트랜스포머를 활용한 자연어처리

2024.07.03

박현빈

목차

1. 책의 목표
2. 트랜스포머 소개
    1. 인코더-디코더
    2. Attention
    3. Transformer
3. 텍스트 분류(코드 실습)
    1. 데이터셋
    2. 토큰화
    3. 모델 구조
    4. 모델 훈련

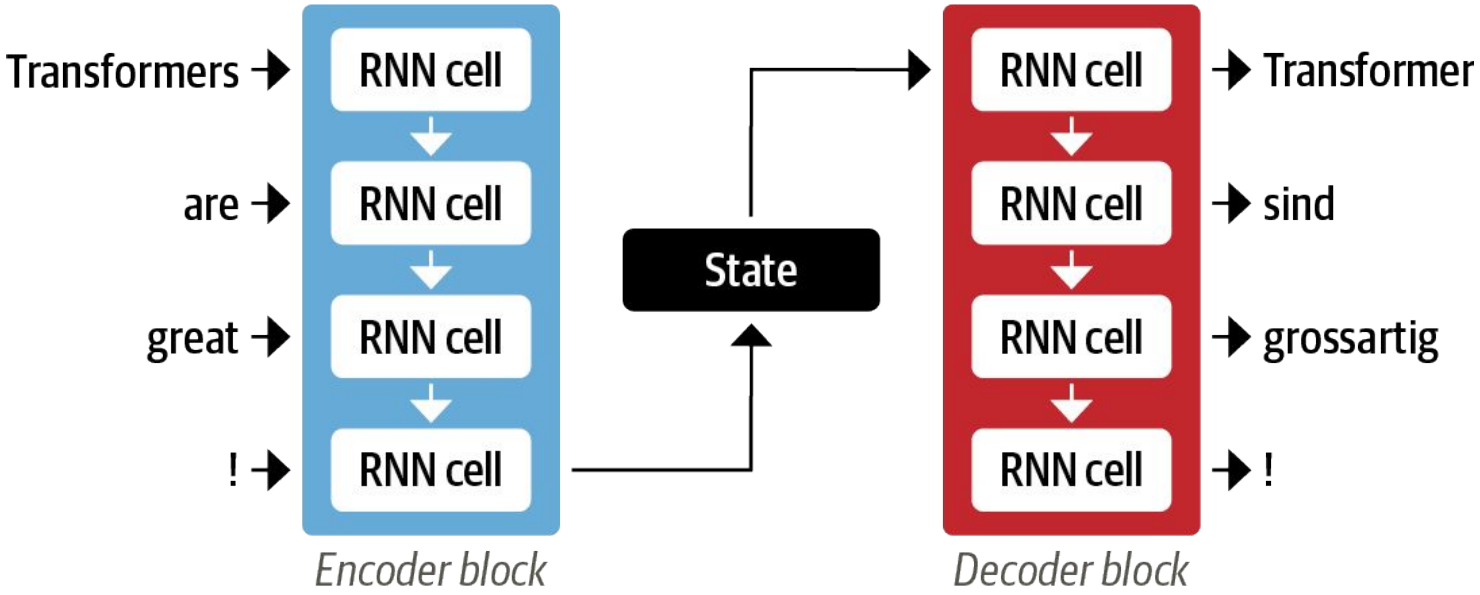
책의 목표

- 독자가 자신만의 언어 애플리케이션을 만들도록 돕는 것
  - 실용적인 사례에 초점을 맞춤
  - 필요한 경우에만 이론을 다룸
  - Pytorch와 Hugging Face의 Transformers 라이브러리 사용

Chapter 1 트랜스포머 소개

Chapter 2 텍스트 분류

Encoder – Decoder

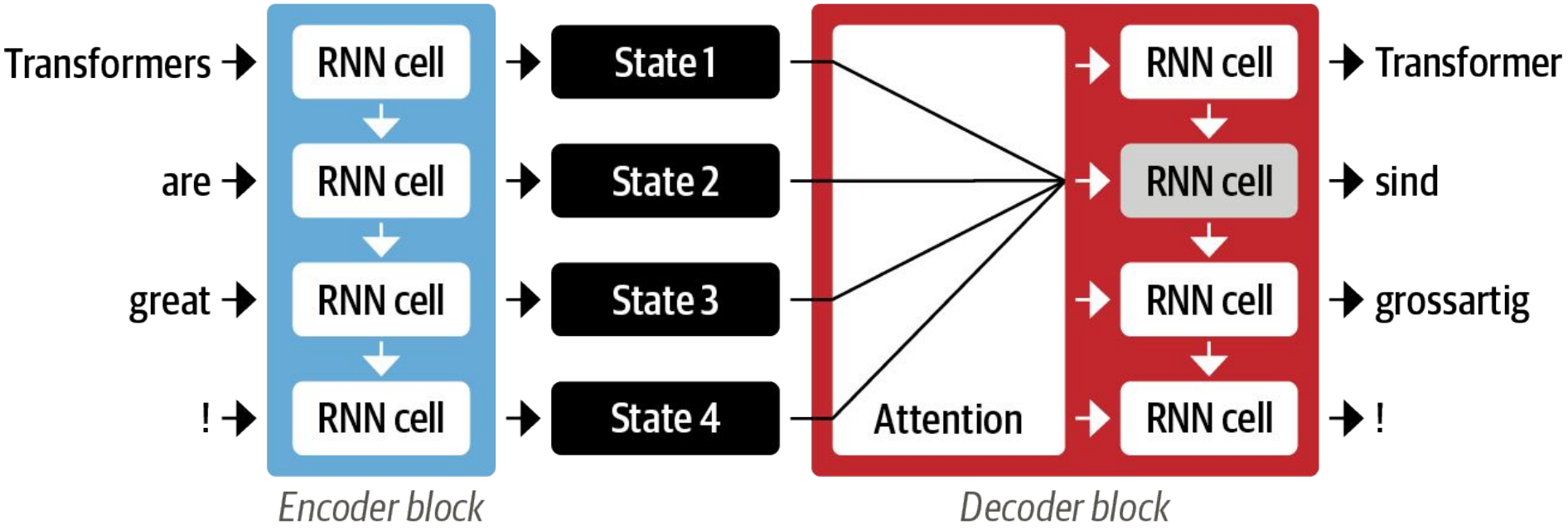


인코더 : input sequence를 하나의 context vector로 압축

디코더 : context vector를 사용해서 새로운 sequence 출력

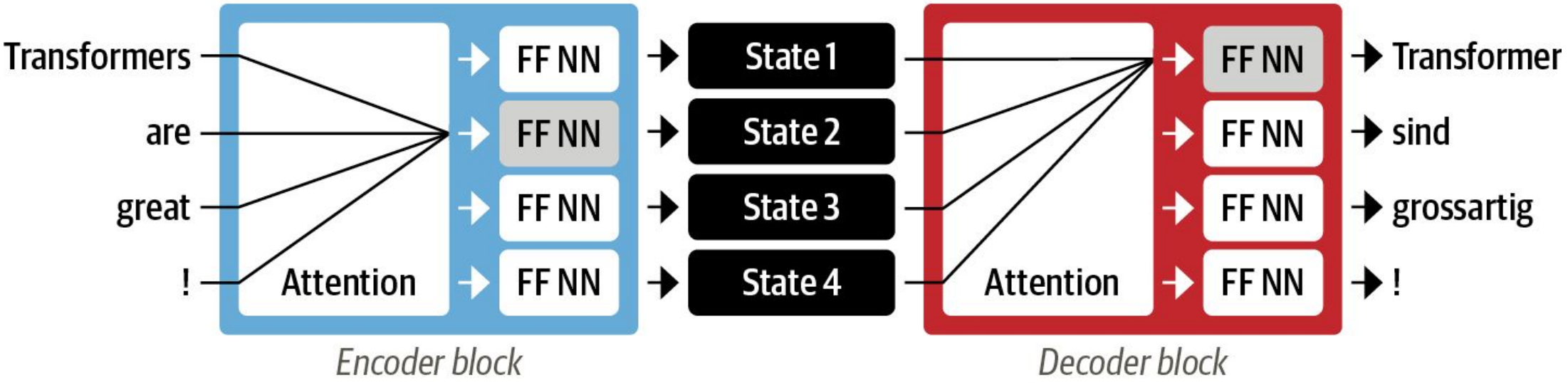
단점 : 인코더가 압축하는 과정에서 정보 손실이 발생

Attention

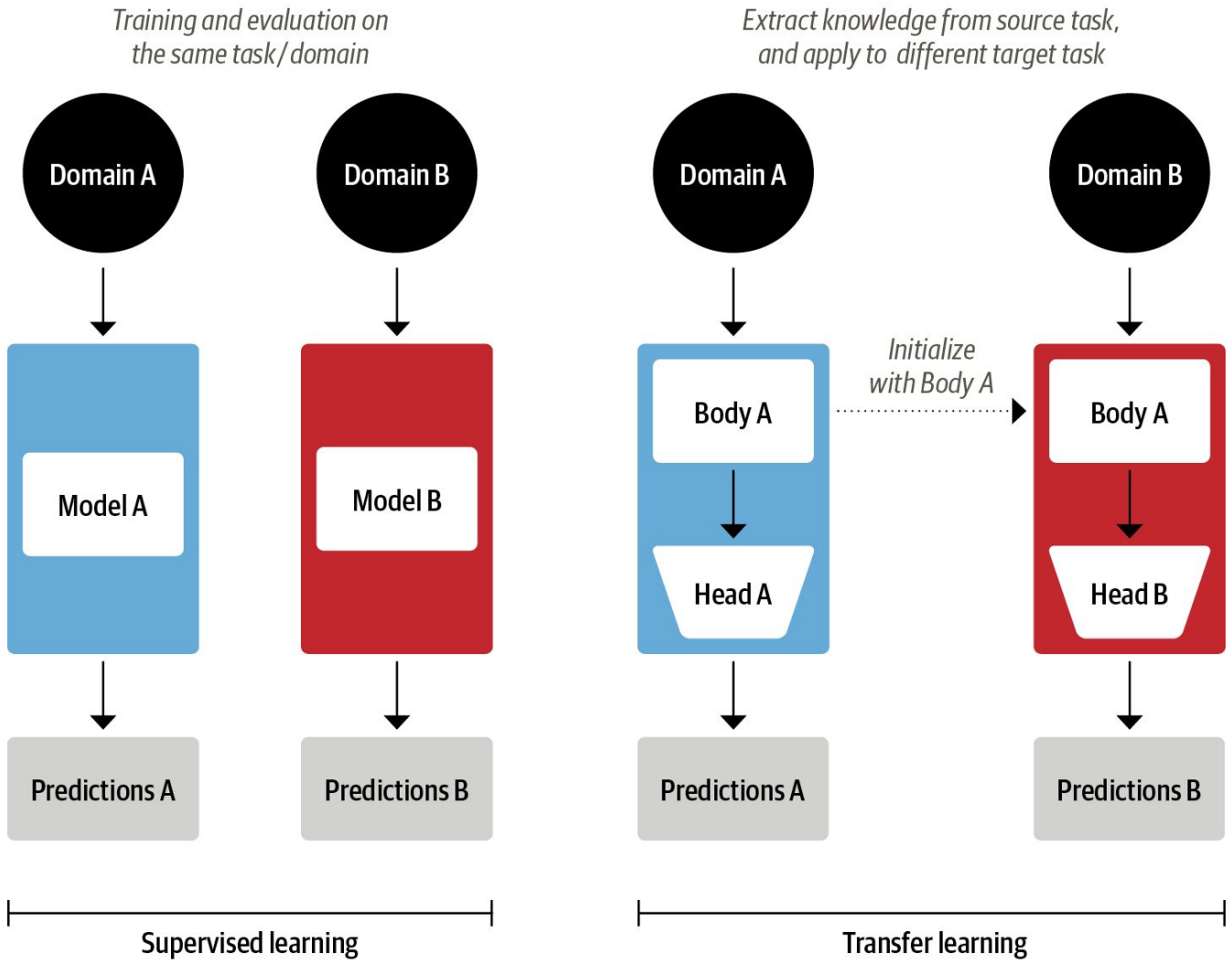


디코더의 각 타임스텝에서 인코더의 모든 hidden state을 참고
각 hidden state에 적절한 가중치(attention)를 할당해서 예측값 생성

Transformer



전이 학습


Training and evaluation on the same task/domain
Extract knowledge from source task, and apply to different target task
Domain A
Domain B
Domain A
Domain B
Model A
Model B
Body A
Head A
Initialize with Body A
Body A
Head B
Predictions A
Predictions B
Predictions A
Predictions B
Supervised learning
Transfer learning

텍스트 분류

데이터셋

- 모델 : DistilBERT
- Hugging Face의 “emotion”

```
from datasets import load_dataset
```

```
emotions = load_dataset("emotion")
```

```
emotions
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 16000
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
})
```

```
train_ds = emotions["train"]
```

```
train_ds[0]
```

```
{'text': 'i didnt feel humiliated', 'label': 0}
```

```
train_ds.features
```

```
{'text': Value(dtype='string', id=None),
 'label': ClassLabel(names=['sadness', 'joy', 'love', 'anger', 'fear', 'surprise'], id=None)}
```

토큰화

- 문자 토큰화

```
text = "Tokenizing text is a core task of NLP."
tokenized_text = list(text)
print(tokenized_text)
```

```
['T', 'o', 'k', 'e', 'n', 'i', 'z', 'i', 'n', 'g', ' ', 't', 'e', 'x', 't', ' ',
'i', 's', ' ', 'a', ' ', 'c', 'o', 'r', 'e', ' ', 't', 'a', 's', 'k', ' ', 'o',
'f', ' ', 'N', 'L', 'P', '.']
```

```
[5, 14, 12, 8, 13, 11, 19, 11, 13, 10, 0, 17, 8, 18, 17, 0, 11, 16, 0, 6, 0, 7,
14, 15, 8, 0, 17, 6, 16, 12, 0, 14, 9, 0, 3, 2, 4, 1]
```

- 단어 토큰화

```
tokenized_text = text.split()
print(tokenized_text)
```

```
['Tokenizing', 'text', 'is', 'a', 'core', 'task', 'of', 'NLP.']
```

Hugging Face로 한 문장 토큰화

```
from transformers import AutoTokenizer

model_ckpt = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
encoded_text = tokenizer(text)
print(encoded_text)
```

```
{'input_ids': [101, 19204, 6026, 3793, 2003, 1037, 4563, 4708, 1997, 17953,
2361, 1012, 102], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

```
tokens = tokenizer.convert_ids_to_tokens(encoded_text.input_ids)
print(tokens)
```

```
['[CLS]', 'token', '##izing', 'text', 'is', 'a', 'core', 'task', 'of', 'nl',
'##p', '.', '[SEP]']
```

```
print(tokenizer.convert_tokens_to_string(tokens))
```

```
[CLS] tokenizing text is a core task of nlp. [SEP]
```

전체 데이터셋 토큰화

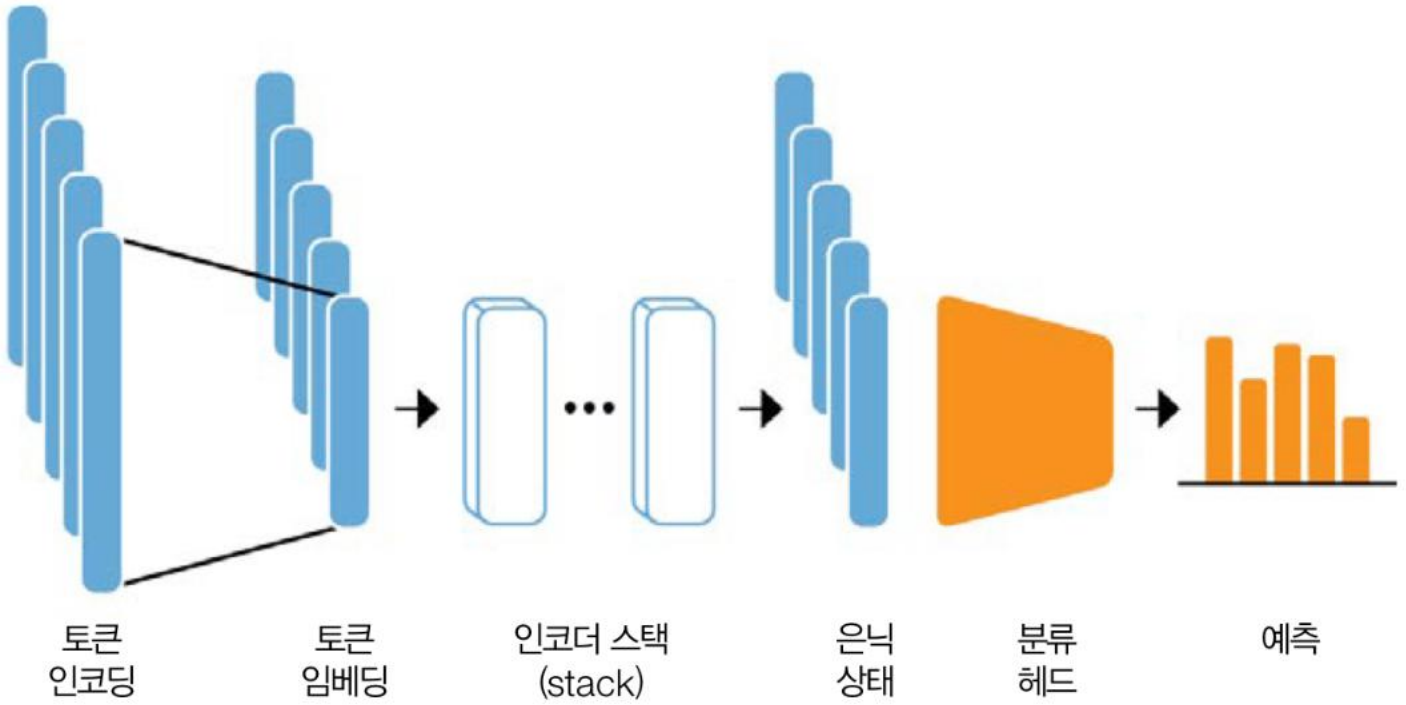
```
def tokenize(batch):
    return tokenizer(batch["text"], padding=True, truncation=True)

print(tokenize(emotions["train"][:2]))
```

```
{'input_ids': [[101, 1045, 2134, 2102, 2514, 26608, 102, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [101, 1045, 2064, 2175, 2013, 3110, 2061, 20625, 2000,
2061, 9636, 17772, 2074, 2013, 2108, 2105, 2619, 2040, 14977, 1998, 2003, 8300,
102]], 'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1]]}
```

```
emotions_encoded = emotions.map(tokenize, batched=True, batch_size=None)
```

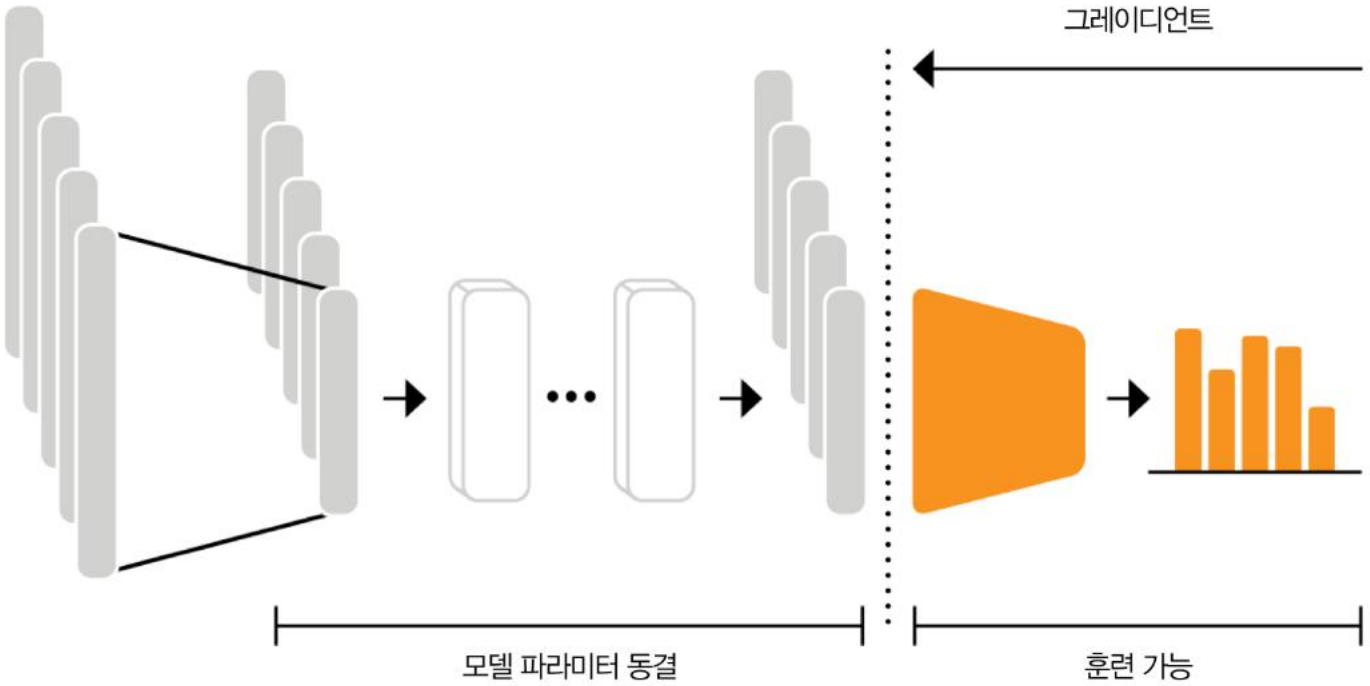
모델 구조(인코더 기반 트랜스포머)



- 토큰 인코딩 : 원-핫 벡터. 벡터의 차원은 vocab의 단어 수
- 토큰 임베딩 : 토큰 인코딩을 저차원으로 매핑
- 인코더 스택 : Self-Attention 기법으로 토큰 임베딩을 은닉 상태로 변환
- 분류 헤드 : 은닉상태를 data로 삼아 예측 생성

모델 훈련(1)

- 헤드만 훈련



1. AutoModel 클래스로 모델 로드

```
from transformers import AutoModel

model_ckpt = "distilbert-base-uncased"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = AutoModel.from_pretrained(model_ckpt).to(device)
```

2. 토큰화 후 last hidden state 생성

Last hidden state = 새롭게 인코딩 된 text

```
text = "this is a test"
inputs = tokenizer(text, return_tensors="pt")
print(f"입력 텐서 크기: {inputs['input_ids'].size()}")
```

```
입력 텐서 크기: torch.Size([1, 6])
```

[batch_size, n_tokens]

```
inputs
```

```
{'input_ids': tensor([[ 101, 2023, 2003, 1037, 3231,  102]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1]])}
```

```
inputs = {k:v.to(device) for k,v in inputs.items()}
with torch.no_grad():
    outputs = model(**inputs)
print(outputs)
```

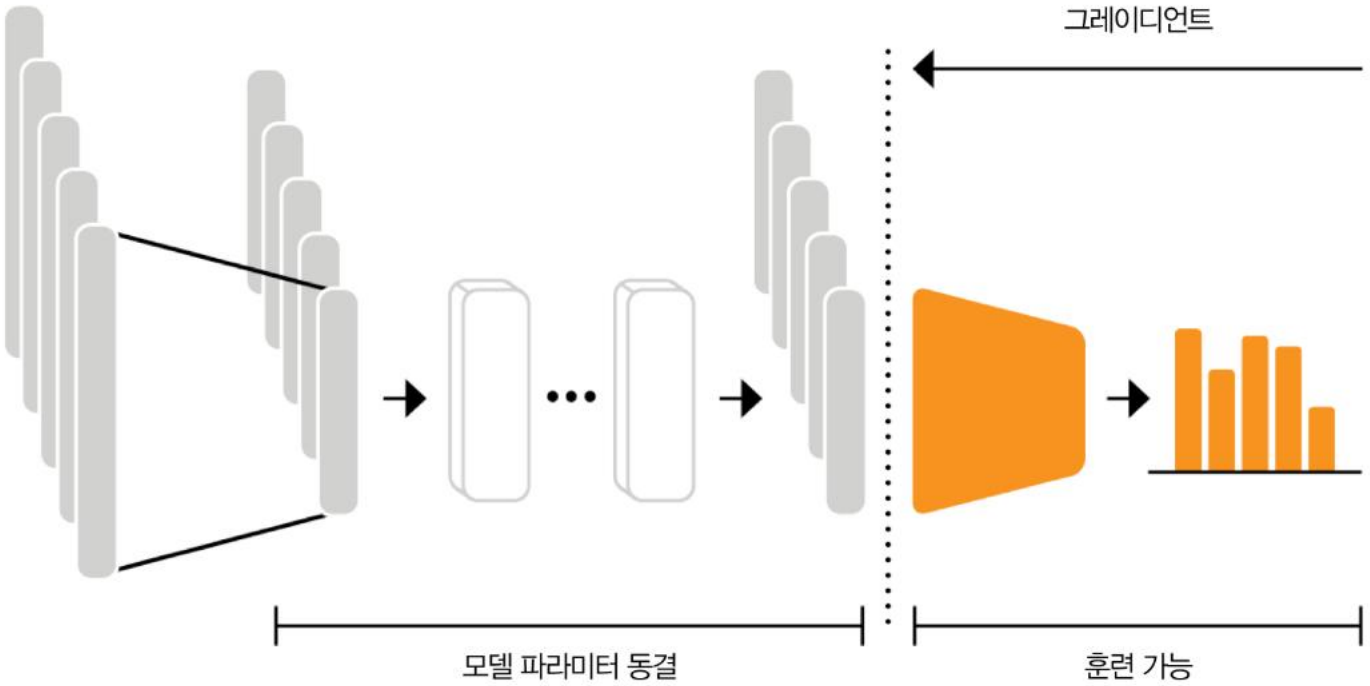
```
BaseModelOutput(last_hidden_state=tensor([[[-0.1565, -0.1862,  0.0528,  ...,
-0.1188,  0.0662,  0.5470],
         [-0.3575, -0.6484, -0.0618,  ..., -0.3040,  0.3508,  0.5221],
         [-0.2772, -0.4459,  0.1818,  ..., -0.0948, -0.0076,  0.9958],
         [-0.2841, -0.3917,  0.3753,  ..., -0.2151, -0.1173,  1.0526],
         [ 0.2661, -0.5094, -0.3180,  ..., -0.4203,  0.0144, -0.2149],
         [ 0.9441,  0.0112, -0.4714,  ...,  0.1439, -0.7288, -0.1619]]],
       device='cuda:0'), hidden_states=None, attentions=None)
```

```
outputs.last_hidden_state.size()
```

```
torch.Size([1, 6, 768])
```

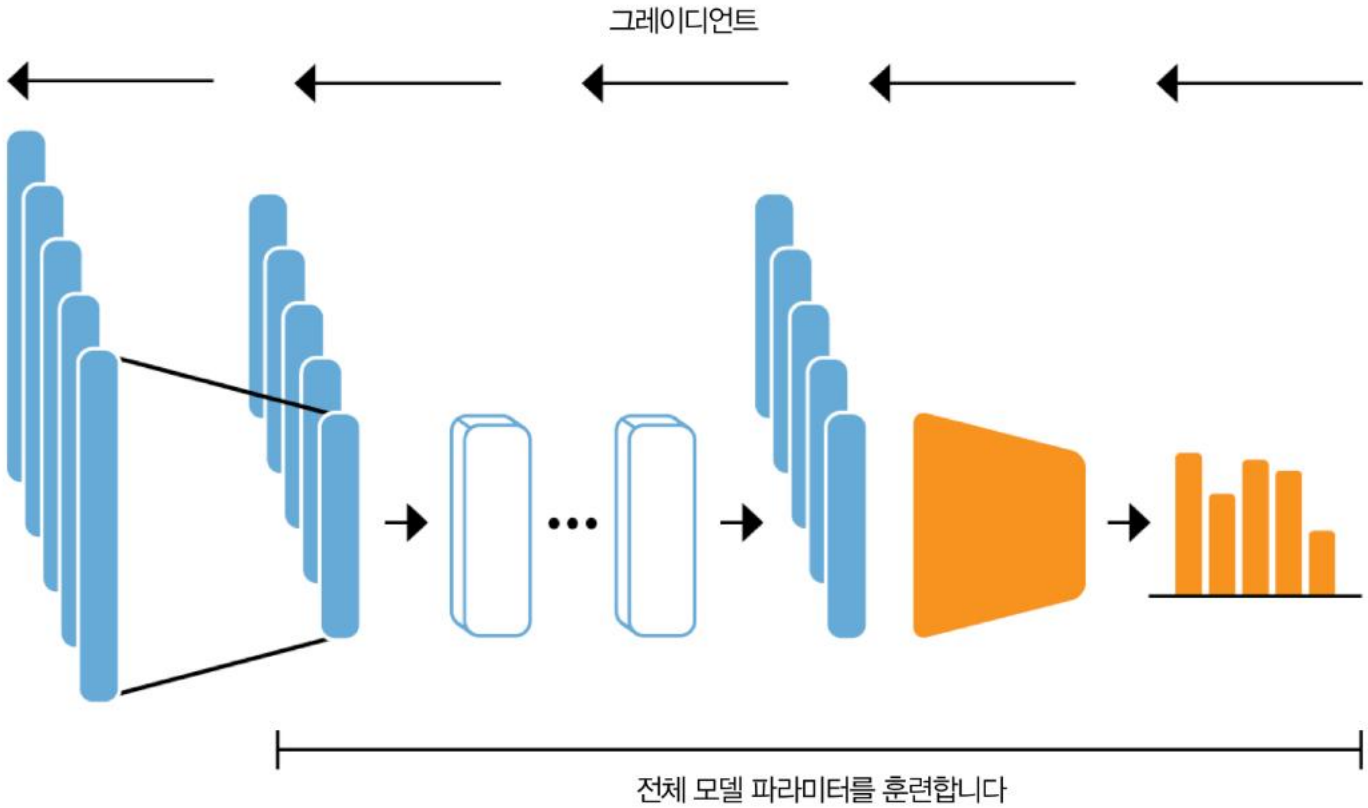
모델 훈련(1)

- 헤드만 훈련



모델 훈련(2)

- End-to-end fine-tunning



1. AutoModelForSequenceClassification 클래스로 모델 로드

```
from transformers import AutoModelForSequenceClassification

num_labels = 6
model = (AutoModelForSequenceClassification
         .from_pretrained(model_ckpt, num_labels=num_labels)
         .to(device))
```

2. Trainer 클래스로 모델 훈련 및 검증

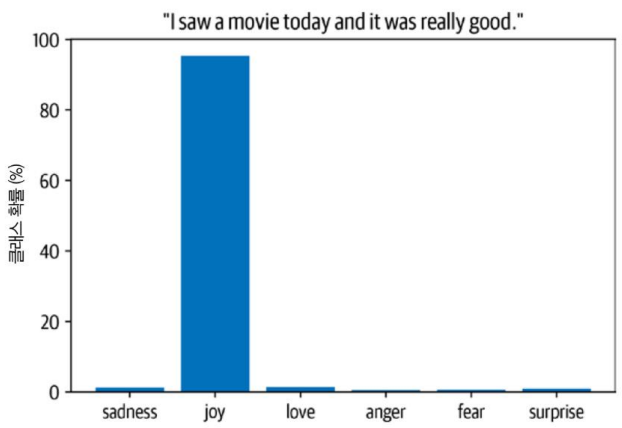
```
from transformers import Trainer, TrainingArguments

batch_size = 64
logging_steps = len(emotions_encoded["train"]) // batch_size
model_name = f"{model_ckpt}-finetuned-emotion"
training_args = TrainingArguments(output_dir=model_name,
                                  num_train_epochs=2,
                                  learning_rate=2e-5,
                                  per_device_train_batch_size=batch_size,
                                  per_device_eval_batch_size=batch_size,
                                  weight_decay=0.01,
                                  evaluation_strategy="epoch",
                                  disable_tqdm=False,
                                  logging_steps=logging_steps,
                                  push_to_hub=True,
                                  save_strategy="epoch",
                                  load_best_model_at_end=True,
                                  log_level="error")
```

```
from transformers import Trainer

trainer = Trainer(model=model, args=training_args,
                  compute_metrics=compute_metrics,
                  train_dataset=emotions_encoded["train"],
                  eval_dataset=emotions_encoded["validation"],
                  tokenizer=tokenizer)
trainer.train();
```

Epoch	Training Loss	Validation Los
1	0.831500	0.31313
2	0.242800	0.21368



Q&A