5 - 6주차 Review

5주차 : Text Classification

6주차 : Transformer 공부 및 구현

2024.08.23

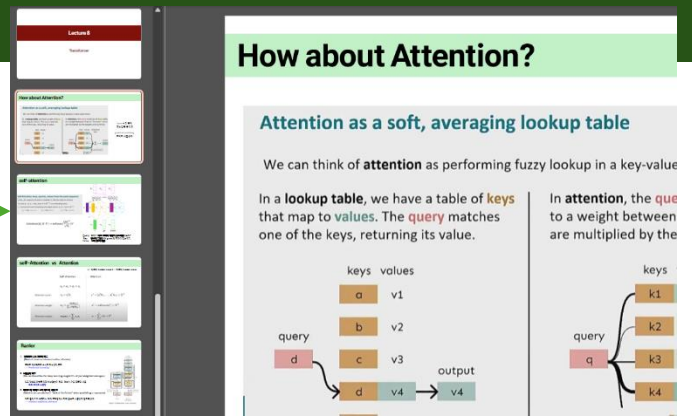
유하영

주차 목표

5주차: Text classification 구현

- 목적: Image Classification에서 벗어나서 Text에 대해서도 Classification 진행
- 내용:
  - IMDB 데이터셋을 Custom Dataset을 활용하여 Load
  - 모델은 3가지로 진행
    1) RNN 계열 모델 (LSTM or GRU) [선택]
    2) Transformer Encoder (직접 구현 or 온라인에서 가져와서 사용) [권장]
    3) Pre-trained Language Model (예; BERT, RoBERTa, BART 등) [권장]
  - Test 데이터셋에 대한 정확도 80% 목표
  - 주의사항: torchtext 활용은 권장되지 않습니다. 최대한 torchtext를 쓰지 않고 코딩 해주세요

3주차 발표



6주차: Transformer 공부 및 구현

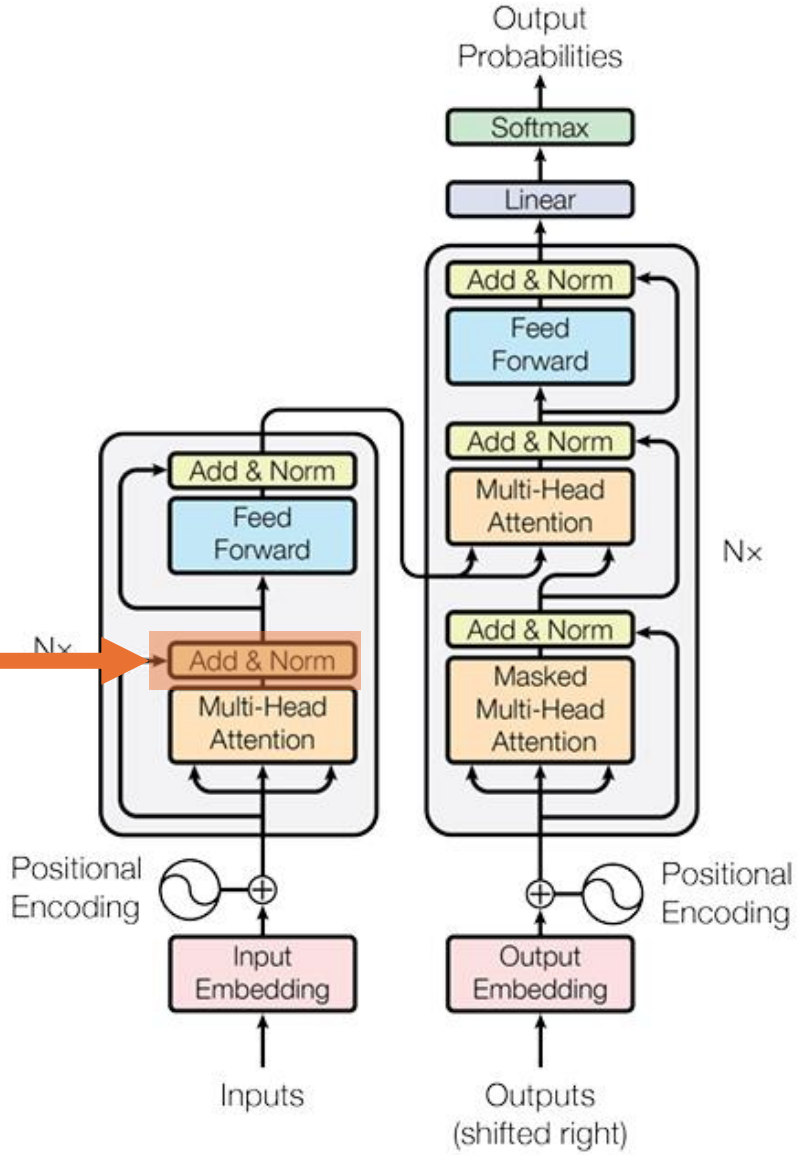
- 목적: Transformer를 적극 활용할 수 있도록 공부하고 코딩을 준비
- 내용:
  - Attention is all you need 논문 세미나
  - (선택) Transformer 직접 구현 or 온라인 코드에 대한 코드 리뷰
    → torch.nn.Transformer 모델을 사용하지 않고 torch.nn에서 주어지는 함수들로만 Transformer 모델 코딩

https://nlp.seas.harvard.edu/2018/04/03/attention.html

Encoder

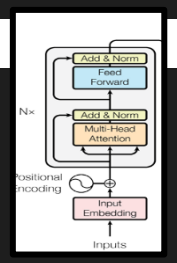
```python
class LayerNorm(nn.Module):
    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))#scaling
        self.b_2 = nn.Parameter(torch.zeros(features))#shifting
        self.eps = eps#표준편차가 0이 되는 것 방지

    def forward(self, x):
        mean =x.mean(-1, keepdim=True)#마지막 차원에 대한 평균 계산
        std = x.std(-1, keepdim=True)#마지막 차원에 대한 표준편차 계산
        return self.a_2*(x - mean) / (std + self.eps) + self.b_2#Normalization 과정
```

```python
class SublayerConnection(nn.Module):#Residual Connection
    def __init__(self, size, dropout):
        super(SublayerConnection, self).__init__()
        self.norm=LayerNorm(size)#Normalization
        self.dropout=nn.Dropout(dropout)

    def forward(self, x, sublayer):
        return x +self.dropout(sublayer(self.norm(x)))#Residual Connection
```



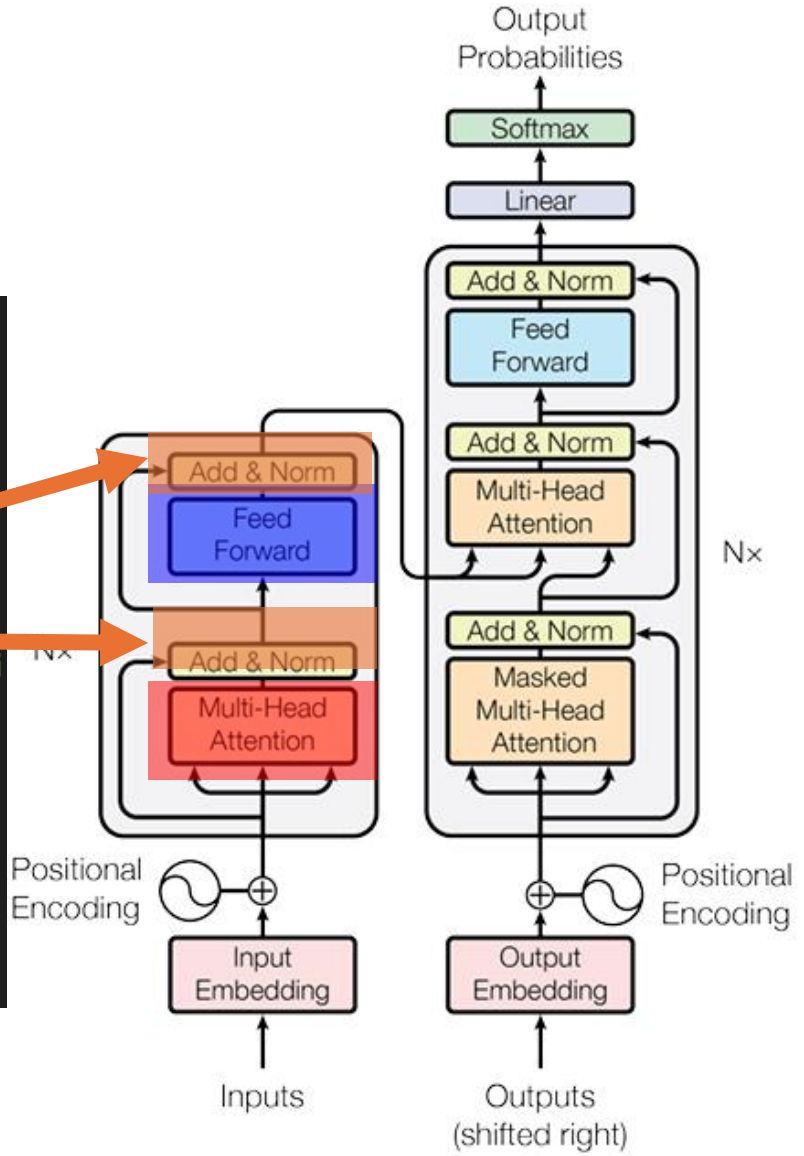
Encoder

```python
# N개의 계층 복사
def clones(module, N):
  #각 계층은 독립적으로 작동, 동일한 구조
  #ModuleList - nn.Modulelist 안에 리스트로 저장되며, 모든 모듈의 파라미터 자동 추적
  return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
```

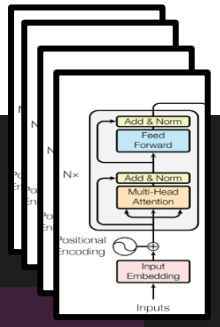


x 1

```python
#encdoer 레이어 정의
class EncoderLayer(nn.Module):
    def __init__(self, size, self_attn, feed_forward, dropout):
        super(EncoderLayer, self).__init__()
        self.self_attn = self_attn#self-attention
        self.feed_forward=feed_forward#FFNN

        #clone을 사용하여 동일한 SublayerConnection을 2개 생성
        self.sublayer = clones(SublayerConnection(size, dropout), 2)#Residual Connection,Normalizati
        self.size = size

    def forward(self, x, mask):
      #첫 번째 서브레이어 : self-attention
      x = self.sublayer[0](x, lambda x: self.self_attn(x,x,x,mask))
      #두 번째 서브레이어 : FFNN
      return self.sublayer[1](x, self.feed_forward)
```



Encoder

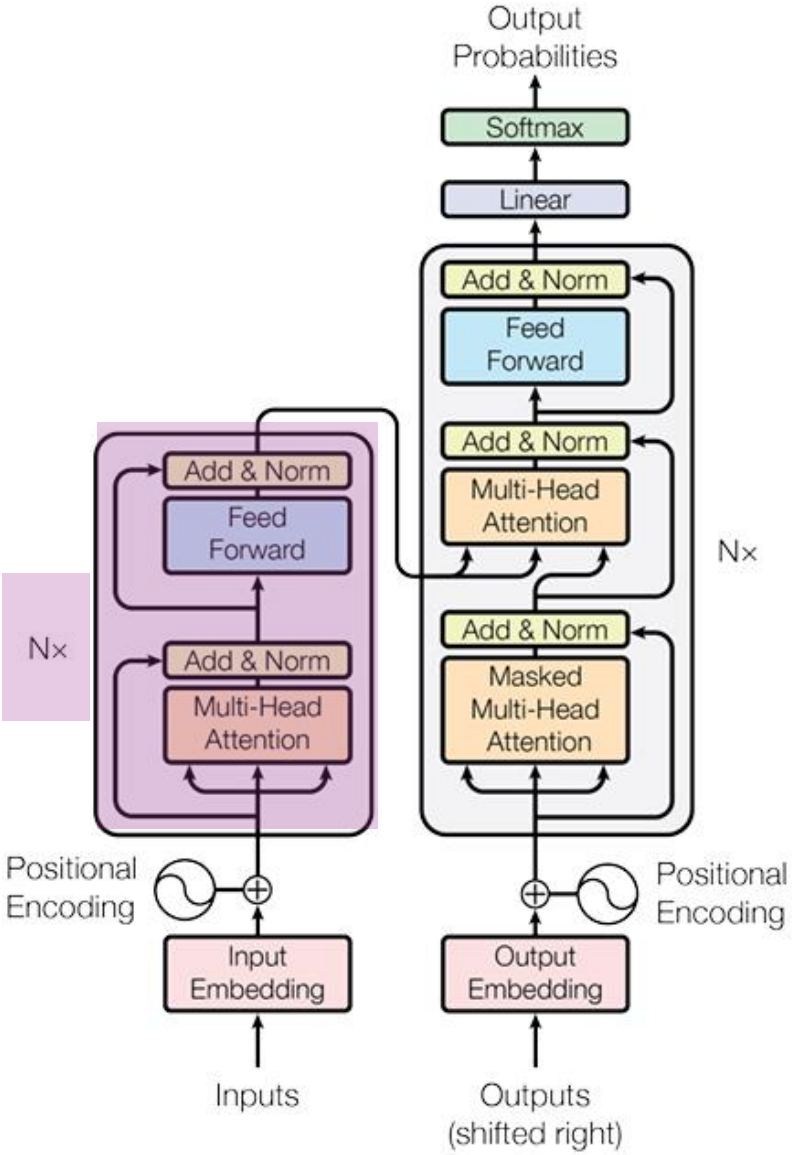


x N

```python
#Encoder n layer
class Encoder(nn.Module):
  def __init__(self, layer, N):
    super(Encoder, self).__init__()
    self.layers = clones(layer, N)#N layers
    self.norm = LayerNorm(layer.size)#LayerNormalization

  #masked
  def forward(self,x,mask):
    for layer in self.layers:
      x = layer(x,mask)

    return self.norm
```

Padding Mask
패딩된 토큰들에 대해서, 해당 위치에서의 계산 무시(불필요한 정보습득 방지)

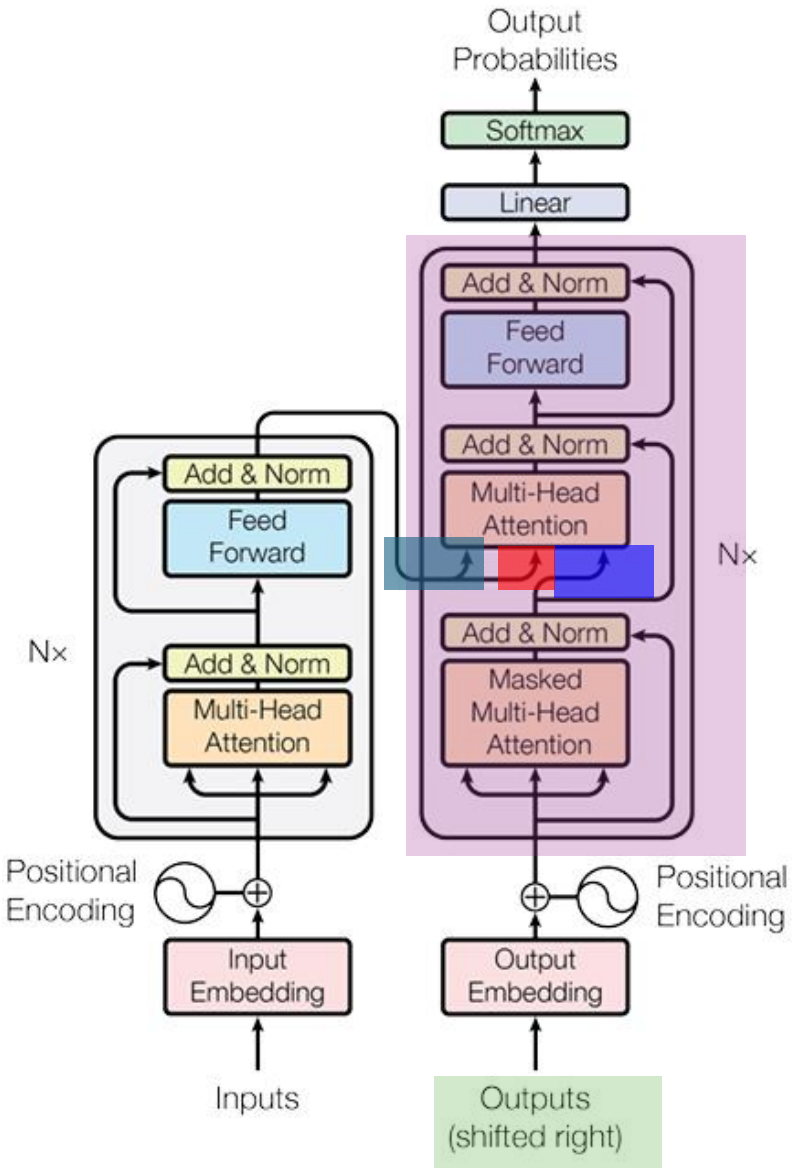
```python
model = EncoderDecoder(
    Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),#인코더
```



Decoder

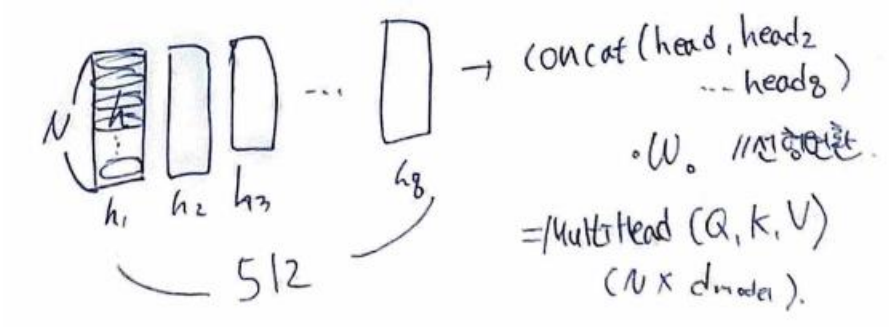
```python
class Decoder(nn.Module):
    def __init__(self,layer,N):
        super(Decoder,self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, memory, src_mask, tgt_mask):
        for layer in self.layers:
            x=layer(x,memory, src_mask,tgt_mask)
        return self.norm(x)
```

```python
class DecoderLayer(nn.Module):
    def __init__(self, size, self_attn, src_attn, feed_forward, dropout):
        super(DecoderLayer, self).__init__()
        self.size = size
        self.self_attn = self_attn
        self.src_attn=src_attn
        self.feed_forward =feed_forward
        self.sublayer=clones(SublayerConnection(size, dropout), 3)

    def forward(self, x, memory, src_mask, tgt_mask):
        m = memory
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, tgt_mask))
        x = self.sublayer[1](x, lambda x: self.src_attn(x, m, m, src_mask))
        return self.sublayer[2](x, self.feed_forward)
```



self-attention

multi-head Attention

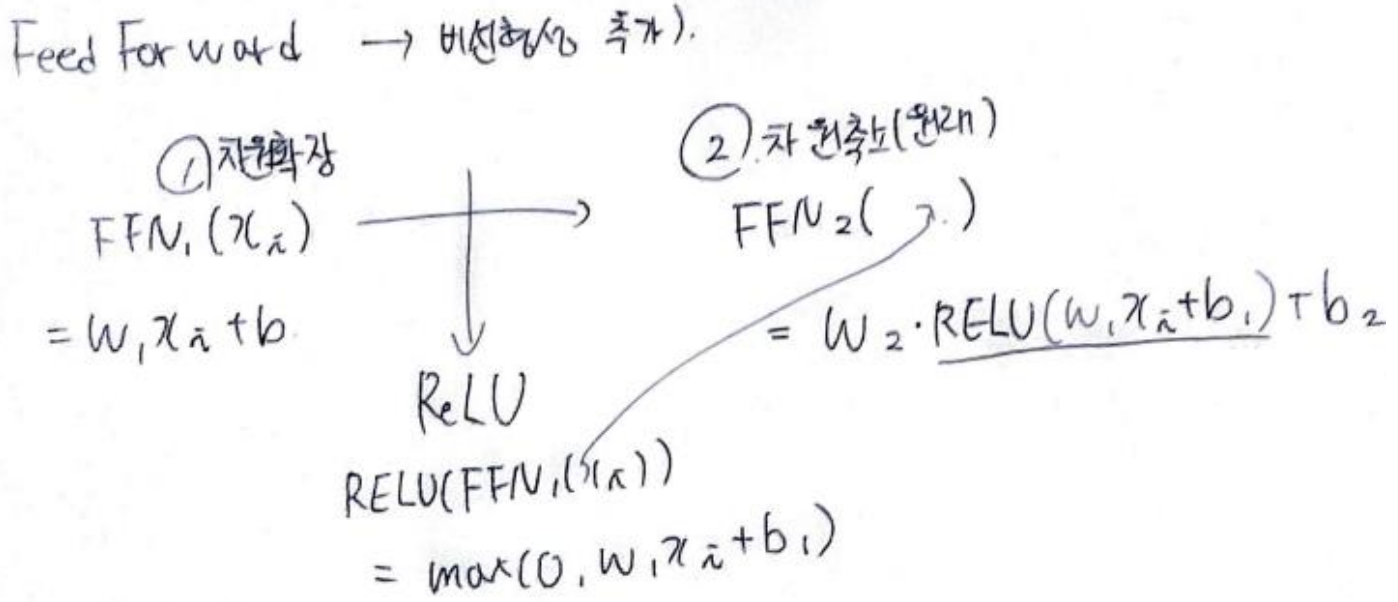



```python
#Attention
def attention(query, key, value, mask=None, dropout=None):
    d_k = query.size(-1)#쿼리의 마지막 차원 크기
    scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(d_k)#scaled-dot-product
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)#mask 위치에는 큰 음수값 넣어, softmax에서 무시도
    p_attn = F.softmax(scores, dim = -1)#확률분포 계산
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn#어텐션 스코어, 가중치 return
```

```python
class MultiHeadedAttention(nn.Module):
    def __init__(self, h, d_model, dropout=0.1):
        super(MultiHeadedAttention, self).__init__()
        assert d_model % h == 0#h로 나누어 떨어지는 지 확인

        self.d_k = d_model // h #각 head의 차원
        self.h = h
        self.linears = clones(nn.Linear(d_model, d_model), 4)#q,k,v,최종결과
        self.attn = None#어텐션 가중치 저장
        self.dropout = nn.Dropout(p=dropout)

    def forward(self, query, key, value, mask=None):
        if mask is not None:
            mask = mask.unsqueeze(1)
        nbatches = query.size(0)#배치크기 추출

        #1. 선형변환, 차원변경
        #[batch_size, seq_len, h, d_k]에서 [batch_size, h, seq_len, d_k]로 변환
        query, key, value = \
            [l(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
             for l, x in zip(self.linears, (query, key, value))]#선형변환]

        #2. 어텐션 적용
        x, self.attn = attention(query, key, value, mask=mask,
                                 dropout=self.dropout)

        #head concat, linear하게 변환
        x = x.transpose(1, 2).contiguous() \
             .view(nbatches, -1, self.h * self.d_k)
        return self.linears[-1](x)#마지막 레이어 반환(최종 결과)

        #차원 : [batch_size, seq_len, d_model]
```

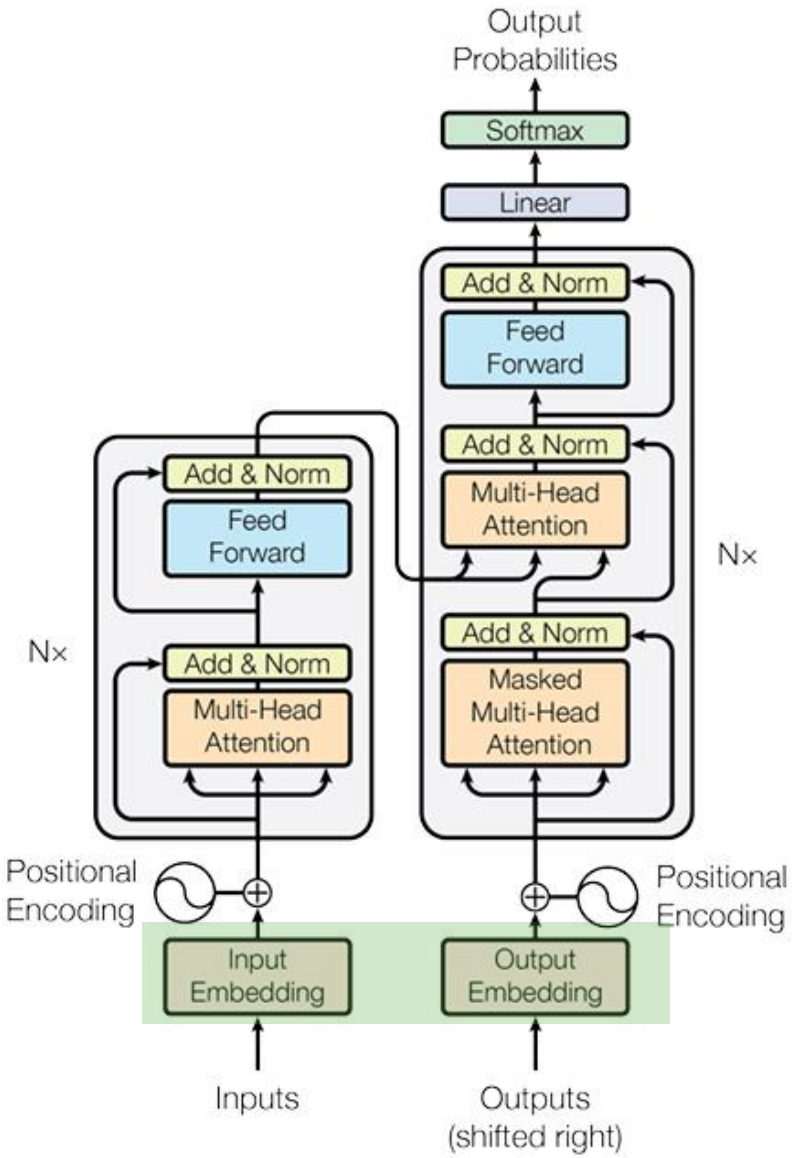
Feed-Forward Networks

```python
#Position-wise Feed-Forward Networks
class PositionwiseFeedForward(nn.Module):
    def __init__(self, d_model, d_ff, dropout=0.1):
        super(PositionwiseFeedForward, self).__init__()
        self.w_1 = nn.Linear(d_model, d_ff)
        self.w_2 = nn.Linear(d_ff, d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.w_2(self.dropout(F.relu(self.w_1(x))))
```



Embedding

```python
#word embedding
class Embeddings(nn.Module):
    def __init__(self, d_model, vocab):
        super(Embeddings, self).__init__()
        self.lut = nn.Embedding(vocab, d_model)#임베딩 벡터로 변환(어휘 사이즈,d_model)
        self.d_model = d_model

    def forward(self, x):
        return self.lut(x) * math.sqrt(self.d_model)
        #d_model의 제곱근으로 스케일링하여, 초기 학습 단계에서 너무 작은 값이 되는 것을 방지
```

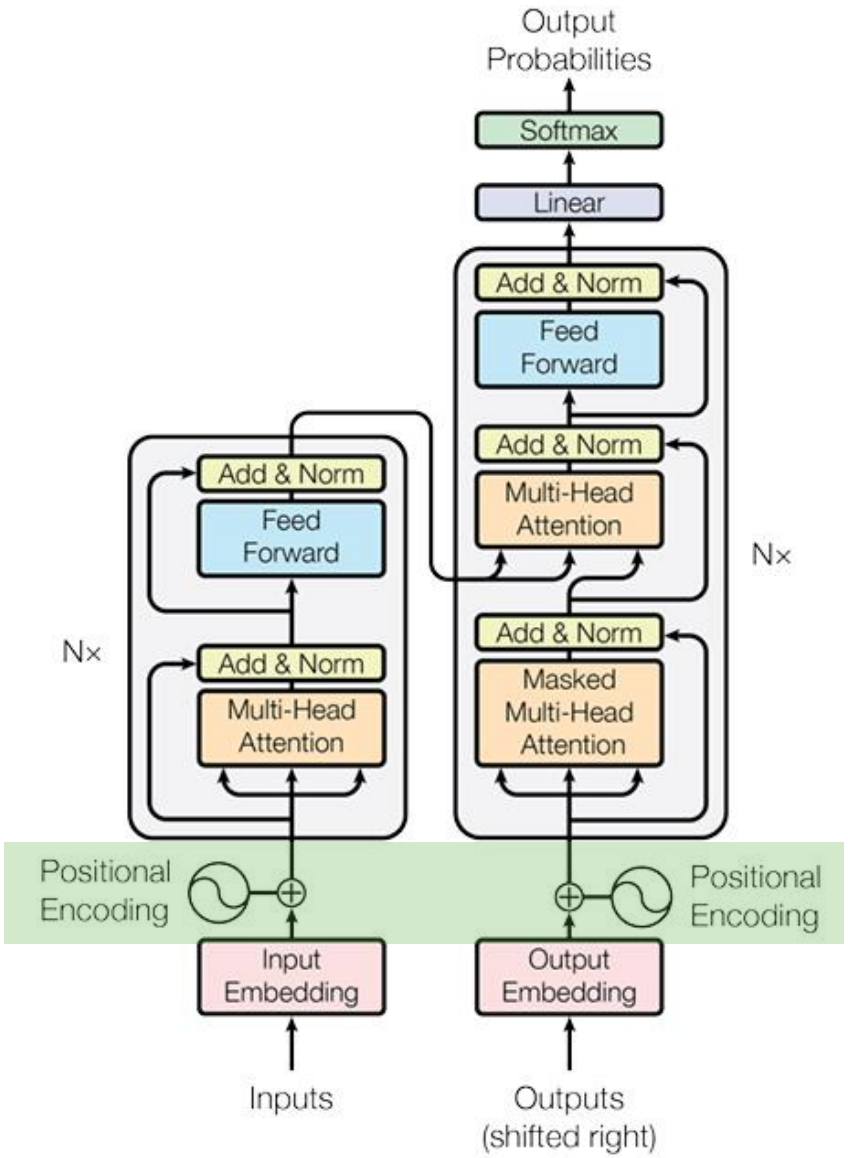


Positional Encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

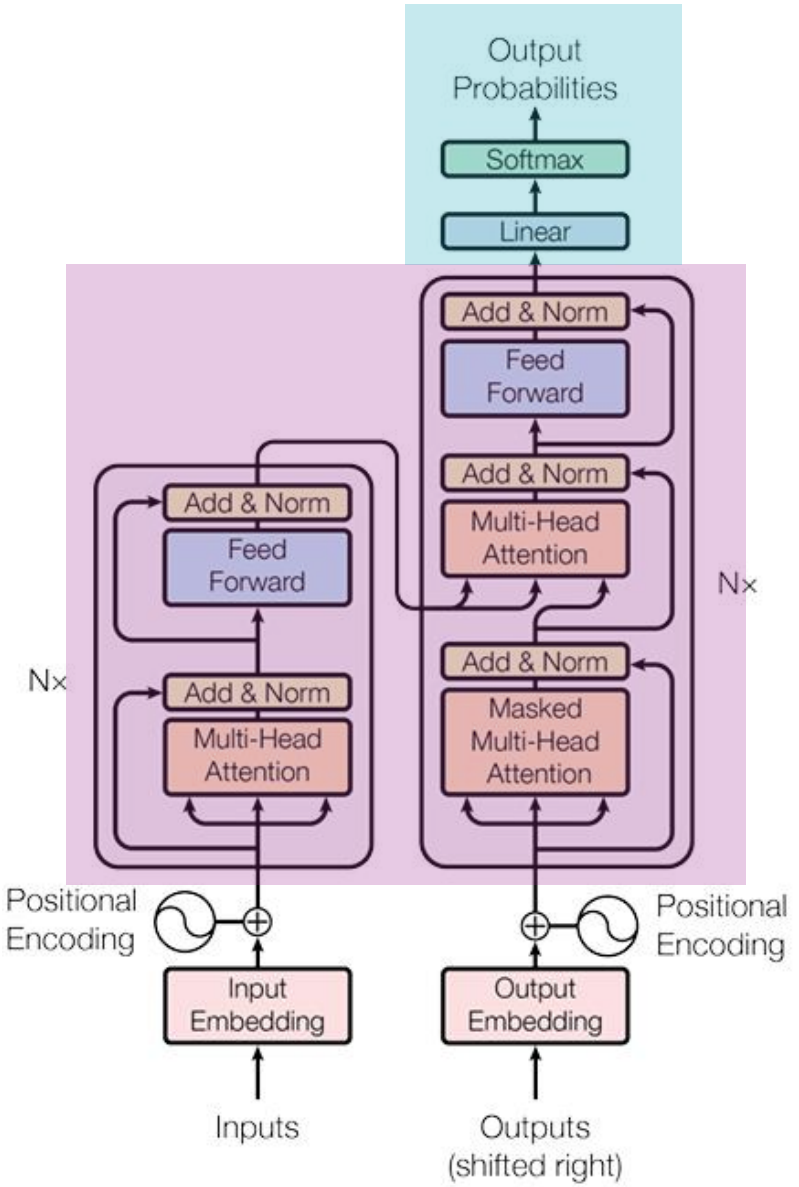
```python
#Positional Encoding
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)
        #max_len: 입력시퀀스의 최대길이

        #PE 계산
        pe = torch.zeros(max_len, d_model)#0으로 초기화
        position = torch.arange(0, max_len).unsqueeze(1)#차원 텐서
        div_term = torch.exp(torch.arange(0, d_model, 2) *
                             -(math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)#배치차원 추가 [1, max_len, d_model]
        self.register_buffer('pe', pe)#모델의 버퍼로 등록-> 파라미터로 지정X

    def forward(self, x):
        x = x + Variable(self.pe[:, :x.size(1)],
                         requires_grad=False)
        return self.dropout(x)
        #[batch_size, seq_len, d_model]
```



Encoder-Decoder

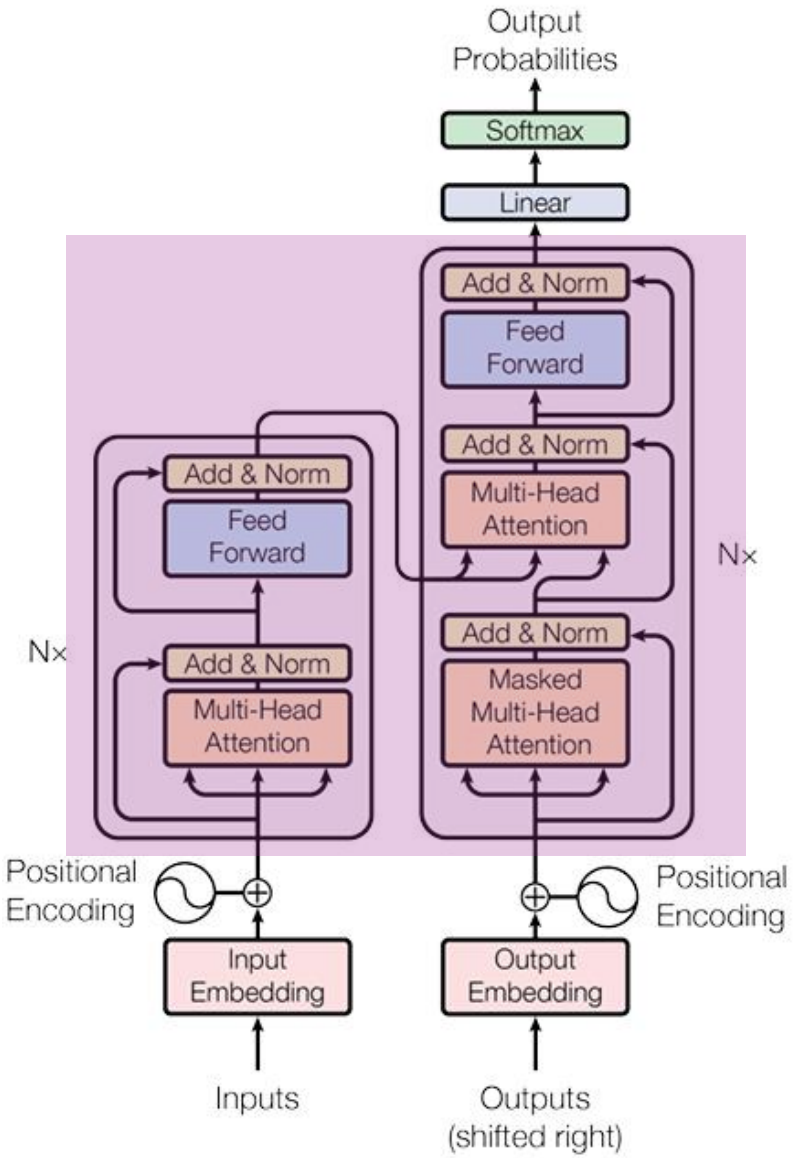
```python
#Transformer model 정의(Encoder-Decoder)
class EncoderDecoder(nn.Module):
  def __init__(self, encoder, decoder, src_embed, tgt_embed, generator):
    super(EncoderDecoder, self).__init__()
    self.encoder = encoder
    self.decoder = decoder
    self.src_embed = src_embed
    self.tgt_embed = tgt_embed
    self.generator = generator

  def forward(self, src, tgt, src_mask, tgt_mask):
    return self.decode(self.encode(src,src_mask),src_mask,tgt,tgt_mask)
  def encode(self, src, src_mask):
    return self.encoder(self.src_embed(src),src_mask)
  def decode(self, memory, src_mask, tgt, tgt_mask):
    return self.decoder(self.tgt_embed(tgt),memory,src_mask,tgt_mask)
```

```python
#Decoder의 마지막 부분/단어 확률 분포 변환 (Linear->softmax)
class Generator(nn.Module):
  def __init__(self, d_model, vocab):
    super(Generator, self).__init__()
    self.proj = nn.Linear(d_model, vocab)
  def forward(self, x):
    return F.log_softmax(self.proj(x), dim=-1) #동일표현 torch.nn.functional.log_softmax
```



최종 모델

```python
#최종 모델
def make_model(src_vocab, tgt_vocab, N=6,
               d_model=512, d_ff=2048, h=8, dropout=0.1):
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)#멀티헤드 셀프어텐션
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)#포지셔널 피드포워드 신경망
    position = PositionalEncoding(d_model, dropout)#입력 데이터에 포지션 정보 추가
    model = EncoderDecoder(
        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),#인코더
        Decoder(DecoderLayer(d_model, c(attn), c(attn),
                             c(ff), dropout), N),#디코더
        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab))#최종적으로 출력된 예측 어휘

    for p in model.parameters():
        if p.dim() > 1:
            nn.init.xavier_uniform(p)
    return model
```



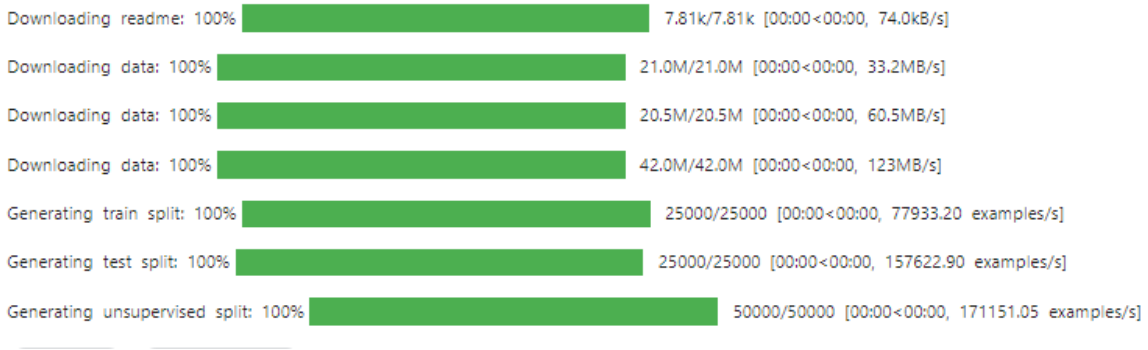
실습

1) Transformer - IMDB

2) pretrained BERT - IMDB

IMDB 데이터셋 load

```python
from datasets import load_dataset
import torch

# 데이터셋 로드
dataset2 = load_dataset('imdb')
```



Train 첫 번째 샘플

{'text': 'I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was first released in 1967. I also heard that at first it was seized by U.S. customs if it ever tried to enter this country, therefore being a fan of films considered "controversial" I really had to see this for myself.

The plot is centered around a young Swedish drama student named Lena who wants to learn everything she can about life. In particular she wants to focus her attentions to making some sort of documentary on what the average Swede thought about certain political issues such as the Vietnam War and race issues in the United States. In between asking politicians and ordinary denizens of Stockholm about their opinions on politics, she has sex with her drama teacher, classmates, and married men.

What kills me about I AM CURIOUS-YELLOW is that 40 years ago, this was considered pornographic. Really, the sex and nudity scenes are few and far between, even then it\'s not shot like some cheaply made porno. While my countrymen mind find it shocking, in reality sex and nudity are a major staple in Swedish cinema. Even Ingmar Bergman, arguably their answer to good old boy John Ford, had sex scenes in his films.

I do commend the filmmakers for the fact that any sex shown in the film is shown for artistic purposes rather than just to shock people and make money to be shown in pornographic theaters in America. I AM CURIOUS-YELLOW is a good film for anyone wanting to study the meat and potatoes (no pun intended) of Swedish cinema. But really, this film doesn\'t have much of a plot.', 'label': 0}

Test 첫 번째 샘플

{'text': 'I love sci-fi and am willing to put up with a lot. Sci-fi movies/TV are usually underfunded, under-appreciated and misunderstood. I tried to like this, I really did, but it is to good TV sci-fi as Babylon 5 is to Star Trek (the original). Silly prosthetics, cheap cardboard sets, stilted dialogues, CG that doesn\'t match the background, and painfully one-dimensional characters cannot be overcome with a \'sci-fi\' setting. (I\'m sure there are those of you out there who think Babylon 5 is good sci-fi TV. It\'s not. It\'s clichéd and uninspiring.) While US viewers might like emotion and character development, sci-fi is a genre that does not take itself seriously (cf. Star Trek). It may treat important issues, yet not as a serious philosophy. It\'s really difficult to care about the characters here as they are not simply foolish, just missing a spark of life. Their actions and reactions are wooden and predictable, often painful to watch. The makers of Earth KNOW it\'s rubbish as they have to always say "Gene Roddenberry\'s Earth..." otherwise people would not continue watching. Roddenberry\'s ashes must be turning in their orbit as this dull, cheap, poorly edited (watching it without advert breaks really brings this home) trudging Trabant of a show lumbers into space. Spoiler. So, kill off a main character. And then bring him back as another actor. Jeeez! Dallas all over again.', 'label': 0}

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label'],
        num_rows: 50000
    })
})
```

Transformer - IMDB

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments
from transformers import Trainer, TrainingArguments

# 토크나이저 로드
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')#AutoTokenizer 사용
# 데이터셋 토큰화
def tokenize_function(examples):
    return tokenizer(examples['text'], padding="max_length", truncation=True)
tokenized_datasets = dataset2.map(tokenize_function, batched=True)
```

```
import torch.nn as nn

d_model =512
h =8 #heads
N = 6 #layers
d_ff = 2048
dropout = 0.1
vocab_size = 30522

self_attn = MultiHeadedAttention(h,d_model)
feed_forward = PositionwiseFeedForward(d_model,d_ff,dropout)
encoder_layer=EncoderLayer(d_model, self_attn, feed_forward, dropout)
encoder=Encoder(encoder_layer, N)

model = TransformerClassifier(encoder,d_model,vocab_size,num_labels=2)
```

Transformer - IMDB

Weights & Biases

```
# W&B 초기화
import wandb
wandb.login(key="b
```

```
#학습 설정 및 Trainer 초기화
from transformers import Trainer, TrainingArguments
from sklearn.metrics import accuracy_score
from torch.nn.functional import cross_entropy
import numpy as np
import torch

training_args = TrainingArguments(
    output_dir='./results',
    eval_strategy='epoch', #매 epoch마다 평가
    logging_strategy='steps',
    logging_steps=10,
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=20,
    weight_decay=0.01, #L2 정규화(가중치 감소)
    report_to='wandb', #W&B 사용
    run_name='Transformer-IMDB-4',
    remove_unused_columns=False
)

def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=1)
    accuracy = accuracy_score(p.label_ids, preds)

    logits = torch.tensor(p.predictions)
    labels = torch.tensor(p.label_ids, dtype=torch.long)
    loss = cross_entropy(logits, labels).item()

    return {'accuracy':accuracy,'loss':loss}

#Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets['train'],
    eval_dataset=tokenized_datasets['test'],
    compute_metrics=compute_metrics,
)
```

```
trainer.train()
```

Changes to your `wandb` environment variables will be ignored because your `wandb` session has already start
docs.

```
wandb: ERROR Dropped streaming file chunk (see wandb/debug-internal.log)
wandb: ERROR Dropped streaming file chunk (see wandb/debug-internal.log)
wandb: ERROR Dropped streaming file chunk (see wandb/debug-internal.log)
wandb: ERROR Dropped streaming file chunk (see wandb/debug-internal.log)
```

Tracking run with wandb version 0.17.7

Run data is saved locally in `/kaggle/working/wandb/run-20240822_215955-6a9uddqi`

Syncing run **Transformer-IMDB-4** to Weights & Biases (docs)

View project at https://wandb.ai/bluebarry37-r/huggingface

View run at https://wandb.ai/bluebarry37-r/huggingface/runs/6a9uddqi

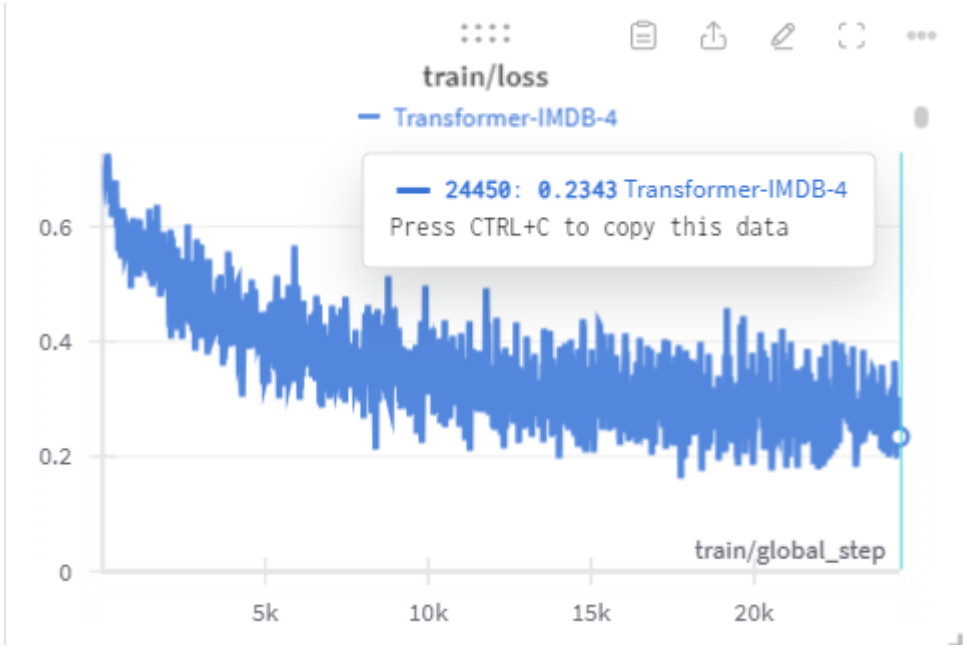
```
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWa
args...)` instead.
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_e
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning:
ze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
```

[24501/31260 2:20:40 < 38:48, 2.90 it/s, Epoch 15.67/20]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.586800	0.533097	0.729920
2	0.502900	0.485402	0.768760
3	0.391400	0.415459	0.811960
4	0.362200	0.398043	0.825280
5	0.330400	0.396451	0.833600
6	0.306500	0.394168	0.836040
7	0.360500	0.374970	0.844440

Transformer - IMDB

Epoch	Training Loss	Validation Loss	Accuracy
1	0.586800	0.533097	0.729920
2	0.502900	0.485402	0.768760
3	0.391400	0.415459	0.811960
4	0.362200	0.398043	0.825280
5	0.330400	0.396451	0.833600
6	0.306500	0.394168	0.836040
7	0.360500	0.374970	0.844440
8	0.348300	0.380493	0.847720
9	0.304500	0.373681	0.849400
10	0.279500	0.362329	0.853760
11	0.349700	0.364457	0.856680
12	0.271700	0.366635	0.858800
13	0.258900	0.363420	0.858320
14	0.244200	0.366747	0.859960
15	0.308100	0.367519	0.861640



pretrained BERT - IMDB

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
```

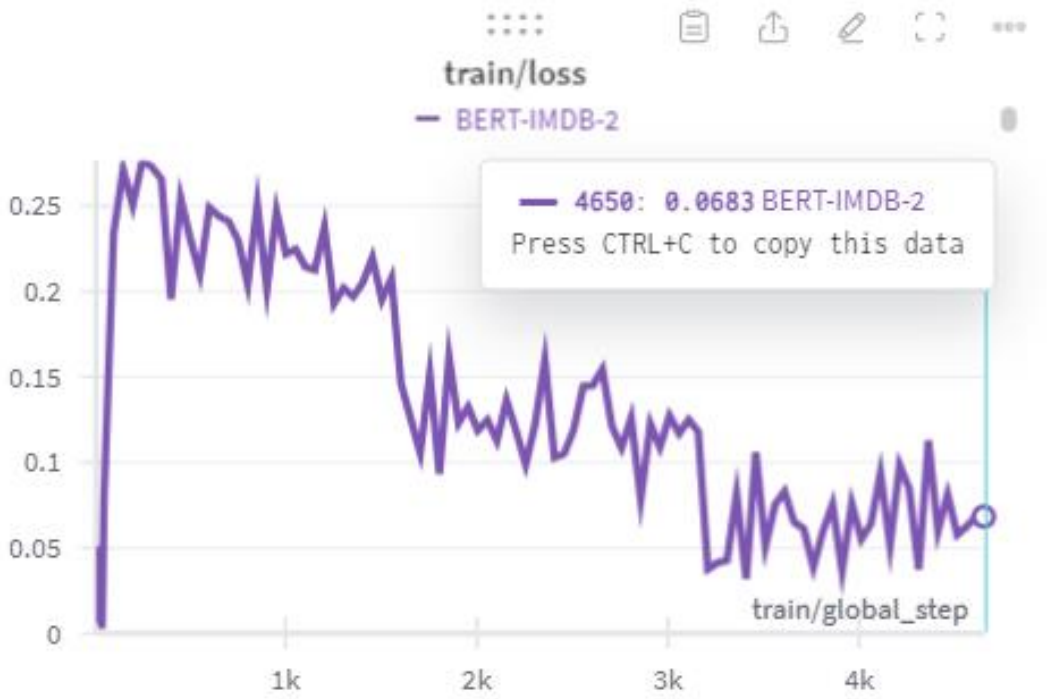
```
#데이터 전처리
def tokenize_function(examples):
    return tokenizer(examples['text'], padding='max_length', truncation=True)

tokenized_datasets = dataset.map(tokenize_function, batched=True)
tokenized_datasets = tokenized_datasets.remove_columns(['text'])
tokenized_datasets.set_format('torch')
```

```
Map: 100% 25000/25000 [03:53<00:00, 105.81 examples/s]
Map: 100% 25000/25000 [03:46<00:00, 111.20 examples/s]
Map: 100% 50000/50000 [07:48<00:00, 106.32 examples/s]
```

```
#버전 버전
import numpy as np
import torch
from sklearn.metrics import accuracy_score
from torch.nn.functional import cross_entropy
from transformers import Trainer, TrainingArguments

# 훈련 및 평가 설정
training_args = TrainingArguments(
    output_dir='./results',
    eval_strategy='epoch',#매 epoch마다 평가
    logging_strategy='steps',
    logging_steps=10,
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,#L2 정규화(가중치 감소)
    report_to='wandb',# W&B
    run_name='BERT-IMDB-3'
)

def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=1)
    accuracy = accuracy_score(p.label_ids, preds)

    logits = torch.tensor(p.predictions)
    labels = torch.tensor(p.label_ids, dtype=torch.long)
    loss = cross_entropy(logits, labels).item()

    return {'accuracy': accuracy, 'loss': loss}

#Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets['train'],
    eval_dataset=tokenized_datasets['test'],
    compute_metrics=compute_metrics,
)
```

Epoch	Training Loss	Validation Loss	Accuracy
1	0.207900	0.184224	0.930320
2	0.125100	0.228064	0.937080
3	0.068300	0.274137	0.939840

pretrained BERT - IMDB



Test Accuracy: 0.93984
Test Loss: 0.2741374373435974

```
#부정 예상리뷰
test_input = "This movie was just way too overrated. The fighting was not professional and in slow motion. I was expecting more from a 200 milli

sentiment_predict(test_input)
```

99.90% 확률로 부정 리뷰입니다.

```
#긍정 예상리뷰
test_input = " I was lucky enough to be included in the group to see the advanced screening in Melbourne on the 15th of April, 2012. And, firstl
Now, the film... how can I even begin to explain how I feel about this film? It is, as the title of this review says a 'comic book triumph'. I w
Seeing Joss Whedon's direction and envisioning of the film come to life on the big screen is perfect. The script is amazingly detailed and laced

sentiment_predict(test_input)
```

99.87% 확률로 긍정 리뷰입니다.

https://wikidocs.net/64904