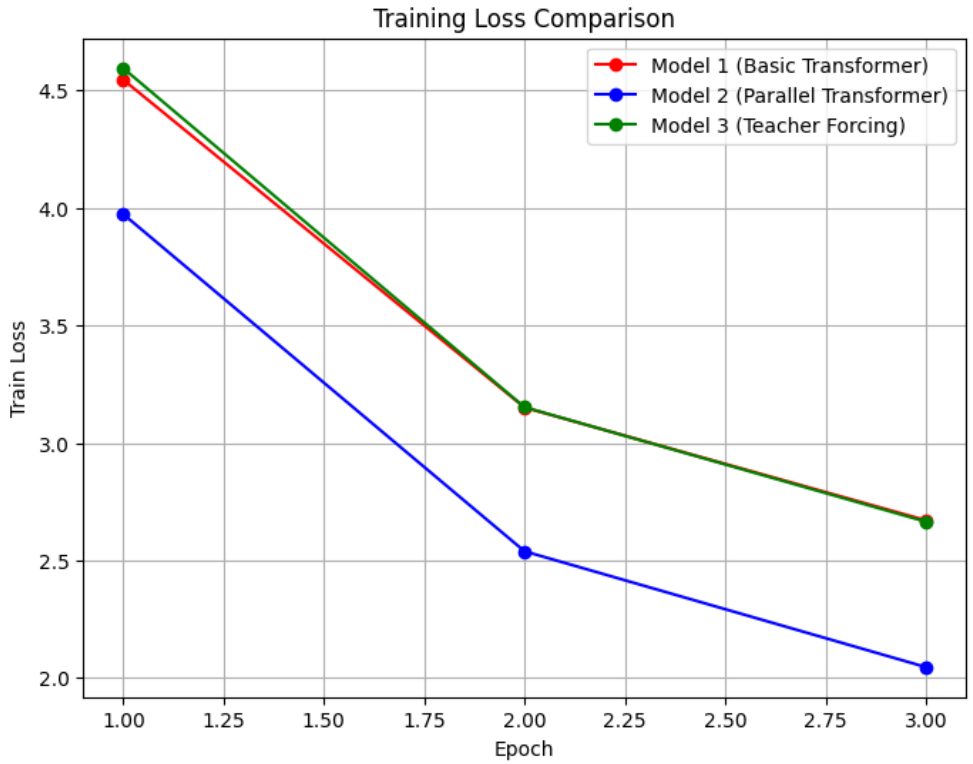
P-Transformer

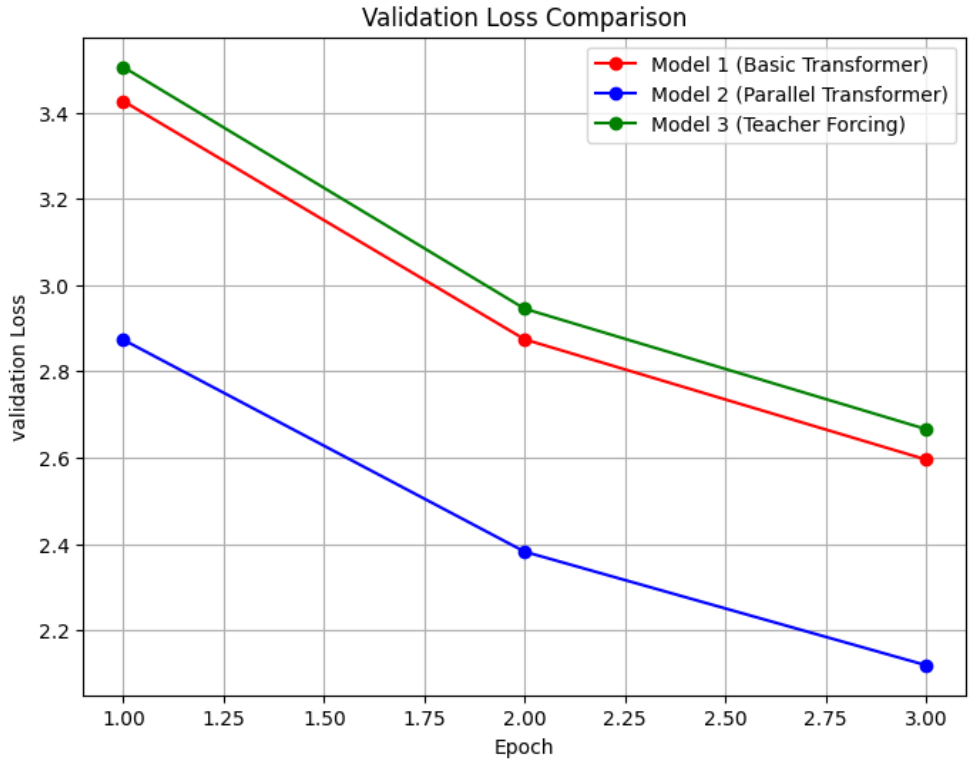
```
def forward(self, src, trg, src_mask, tgt_mask, src_padding_mask, tgt_padding_mask, memory_key_padding_mask):
    src_emb = self.positional_encoding(self.src_tok_emb(src))
    tgt_emb = self.positional_encoding(self.tgt_tok_emb(trg))

    memory = src_emb
    output = tgt_emb

    #인코더->디코더 순으로 레이어들을 순차적으로 통과
    for i in range(self.num_layers):
        #인코더
        memory = self.enc_layers[i](memory, src_mask, src_padding_mask)

        #디코더
        output = self.dec_layers[i](output, memory, tgt_mask, None, tgt_padding_mask, memory_key_padding_mask)
```

Teacher Forcing 추가

```
def forward(self, src, trg, src_mask, tgt_mask, src_padding_mask, tgt_padding_mask, memory_key_padding_mask, teacher_forcing_ratio=0.5):
    src_emb = self.positional_encoding(self.src_tok_emb(src))
    tgt_emb = self.positional_encoding(self.tgt_tok_emb(trg))

    memory = src_emb
    output = tgt_emb

    for i in range(self.num_layers):
        memory = self.enc_layers[i](memory, src_mask, src_padding_mask)

        use_teacher_forcing = random.random() < teacher_forcing_ratio

        #인코더->디코더
        if use_teacher_forcing:
            #Teacher Forcing
            output = self.dec_layers[i](tgt_emb, memory, tgt_mask, None, tgt_padding_mask, memory_key_padding_mask)
        else:
            output = self.dec_layers[i](output, memory, tgt_mask, None, tgt_padding_mask, memory_key_padding_mask)
    return self.generator(output)
```

# 학습결과




```
EMB_SIZE = 512#512
NHEAD = 4#8
FFN_HID_DIM = 1024#2048
BATCH_SIZE = 8
NUM_ENCODER_LAYERS = 6 #6
```

Evaluating on Test dataset

| | Base-Transformer | P-Transformer | P-Transformer with Teacher Forcing |
|---|---|---|---|
| loss | 2.4953 | **2.056** | 2.5673 |
| BLEU-4 | 0.1567 | **0.209** | 0.1584 |

# 결론

Evaluating on Test dataset

| | Base-Transformer | P-Transformer | P-Transformer with Teacher Forcing |
|---|---|---|---|
| loss | 2.4953 | **2.056** | 2.5673 |
| BLEU-4 | 0.1567 | **0.209** | 0.1584 |

1. p-Transformer모델은 더 많은 상호작용과, 피드백이 지속적으로 이뤄져서 학습에 더 유리. 이에 가장 높은 성능을 보임

2. Teacher Forcing이 적용된 p-Transformer 모델은 훈련시점에는 빠르고 정확히 학습되지만, 예측 값에 의존하므로 추론 성능이 상대적으로 떨어진 것으로 추측됨.(추론단계에서 성능저하)

p-Transformer 모델은 인코더와 디코더가 지속적으로 상호작용하므로, 보다 즉각적으로 정보를 받아 이용할 수 있는 것으로 생각됨. 이에 복잡한 상호작용이 요구되는 QA 모델 등에서 사용되지 않을까 생각합니다.