허깅페이스 트랜스포머 모델 사용하기

허깅페이스

Transformers 라이브러리

https://huggingface.co/

이 슬라이드에서 사용한 서체 :
- Open Sans(https://ko.cooltext.com/Download-Font-Open+Sans)
- KoPubWorld돋움체(http://www.kopus.org/biz/electronic/font.aspx)

1장. 허깅페이스와 트랜스포머 알아보기

허깅페이스 트랜스포머를 이용한 자연어처리

허깅페이스(Hugging Face)

자연어 처리와 관련된 다양한 기술과 리소스를 제공하는 회사 및 온라인 플랫폼

- https://huggingface.co/

Transformers 라이브러리

- **Transformer 모델들**을 쉽게 사용할 수 있도록 도와주는 오픈소스 라이브러리리
- 최신의 딥 러닝 기반 자연어 처리 모델을 쉽게 사용할 수 있도록 지원함
- BERT, GPT, RoBERTa 등 다양한 사전 훈련된 모델들을 포함하고 있으며, 다양한 NLP 작업에 대해 미세 조정(Fine-tuning)이 가능

> 허깅페이스는 커뮤니티와의 협력을 강조하며, 다양한 NLP 기술 발전에 기여하는 것을 목표로 하고 있습니다. 이들의 라이브러리와 도구들은 NLP 개발자들에게 효율적이고 강력한 도구를 제공하여, 복잡한 자연어 이해와 생성 문제를 해결하는 데 도움을 줍니다.

사전 훈련된 모델 호스팅

- 다양한 사전 훈련된 언어 모델을 제공하며, 이를 통해 개발자들은 새로운 NLP 애플리케이션을 빠르게 구축하고 테스트할 수 있음
- 모델을 사용할 때 필요한 인퍼런스(추론) 환경도 제공함

Datasets 라이브러리

- 다양한 공개 데이터셋을 손쉽게 접근하고 활용할 수 있는 라이브러리인 datasets를 제공
- 이 라이브러리는 데이터셋을 로드하고 전처리하는 기능을 지원하여, NLP 모델 훈련 및 평가에 유용하게 사용됨

Hub

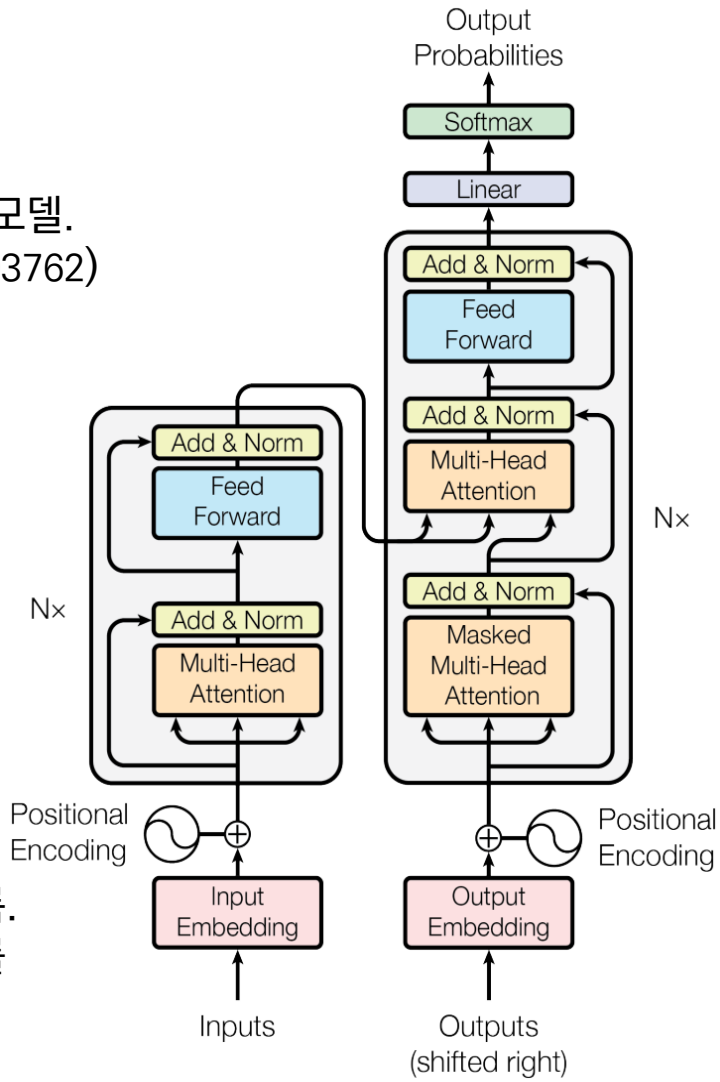
- 허깅페이스 Hub는 커뮤니티가 공유한 다양한 모델, 토크나이저, 데이터셋 등의 리소스를 검색하고 공유할 수 있는 플랫폼
- 개발자들은 Hub을 통해 다른 사람들이 공유한 리소스를 활용하여 자신의 프로젝트를 보다 효율적으로 개발할 수 있음

https://huggingface.co/welcome

Transformer

작동 원리

- GPT 모델은 'Transformer(트랜스포머)'라는 아키텍처를 기반으로 함
  - 트랜스포머 모델은 자연어 처리와 같은 순차적인 데이터를 처리하기 위해 설계된 딥러닝 모델.
  - 2017년 구글이 “Attention is all you need” 논문에서 발표한 모델(https://arxiv.org/abs/1706.03762)
- **입력 처리**: 문장을 단어 단위로 쪼개어 각 단어를 수치로 변환(임베딩)합니다. 이 변환된 숫자 벡터는 모델이 이해할 수 있는 형태로, 각 단어의 의미를 담고 있음.
- **어텐션 메커니즘 (Attention Mechanism)**: 트랜스포머의 핵심 아이디어는 어텐션 메커니즘입니다. 어텐션은 입력의 모든 단어가 서로 얼마나 중요한지를 계산하여, 중요한 단어에 더 집중할 수 있도록 합니다. 쉽게 말해, 문장 내에서 어떤 단어들이 서로 관련이 있는지를 판단하고, 그 연관성을 바탕으로 정보의 흐름을 조절함.
- **인코더와 디코더**: 트랜스포머는 인코더와 디코더로 구성.
  - **인코더**: 입력 문장을 받아 어텐션을 통해 의미 있는 벡터로 변환.
  - **디코더**: 인코더에서 얻은 벡터를 받아 새로운 문장을 생성하거나, 번역 작업 등을 수행.
- **병렬 처리**: 트랜스포머는 RNN이나 LSTM과 달리 모든 단어를 한 번에 처리할 수 있어 학습이 빠름. 이는 어텐션 메커니즘 덕분에 가능하며, 이전 단어들을 순차적으로 보지 않아도 각 단어의 중요도를 빠르게 계산할 수 있음.



허깅페이스의 Transformers

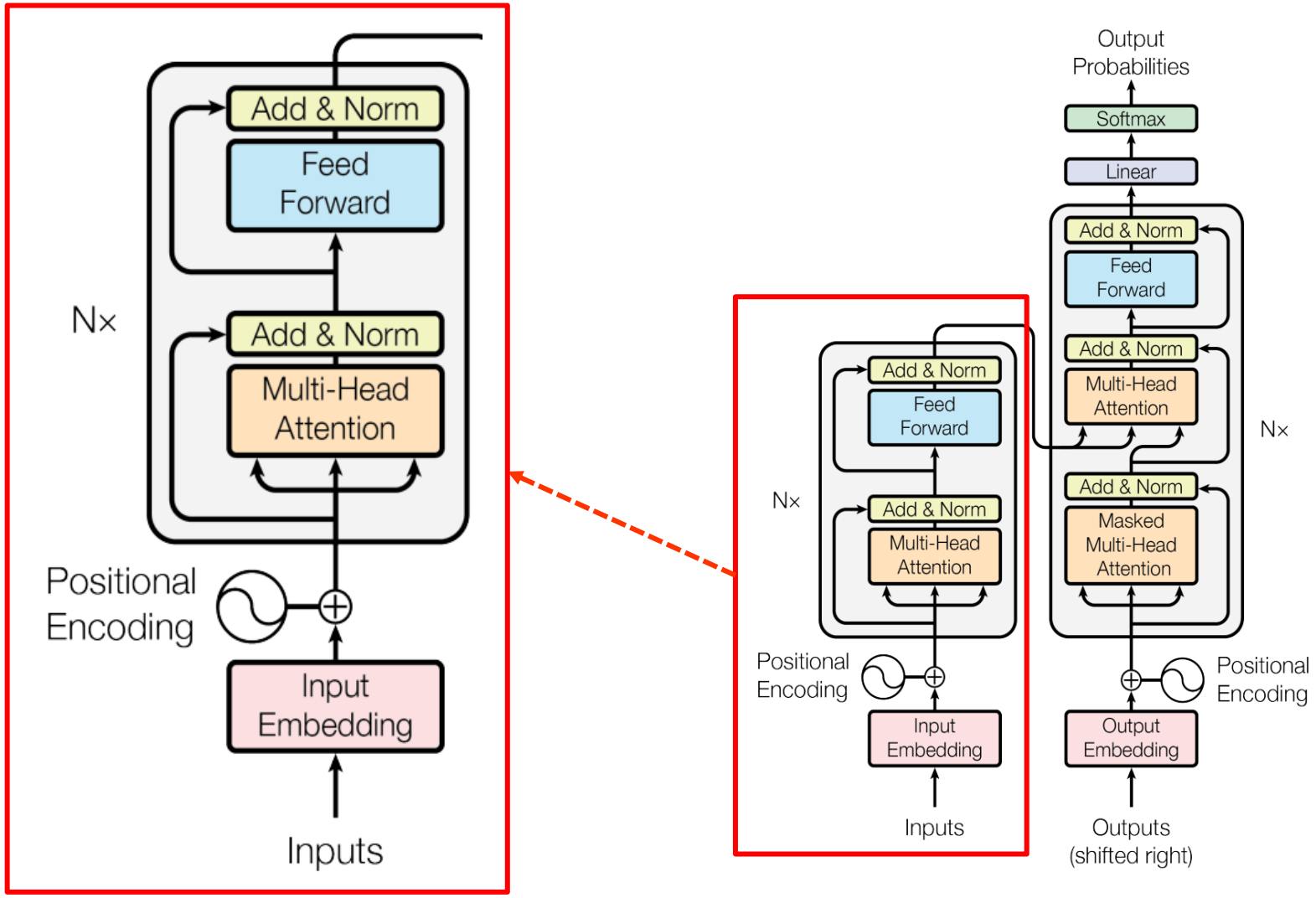
Transformers

- 자연어 처리(NLP)와 자연어 생성(NLG) 등 다양한 AI 모델을 쉽게 사용할 수 있게 해주는 오픈 소스 라이브러리
- BERT, GPT-3, T5 등 유명한 사전 학습된 트랜스포머 모델을 포함하며, 텍스트 생성, 번역, 감정 분석, 요약 등 다양한 언어 처리 작업을 지원함.

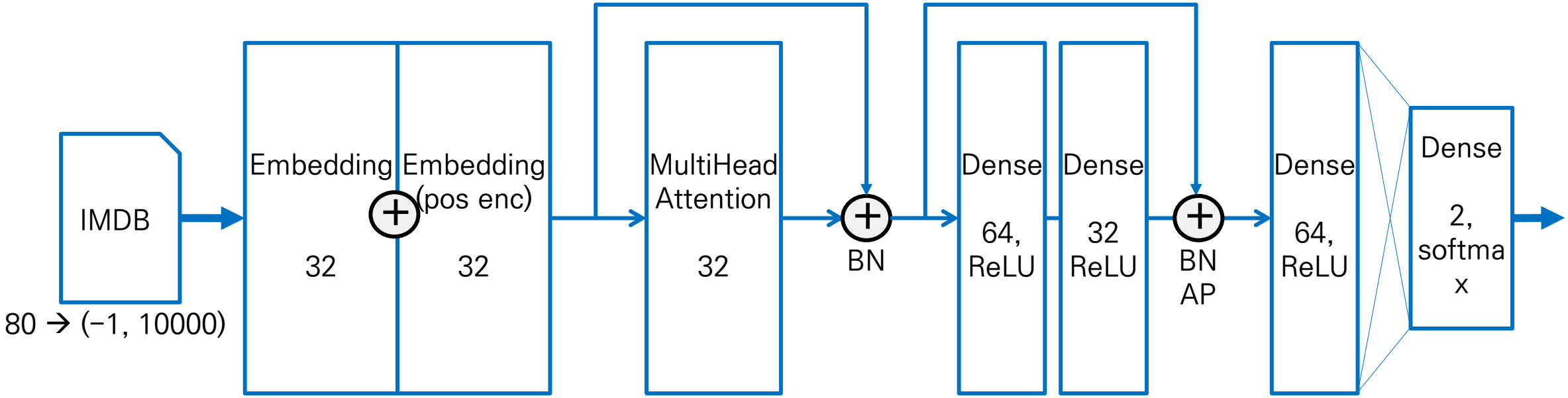
주요 기능 및 특징

- 사전 학습된 모델: 허깅페이스는 BERT, GPT, T5, RoBERTa 등 다양한 사전 학습된 모델을 제공함. 이 모델들은 Hugging Face Model Hub에서 쉽게 다운로드하여 사용할 수 있음.
- 다양한 태스크 지원: 트랜스포머 모델을 통해 감정 분석, 번역, 요약, 질의 응답, 텍스트 생성, 텍스트 분류, 개체명 인식(NER) 등 여러 NLP 작업을 수행할 수 있음.
- 사용자 친화적 인터페이스: Hugging Face 라이브러리는 Python 기반으로 직관적인 API를 제공하며, 초보자부터 전문가까지 쉽게 사용할 수 있도록 설계되었음.
- 모델 미세 조정(Fine-tuning): 사용자는 자신만의 데이터셋으로 사전 학습된 모델을 미세 조정하여 특정 작업에 최적화된 모델을 만들 수 있음
- 허브(Hugging Face Model Hub): 이곳에서 다양한 사전 학습된 모델을 다운로드하거나, 자신이 만든 모델을 업로드하고 공유할 수 있음. 연구자나 개발자가 각자의 모델을 쉽게 공유하고 활용할 수 있는 플랫폼임.
- 프레임워크 호환성: PyTorch와 TensorFlow 두 가지 주요 딥러닝 프레임워크를 모두 지원하며, 동일한 코드로 두 프레임워크에서 실행할 수 있는 높은 유연성을 제공함
- 통합 API: 허깅페이스는 API도 제공하여 로컬에서 모델을 실행하지 않고도 REST API 방식으로 자연어 처리 기능을 활용할 수 있음

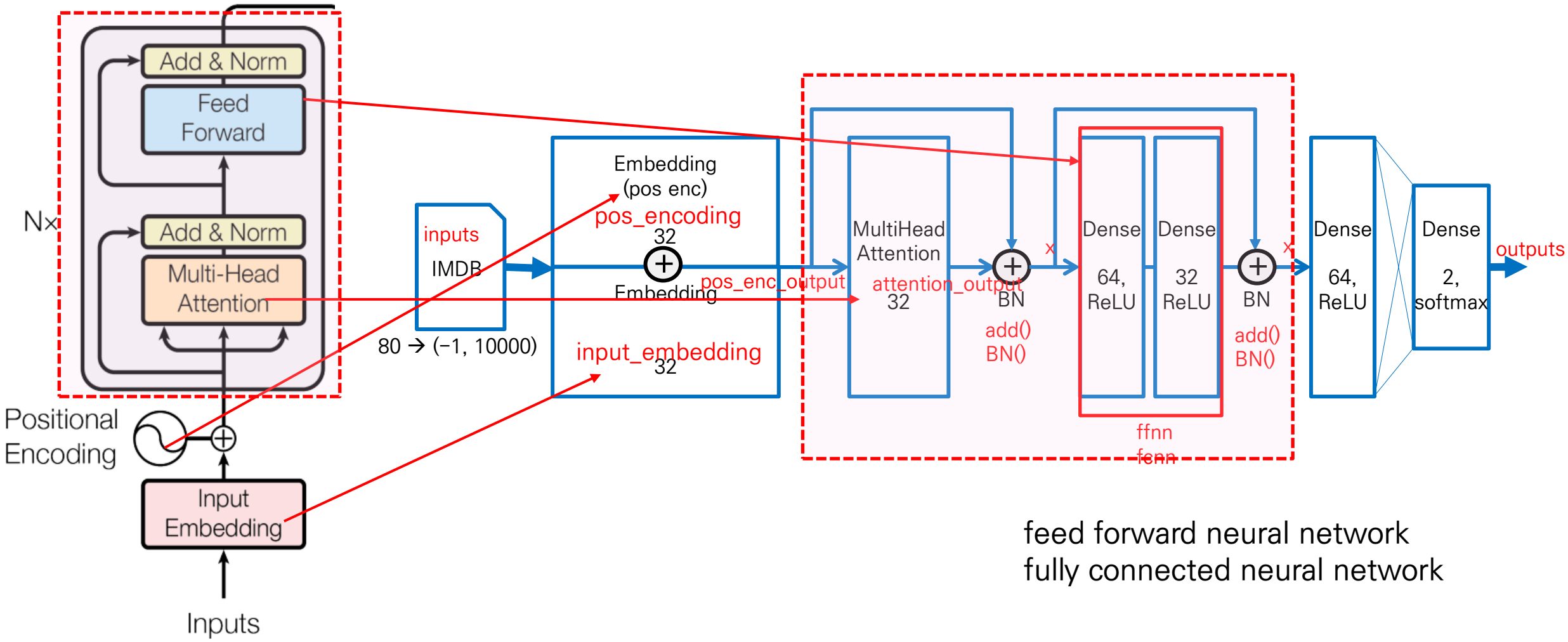
트랜스포머의 인코더 블록을 이용한 영화평 분류



트랜스포머의 인코더 블록을 이용한 영화평 분류 (모델 구조)



트랜스포머의 인코더 블록을 이용한 영화평 분류 (모델 구조)



feed forward neural network
fully connected neural network

트랜스포머의 인코더 블록을 이용한 영화평 분류


```python
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential, Model

inputs = layers.Input(shape=(80,))

input_embedding = layers.Embedding(input_dim=10000, output_dim=32)(inputs)
positions = tf.range(start=0, limit=80)
pos_encoding = layers.Embedding(input_dim=80, output_dim=32)(positions)
pos_enc_output = pos_encoding + input_embedding

attention_output = layers.MultiHeadAttention(num_heads=3, key_dim=32)(pos_enc_output, pos_enc_output)
x = layers.add([pos_enc_output, attention_output])
x = layers.BatchNormalization()(x)

ffnn = Sequential([layers.Dense(64, activation="relu"),
                   layers.Dense(32, activation="relu")])(x)
x = layers.add([ffnn, x])
x = layers.BatchNormalization()(x)
x = layers.GlobalAveragePooling1D()(x)
x = layers.Dropout(0.1)(x)

x = layers.Dense(64, activation="relu")(x)
x = layers.Dropout(0.1)(x)

outputs = layers.Dense(2, activation="softmax")(x)
model = Model(inputs=inputs, outputs=outputs)

model.summary()
```

트랜스포머 인코더 블록을 구현해서 영화평 분류에 사용한 예입니다.

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 80)	0	-
embedding (Embedding)	(None, 80, 32)	320,000	input_layer[0][0]
add (Add)	(None, 80, 32)	0	embedding[0][0]
multi_head_attention (MultiHeadAttention)	(None, 80, 32)	12,608	add[0][0], add[0][0]
add_1 (Add)	(None, 80, 32)	0	add[0][0], multi_head_attention[…
batch_normalization (BatchNormalization)	(None, 80, 32)	128	add_1[0][0]
sequential (Sequential)	(None, 80, 32)	4,192	batch_normalization[0…
add_2 (Add)	(None, 80, 32)	0	sequential[0][0], batch_normalization[0…
batch_normalization_1 (BatchNormalization)	(None, 80, 32)	128	add_2[0][0]
global_average_pooling1d (GlobalAveragePooling1D)	(None, 32)	0	batch_normalization_1…
dropout_1 (Dropout)	(None, 32)	0	global_average_poolin…
dense_2 (Dense)	(None, 64)	2,112	dropout_1[0][0]
dropout_2 (Dropout)	(None, 64)	0	dense_2[0][0]
dense_3 (Dense)	(None, 2)	130	dropout_2[0][0]

Total params: 339,298 (1.29 MB)
Trainable params: 339,170 (1.29 MB)
Non-trainable params: 128 (512.00 B)

트랜스포머의 인코더 블록을 이용한 영화평 분류


```
[2]  1 model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
[3]  1 from tensorflow.keras.datasets import imdb
     2 (X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
     3 print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ———————————————————————— 0s 0us/step
(25000,) (25000,) (25000,) (25000,)
```

```
[4]  1 from tensorflow.keras.preprocessing.sequence import pad_sequences
     2 X_train_pad = pad_sequences(X_train, maxlen=80, truncating='post', padding='post')
     3 X_test_pad = pad_sequences(X_test, maxlen=80, truncating='post', padding='post')
```

```
     1 %%time
     2 model.fit(X_train_pad, y_train, epochs=10, batch_size=200)
```

```
Epoch 1/10
125/125 ———————————————————————— 40s 280ms/step - accuracy: 0.6867 - loss: 0.5733
```

트랜스포머의 인코더 블록을 이용한 영화평 분류


```
[6]  1 model.evaluate(X_test_pad, y_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13s 16ms/step - accuracy: 0.7675 - loss: 1.7022
[1.7143645286560059, 0.7689999938011169]
```

```
[7]  1 print(X_test_pad.shape)
```

```
(25000, 80)
```

```
[8]  1 import numpy as np
     2 pred = model.predict(X_test_pad)
     3 # pred = (pred > 0.5).astype(int)
     4 pred = np.argmax(pred, axis=1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step
```

```
[9]  1 from sklearn.metrics import confusion_matrix
     2 print(confusion_matrix(y_test, pred))
```

```
[[10063  2437]
 [ 3338  9162]]
```

2장. 영화평 분류

허깅페이스 트랜스포머를 이용한 자연어처리

영화평 분류 모델

- 허깅페이스(Hugging Face) 트랜스포머 모델 중 영화 평 분류에 가장 널리 사용되는 모델은 BERT와 RoBERTa 기반 모델들임.
- IMDb 데이터셋을 사용한 영화 평 분류에 적합한 모델로는 "distilbert-base-uncased-finetuned-sst-2-english"와 "roberta-base" 모델이 많이 사용됨.
- 이 모델들은 감정 분석(Sentiment Analysis)에 특히 강력한 성능을 보임.

추천 모델

- distilbert-base-uncased-finetuned-sst-2-english
  - SST-2 데이터셋(Stanford Sentiment Treebank)으로 미세 조정된 BERT의 경량화 버전임.
  - 영화 평에서 감정(긍정/부정)을 분류하는 데 뛰어남.
- roberta-base
  - RoBERTa는 BERT의 변형
  - 더 많은 데이터로 학습되어 텍스트 분류 작업에 높은 성능을 보여줌.

- **DistilBERT**: "distilled" BERT (Bidirectional Encoder Representations from Transformers) 모델의 줄임말로, BERT 모델을 경량화한 버전입니다. DistilBERT는 원본 BERT 모델보다 작고 빠르지만 성능이 거의 동일하게 유지되도록 설계되었습니다. 머신 러닝 모델에서 "Distilled"라는 용어는 지식 증류(knowledge distillation)라는 기술을 의미합니다. 지식 증류는 큰 모델(teacher model)에서 작은 모델(student model)로 지식을 전이하여, 성능은 유사하게 유지하면서도 더 작고 효율적인 모델을 만드는 방법입니다.
- **base**: 모델의 크기를 나타내며, BERT 모델에는 base와 large가 있습니다. 여기서 base는 작은 크기의 BERT 모델로, 주로 파라미터 수를 줄인 모델입니다.
- **uncased**: 모델이 대소문자를 구분하지 않도록 훈련된 것을 의미합니다. 즉, 모든 입력 텍스트를 소문자로 변환하여 처리합니다.
- **finetuned**: 특정 데이터셋을 사용해 모델을 추가로 학습시켰음을 나타냅니다.
- **SST-2**: 모델이 파인 튜닝된 데이터셋을 의미하며, Stanford Sentiment Treebank (SST) 데이터셋의 2 클래스 버전을 가리킵니다. SST-2 데이터셋은 감정 분석에 주로 사용됩니다.

영화평 분류 예시


```python
from transformers import pipeline

# 감정 분석 파이프라인 설정
sentiment_analysis = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

# 예시 영화 평
reviews = [
    "This movie was absolutely fantastic! The acting, the plot, everything was perfect.",
    "I did not enjoy this movie at all. The story was predictable and the acting was terrible.",
    "A solid performance by the lead actor, but the plot could have been better.",
    "What a waste of time! I can't believe I sat through the whole movie."
]

# 영화 평 분석
results = sentiment_analysis(reviews)

# 결과 출력
for review, result in zip(reviews, results):
    print(f"Review: {review}\nSentiment: {result['label']} with score {result['score']:.2f}\n")
```

파인 튜닝

Hugging Face의 Transformers 라이브러리와 Datasets 라이브러리를 사용하여 IMDb 데이터셋을 불러오고, Trainer 클래스를 이용해 모델을 파인튜닝함.

과정

1. IMDb 데이터셋을 로드하고, 데이터를 모델에 맞게 토큰화함.
2. 사전 학습된 BERT 또는 DistilBERT(distilbert-base-uncased) 모델을 불러옴.
3. Hugging Face의 Trainer 클래스를 사용하여 파인튜닝을 진행함.
4. 학습이 끝난 후 모델을 저장하고 평가함.

데이터셋 불러오기


```
pip install datasets
```

datasets 라이브러리를 설치해야 합니다.
- IMDb 데이터셋은 Hugging Face의 datasets 라이브러리를 통해 쉽게 불러올 수 있습니다.

```
from datasets import load_dataset

# IMDb 데이터셋 불러오기
dataset = load_dataset("imdb")

# 데이터셋 확인
print(dataset)
```

이 코드는 IMDb 데이터셋을 자동으로 다운로드하고 로드합니다. 이 데이터셋에는 train, test 세트가 포함되어 있습니다.

```
README.md: 100% 7.81k/7.81k [00:00<00:00, 314kB/s]
train-00000-of-00001.parquet: 100% 21.0M/21.0M [00:00<00:00, 45.2MB/s]
test-00000-of-00001.parquet: 100% 20.5M/20.5M [00:00<00:00, 130MB/s]
unsupervised-00000-of-00001.parquet: 100% 42.0M/42.0M [00:00<00:00, 164MB/s]
Generating train split: 100% 25000/25000 [00:00<00:00, 71602.13 examples/s]
Generating test split: 100% 25000/25000 [00:00<00:00, 69594.71 examples/s]
Generating unsupervised split: 100% 50000/50000 [00:00<00:00, 96648.05 examples/s]
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
    })
})
```

데이터셋 전처리

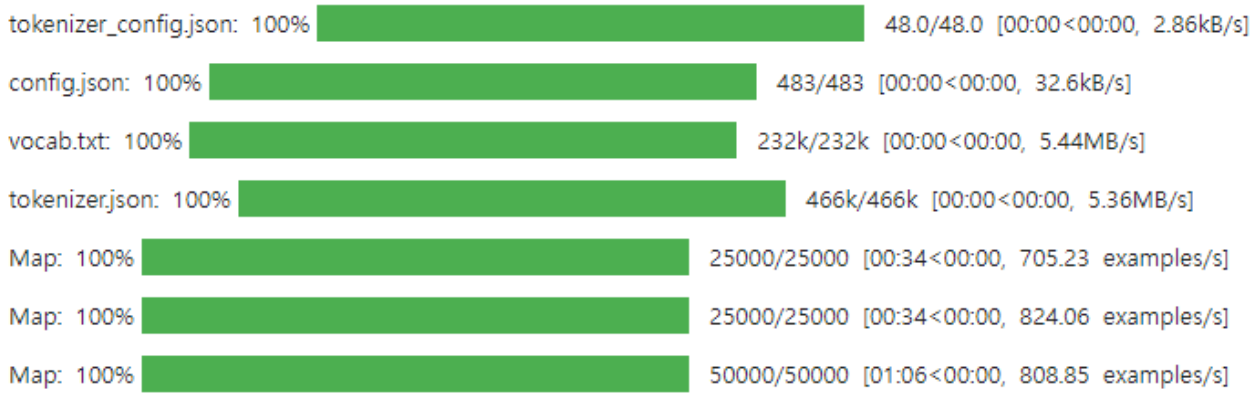
2장. 영화평 분류 / 파인튜닝

```python
from transformers import AutoTokenizer

# 토크나이저 설정 (distilbert-base-uncased 사용)
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

# 전처리 함수 정의
def preprocess_function(examples):
    return tokenizer(examples["text"], truncation=True, padding="max_length", max_length=512)

# 데이터셋 전처리
tokenized_datasets = dataset.map(preprocess_function, batched=True)
```



BERT나 RoBERTa 모델을 사용하려면, 텍스트 데이터를 토큰화해야 합니다. Hugging Face의 AutoTokenizer를 사용해 텍스트를 모델이 처리할 수 있는 형식으로 변환합니다.

이 코드는 IMDb 데이터셋의 텍스트 데이터를 distilbert-base-uncased 모델의 입력 형식에 맞게 변환합니다. 텍스트는 512개의 토큰으로 잘리고, 패딩이 적용됩니다.

모델 설정

- 이제 IMDb 데이터셋에서 감정 분석을 수행할 수 있는 사전 학습된 DistilBERT 모델을 불러옵니다.
- 여기서 num_labels=2는 IMDb 데이터셋의 라벨이 두 가지(긍정/부정)임을 나타냅니다.

```
from transformers import AutoModelForSequenceClassification

# 감정 분석을 위한 DistilBERT 모델 불러오기 (2개의 클래스로 분류)
model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=2)
```

```
model.safetensors: 100%|██████████| 268M/268M [00:02<00:00, 134MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'cl
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

모델 학습 설정


```python
from transformers import TrainingArguments, Trainer

# 학습 인자 설정
training_args = TrainingArguments(
    output_dir="./results",              # 출력 디렉토리
    evaluation_strategy="epoch",         # 평가 주기 (매 에포크마다)
    learning_rate=2e-5,                  # 학습률
    per_device_train_batch_size=16,      # 학습 시 배치 크기
    per_device_eval_batch_size=16,       # 평가 시 배치 크기
    num_train_epochs=3,                  # 에포크 수
    weight_decay=0.01,                   # 가중치 감소 (regularization)
)

# Trainer 생성
trainer = Trainer(
    model=model,                         # 학습할 모델
    args=training_args,                  # 학습 인자
    train_dataset=tokenized_datasets["train"],  # 학습 데이터셋
    eval_dataset=tokenized_datasets["test"],    # 평가 데이터셋
)
```

Hugging Face의 Trainer 클래스를 사용해 모델을 학습시킬 수 있습니다. 먼저 Trainer에 전달할 인자들을 설정합니다.

모델 파인튜닝 / 모델 평가

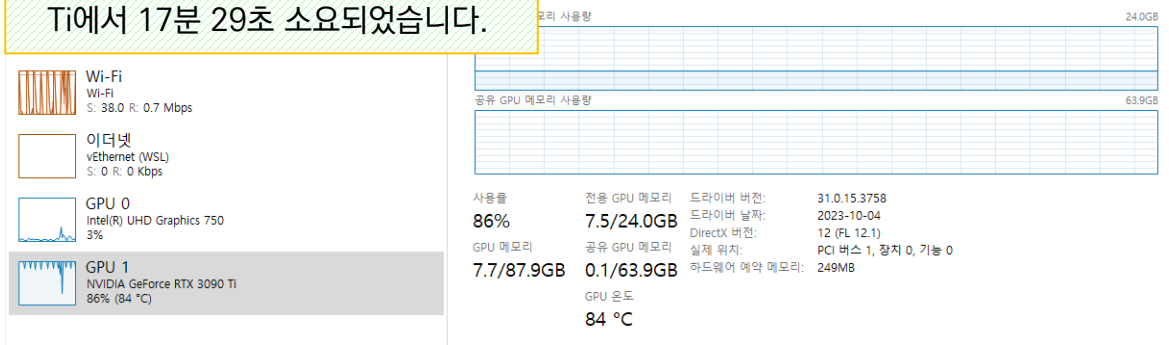
2장. 영화평 분류 / 파인튜닝

```
# 모델 학습
trainer.train()
```

[4689/4689 17:29, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.224200	0.248475
2	0.160000	0.222557
3	0.086900	0.287623

- CPU환경에서 3Epoch 학습하는데 10시간 정도 소요됩니다.
- NVIDIA GeForce RTX 3090 Ti에서 17분 29초 소요되었습니다.




```
TrainOutput(global_step=4689, training_loss=0.16810285991685786, metrics={'train_runtime': 1051.2343, 'train_samples_per_second': 71.345, 'train_steps_per_second': 4.46, 'total_flos': 9935054899200000.0, 'train_loss': 0.16810285991685786, 'epoch': 3.0})
```

```
# 모델 평가
results = trainer.evaluate()
print(results)
```

[1563/1563 01:31]

```
{'eval_loss': 0.2876228392124176, 'eval_runtime': 92.0404, 'eval_samples_per_second': 271.62, 'eval_steps_per_second': 16.982, 'epoch': 3.0}
```

모델 저장하기 / 불러오기


```
# 모델과 토크나이저 경로
model_path = "./fine_tuned_distilbert_imdb"

# 모델 저장
model.save_pretrained(model_path)
tokenizer.save_pretrained(model_path)
```

```
('./fine_tuned_distilbert_imdb₩₩tokenizer_config.json',
 './fine_tuned_distilbert_imdb₩₩special_tokens_map.json',
 './fine_tuned_distilbert_imdb₩₩vocab.txt',
 './fine_tuned_distilbert_imdb₩₩added_tokens.json',
 './fine_tuned_distilbert_imdb₩₩tokenizer.json')
```

학습된 모델의 압축을 풀어 Colab에 업로드 하세요.

```
# 모델과 토크나이저 로드
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForSequenceClassification.from_pretrained(model_path)

# 모델을 평가 모드로 전환
model.eval()
```

불러온 모델은 평가 모드로 전환해야 모델을 평가할 수 있습니다.

```
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

영화평 분류 예측하기


```python
import torch

# 예시 입력 텍스트
input_text = "This movie was absolutely fantastic!"

# 입력 텍스트를 토큰화
inputs = tokenizer(input_text, return_tensors="pt")

# 모델에 입력 전달
with torch.no_grad():
    outputs = model(**inputs)

# 로짓 값에서 예측 레이블 추출
logits = outputs.logits
predicted_class_id = torch.argmax(logits, dim=-1).item()

# 예측된 클래스 확인
print(f"Predicted class ID: {predicted_class_id}")
```

```
Predicted class ID: 1
```

3장. 챗봇 만들기

허깅페이스 트랜스포머를 이용한 자연어처리

gpt2 모델을 이용한 챗봇 만들기

- 허깅페이스의 gpt2를 이용해서 챗봇을 만들 수 있습니다.

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# 모델과 토크나이저 불러오기 (사전 학습된 GPT-2 모델 사용)
model_name = "gpt2"  # gpt2 대신 'gpt2-medium', 'gpt2-large' 등의 모델을 선택할 수 있습니다.
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# pad_token을 eos_token으로 설정
tokenizer.pad_token = tokenizer.eos_token

# pad_token_id를 eos_token_id로 설정
model.config.pad_token_id = tokenizer.eos_token_id
```

gpt2 모델을 이용한 챗봇 만들기


```
def generate_response(input_text):
    # 입력 텍스트를 토큰화하고, attention_mask 반환
    inputs = tokenizer.encode_plus(
        input_text,
        return_tensors="pt",
        padding=True,
        truncation=True,
        max_length=512
    )

    # 모델에 입력과 attention_mask를 전달하여 답변 생성
    output = model.generate(
        inputs['input_ids'],
        attention_mask=inputs['attention_mask'],  # attention_mask 설정
        max_length=100,
        num_return_sequences=1,
        pad_token_id=tokenizer.eos_token_id  # pad_token_id 설정
    )

    # 응답 디코딩
    response = tokenizer.decode(output[0], skip_special_tokens=True)
    return response
```

gpt2 모델을 이용한 챗봇 만들기


```
# 간단한 대화 인터페이스
if __name__ == "__main__":
    print("챗봇과 대화하려면 메시지를 입력하세요 ('종료' 입력 시 종료):")
    while True:
        user_input = input("사용자: ")
        if user_input.lower() == "종료":
            break
        bot_response = generate_response(user_input)
        print(f"챗봇: {bot_response}")
```

4장. 이미지에서 객체 탐지하기

허깅페이스 트랜스포머를 이용한 자연어처리

이미지 객체 탐지

- 허깅페이스의 트랜스포머 라이브러리는 자연어 처리 외에도 Vision Transformer(ViT)와 같은 트랜스포머 기반 모델을 활용하여 이미지 분류 및 객체 탐지(Object Detection) 작업에도 사용할 수 있음.
- 객체 탐지(Object Detection)를 수행하기 위해선 주로 DEtection TRansformers (DETR) 모델을 사용함
- DETR은 이미지에서 객체를 탐지하고, 그 위치를 바운딩 박스로 반환해줌.

```
pip install transformers torch torchvision matplotlib

pip install timm
```

Google Colab에서는 timm 라이브러리만 설치하세요.

모델 불러오기


```
from transformers import DetrForObjectDetection, DetrImageProcessor
import torch
from PIL import Image, ImageDraw
import requests

# DETR 모델과 프로세서를 로드합니다.
model = DetrForObjectDetection.from_pretrained("facebook/detr-resnet-50")
processor = DetrImageProcessor.from_pretrained("facebook/detr-resnet-50")
```

DetrForObjectDetection은 DETR 모델을 로드하고,
DetrImageProcessor는 이미지를 전처리합니다.

이미지에서 객체 탐지 예측하기


```
import matplotlib.pyplot as plt

# 예시 이미지를 불러옵니다.
image = Image.open("sample.jpg")

# 이미지를 모델에 맞게 전처리합니다.
inputs = processor(images=image, return_tensors="pt")

# 모델 예측
with torch.no_grad():
    outputs = model(**inputs)

# 예측 결과에서 로짓 값을 가져옵니다.
target_sizes = torch.tensor([image.size[::-1]])  # 모델에 맞게 사이즈를 맞춥니다.
results = processor.post_process_object_detection(outputs, target_sizes=target_sizes)[0]
```

바운딩 박스 그리고 출력하기


```
# 바운딩 박스를 이미지에 그립니다.
draw = ImageDraw.Draw(image)

for score, label, box in zip(results["scores"], results["labels"], results["boxes"]):
    if score > 0.9:  # 예측 확률이 90% 이상인 객체만 필터링합니다.
        box = [round(i, 2) for i in box.tolist()]
        draw.rectangle(box, outline="red", width=3)
        draw.text((box[0], box[1]), f"{model.config.id2label[label.item()]}: {round(score.item(), 3)}", fill="red")

# Matplotlib을 이용해 이미지 출력
plt.figure(figsize=(8, 8))
plt.imshow(image)
plt.axis("off")  # 축 제거
plt.show()
# 객체 탐지 결과 이미지 출력
# image.show()
```