

# GEN AI 인텐시브 과정

---

강사장철원

## Section 0

### 코스소개

DAY1

DAY2

DAY3

DAY4

DAY5

DAY6

DAY7

DAY8

LLM  
Basic  
Concept

Transformers  
paper  
review

Transformers  
LangChain  
LangGraph

LLM  
service  
develop

Final Project

□ 풀 파인튜닝

□ PEFT

□ LoRA

# GEN AI 인텐시브 과정

Section 1. 파인튜닝

---

**Section 1-1. 긍정 분석 모델 풀 파인튜닝(Full Fine Tuning)**

# import 트랜스포머 라이브러리

```
import numpy as np
import pandas as pd
import torch
from datasets import Dataset → 허깅페이스에서 만든 데이터 관련 라이브러리, 모델 학습 파이프라인에 사용하기 위함
from sklearn.model_selection import train_test_split
from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments
```

↓  
허깅페이스에서 제공하는 통합 학습 실행기  
(모델 학습, 저장, 평가, 예측까지 올인원)

↓  
학습 환경을 설정하는 하이퍼파라미터 묶음 객체  
(학습을 어떻게 할지에 대한 설정)

## Section

### 공부정 분석모델풀파인튜닝

# 학습시킴 데이터 확인

```
df = pd.read_csv("../data/review_data.csv", encoding='cp949')
```

df

text labels

0 배우들 연기도 너무 좋았어요. 1

1 스토리가 탄탄하고 연출도 훌륭했어요. 1

2 정말 감동적인 영화였습니다. 눈물이 멈추질 않았어요. 1

3 끝까지 집중해서 봤습니다. 완전 추천해요! 1

⋮

16 기대한 만큼 실망만 컸어요. 0

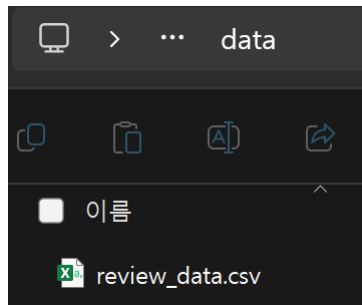
17 시간 아깝고 돈 아깝네요. 0

18 한 번 보고 다시는 보고 싶지 않아요. 0

19 결말도 어이없고 전체적으로 별로였어요. 0

긍정이면 1

부정이면 0



	A	B
1	text	labels
2	배우들 연기도 너무 좋았어요.	1
3	스토리가 탄탄하고 연출도 훌륭했어요.	1
4	정말 감동적인 영화였습니다. 눈물이 멈추질 않았어요.	1
5	끝까지 집중해서 봤습니다. 완전 추천해요!	1
6	감성과 메시지가 모두 살아있는 영화였어요.	1
7	오랜만에 이런 좋은 영화를 봐서 기분이 좋아요.	1
8	음악, 연출, 연기 모두 완벽했어요.	1
9	시간 가는 줄 몰랐어요. 최고의 영화!	1
10	스토리 전개가 매끄럽고 감동적이었어요.	1
11	친구들에게 꼭 추천하고 싶은 영화예요.	1
12	지루해서 중간에 졸았어요.	0
13	배우 연기가 너무 어색해서 몰입이 안 됐어요.	0
14	스토리가 산만하고 전개가 엉성해요.	0
15	예고편이 다였네요. 본편은 별로예요.	0
16	너무 뻘한 이야기라 재미가 없었어요.	0
17	연출이 허술하고 대사도 부자연스러워요.	0
18	기대한 만큼 실망만 컸어요.	0
19	시간 아깝고 돈 아깝네요.	0
20	한 번 보고 다시는 보고 싶지 않아요.	0
21	결말도 어이없고 전체적으로 별로였어요.	0

# train/test 분할 & 허깅페이스 데이터셋 변환

트레이닝 데이터 16개, 테스트 데이터 4개

```
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['labels'], random_state=0)
```

*# Huggingface Dataset 형태로 변환*

```
train_dataset = Dataset.from_pandas(train_df)
```

```
test_dataset = Dataset.from_pandas(test_df)
```

허깅페이스 Dataset 객체로 변환

- 이후 사용할 trainer 함수와 호환을 위해 사용

```
train_dataset
```

```
Dataset({
  features: ['text', 'labels', '__index_level_0__'],
  num_rows: 16
})
```

```
train_dataset[0]
```

```
{'text': '정말 감동적인 영화였습니다. 눈물이 멈추질 않았어요.', 'labels': 1, '__index_level_0__': 2}
```

## Section

### 긍부정 분석 모델 풀 파인튜닝

# 모델 & 전처리

```
model_name = "beomi/kcbert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

```
def preprocess(data):
    res = tokenizer(data["text"], truncation=True, padding="max_length", max_length=64)
    return res
```

문장이 너무 길어서 max\_length를 초과하면 자르겠다.

길이가 너무 짧으면 0으로 채우겠다.

return\_tensors 옵션 안쓰면 기본 딕셔너리로 출력

```
train_dataset = train_dataset.map(preprocess, batched=True)
test_dataset = test_dataset.map(preprocess, batched=True)
```

문장 최대길이 64토큰

Map: 100%  16/16 [00:00<00:00, 1317.75 examples/s]

Map: 100%  4/4 [00:00<00:00, 399.82 examples/s]

```
train_dataset
```

```
Dataset({
  features: ['text', 'labels', '__index_level_0__', 'input_ids', 'token_type_ids', 'attention_mask'],
  num_rows: 16
})
```

```
train_dataset[0]
```

```
{'text': '정말 감동적인 영화였습니다. 눈물이 멈추질 않았어요.',
 'labels': 1,
 '__index_level_0__': 2,
 'input_ids': [2,
 8050,
```



# 학습 환경 설정

# 4. *Trainer* 설정 - 모델 저장 안함

```
train_args = TrainingArguments(  
    output_dir="./saved_models/basic_sentiment1",  
    eval_strategy="epoch",  
    save_strategy="no",  
    per_device_train_batch_size=4,  
    per_device_eval_batch_size=4,  
    num_train_epochs=3,  
    logging_dir="./logs",  
    logging_strategy="epoch",  
    use_cpu=True  
)
```

모델, 로그 저장 폴더

**epoch vs steps**(다음 페이지 설명)

CPU 사용 여부, False로 설정하면 GPU를 사용하겠다는 의미

The diagram illustrates the relationship between training data, mini-batches, epochs, and steps. It shows a large grid of 700 training data points (blue) and a smaller grid of 300 test data points (gray). The training data is divided into 7 mini-batches of 100 each. A large blue arrow points from the training data to the right, indicating the flow of data. To the right of the arrow, it states "1 epoch = 7 step". Below this, the terms "epoch" and "step" are defined:

- epoch**
  - 전체 트레이닝 데이터
- step**
  - 하나의 미니배치를 드업데이트 하는 과정

1 epoch = 7 step

# epoch

- 전체 트레이닝 데이터를 한 번 모두 사용하는 주기

## step

- 하나의 미니배치를 모델에 넣고 파라미터를 한번 업데이트 하는 과정

# Trainer(학습기) 설정 및 학습

[15]: # 정확도 계산 함수

```
def accuracy_score(predict):  
    preds = np.argmax(predict.predictions, axis=1)  
    acc = (preds == predict.label_ids).mean()  
    return {"accuracy": acc}
```

[16]: trainer = Trainer(  
 model=model, 사용할 모델  
 args=train\_args, 학습 환경  
 train\_dataset=train\_dataset, 트레이닝 데이터  
 eval\_dataset=test\_dataset, 테스트 데이터  
 compute\_metrics=accuracy\_score, 평가 기준  
)

[17]: # 5. 학습

```
trainer.train()
```

[12/12 00:28, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.762500	0.534013	0.750000
2	0.530700	0.486727	0.750000
3	0.258300	0.367389	1.000000

# 예측

# 6. 테스트 예측

```
test_texts = ["이 제품 너무 좋아요!", "별로예요. 추천 안함."]
inputs = tokenizer(test_texts, return_tensors="pt", padding=True, truncation=True, max_length=64)

model.eval()
with torch.no_grad():
    outputs = model(**inputs)
preds = torch.argmax(outputs.logits, dim=1)
print("예측 결과:", preds.tolist())
```

예측 결과: [1, 0]

긍정   부정

[19]: vars(outputs)

[19]: {'loss': None,  
      'logits': tensor([[ -0.5843,  0.7482],  
                      [ 0.5842, -0.4242]]),  
      'hidden\_states': None,  
      'attentions': None}

# 최종 모델 저장

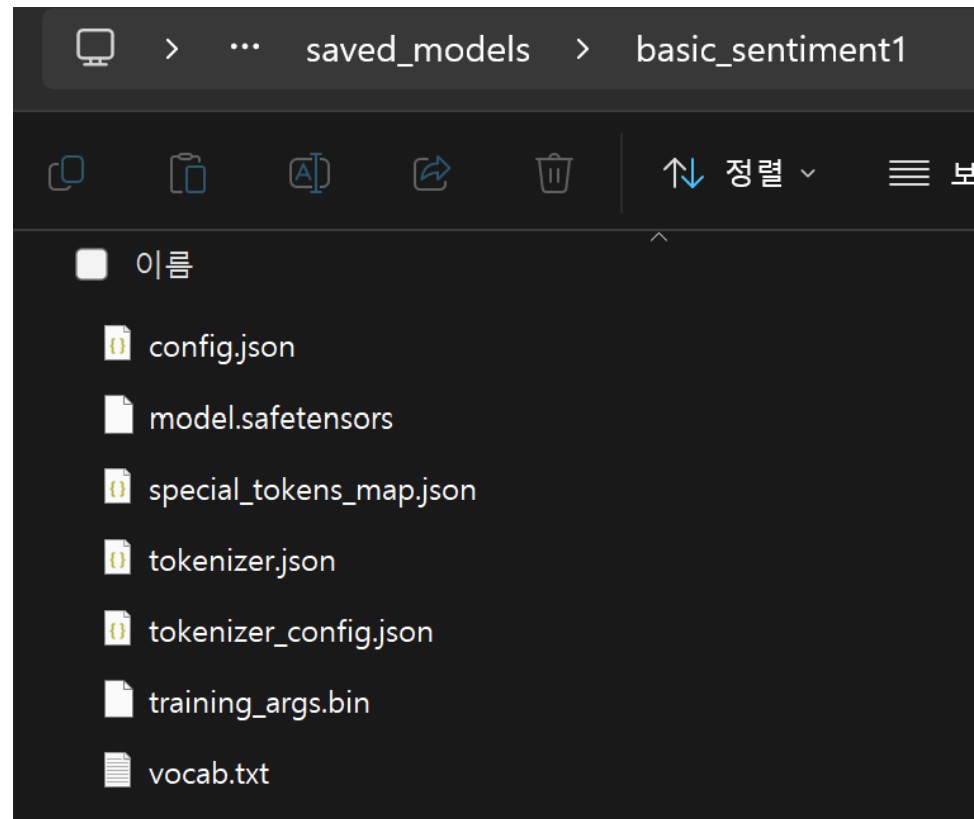
# 모델 저장

```
trainer.save_model("./saved_models/basic_sentiment1")  
tokenizer.save_pretrained("./saved_models/basic_sentiment1")
```

모델 저장

토큰라이저도 함께 저장

```
('./saved_models/basic_sentiment1\\tokenizer_config.json',  
 './saved_models/basic_sentiment1\\special_tokens_map.json',  
 './saved_models/basic_sentiment1\\vocab.txt',  
 './saved_models/basic_sentiment1\\added_tokens.json',  
 './saved_models/basic_sentiment1\\tokenizer.json')
```



# 저장된 모델 불러와서 사용

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

# 저장된 모델과 토크나이저 불러오기
model = AutoModelForSequenceClassification.from_pretrained("./saved_models/basic_sentiment1")
tokenizer = AutoTokenizer.from_pretrained("./saved_models/basic_sentiment1")

# 예측 텍스트 준비
test_texts = ["이 제품 너무 좋아요!", "별로예요. 추천 안함."]

# 토크나이징 및 텐서화
inputs = tokenizer(test_texts, return_tensors="pt", padding=True, truncation=True, max_length=64)

# 추론
model.eval()
with torch.no_grad():
    outputs = model(**inputs)

preds = torch.argmax(outputs.logits, dim=1)
print("예측 결과:", preds.tolist())
```

예측 결과: [1, 0]

# GEN AI 인텐시브 과정

Section 1. 파인튜닝

---

Section 1-2. 긍부정 분석 모델 풀 파인튜닝(베스트 모델 저장)

# import 트랜스포머 라이브러리 - 이전 실습과 동일

```
import numpy as np
import pandas as pd
import torch
from datasets import Dataset
from sklearn.model_selection import train_test_split
from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments
```



## Section

### 긍부정 분석 풀 모델 파인튜닝(베스트 모델 저장)

# 데이터 전처리 - 이전 실습과 동일

```
df = pd.read_csv("./data/review_data.csv", encoding='cp949')
```

```
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['labels'], random_state=0)
```

*# Huggingface Dataset 형태로 변환*

```
train_dataset = Dataset.from_pandas(train_df)
test_dataset = Dataset.from_pandas(test_df)
```

```
model_name = "beomi/kcbert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at r.bias', 'classifier.weight']  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions

```
def preprocess(data):
    res = tokenizer(data["text"], truncation=True, padding="max_length", max_length=64)
    return res
```

```
train_dataset = train_dataset.map(preprocess, batched=True)
test_dataset = test_dataset.map(preprocess, batched=True)
```

Map: 100%  16/16 [00:00<00:00, 594.66 examples/s]

Map: 100%  4/4 [00:00<00:00, 214.37 examples/s]

# 학습 환경 설정 - 베스트 모델 저장

# 4. *Trainer* 설정 - 베스트 모델 저장

```
training_args = TrainingArguments(  
    output_dir="./saved_models/basic_sentiment2",  
    eval_strategy="epoch",  
    save_strategy="epoch",  
    save_total_limit=1,  
    load_best_model_at_end=True,  
    metric_for_best_model="accuracy",  
    greater_is_better=True,  
    per_device_train_batch_size=4,  
    per_device_eval_batch_size=4,  
    num_train_epochs=3,  
    logging_dir="./logs",  
    logging_steps=10,  
    use_cpu=True  
)
```

→ 에포크 단위로 저장

→ 모델은 최대 1개만 저장

→ 베스트 모델 불러오기

→ 베스트 모델의 기준

→ 평가 기준이 높을수록 좋다는 것을 의미

→ 로깅 주기 : 10 step

## Section

공부정 분석폴 모델파인튜닝(베스트 모델 저장)

# 학습기 설정 및 학습

# 정확도 계산 함수

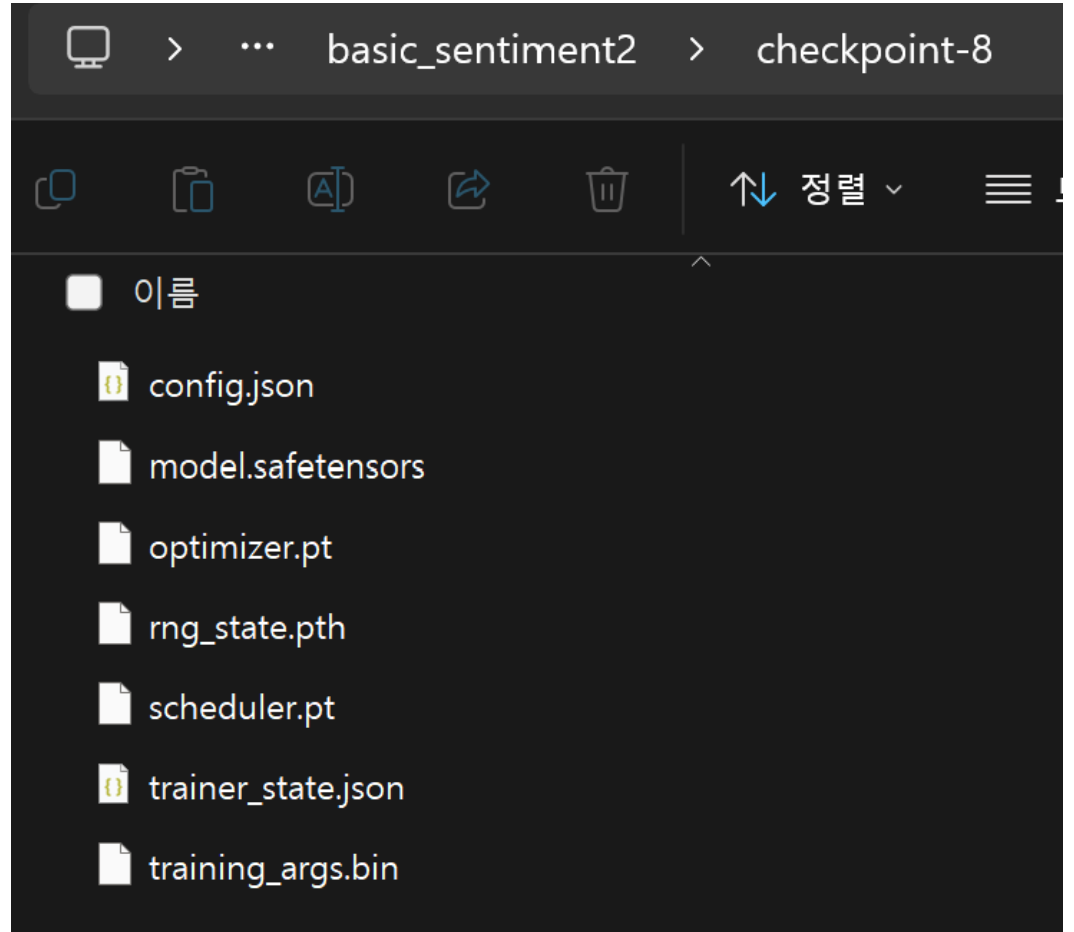
```
def compute_metrics(predict):  
    preds = np.argmax(predict.predictions, axis=1)  
    acc = (preds == predict.label_ids).mean()  
    return {"accuracy": acc}
```

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=train_dataset,  
    eval_dataset=test_dataset,  
    compute_metrics=compute_metrics,  
)
```

# 5. 학습

```
trainer.train()
```

모델 저장 옵션을 설정했으므로  
베스트 모델이 저장됨



[12/12 00:44, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
-------	---------------	-----------------	----------

1	No log	0.643477	0.500000
2	No log	0.535223	1.000000
3	0.638000	0.505539	0.750000

1 epoch에 4 step 진행되어,  
앞서 loggin\_steps로 설정한 10 step이 되지 않아 로그 안남음

2 epoch에 8 step 진행되어,  
앞서 loggin\_steps로 설정한 10 step이 되지 않아 로그 안남음

# 예측 테스트

# 6. 테스트 예측

```
test_texts = ["이 제품 너무 좋아요!", "별로예요. 추천 안함."]
inputs = tokenizer(test_texts, return_tensors="pt", padding=True, truncation=True, max_length=64)
with torch.no_grad():
    outputs = model(**inputs)
preds = torch.argmax(outputs.logits, dim=1)
print("예측 결과:", preds.tolist())
```

예측 결과: [1, 0]

## Section

긍부정 분석 풀 모델 파인튜닝(베스트 모델 저장)

# 만약 특정 체크포인트의 모델을 불러온 후 추가 학습 원할 시

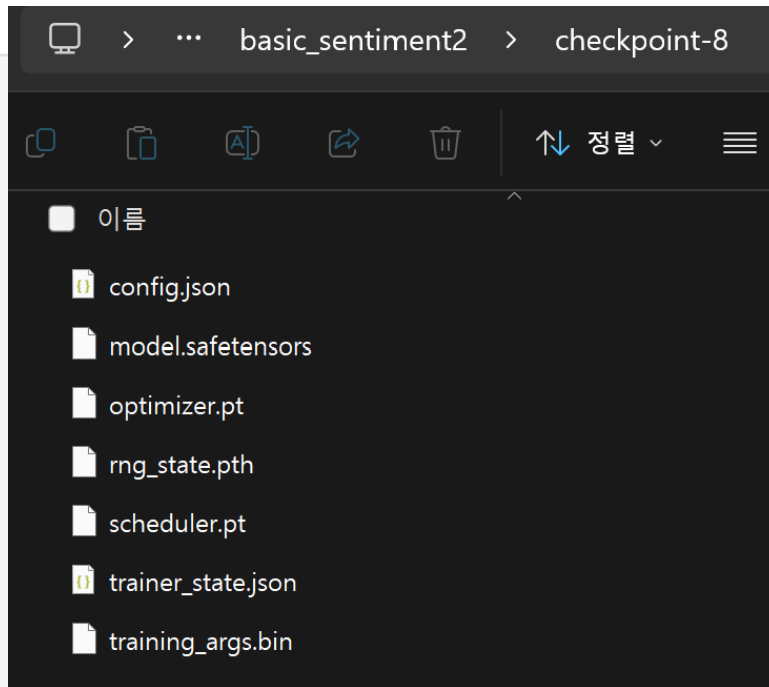
```
from transformers import Trainer
```

```
# 기존에 세팅된 trainer가 있다고 가정할 때
```

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=train_dataset,  
    eval_dataset=test_dataset,  
    compute_metrics=compute_metrics,  
)
```

```
# checkpoint-8에서 이어서 학습 시작!
```

```
trainer.train(resume_from_checkpoint="./saved_models/basic_sentiment2/checkpoint-8")
```



[12/12 00:15, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
-------	---------------	-----------------	----------

3	0.487000	0.505539	0.750000
---	----------	----------	----------



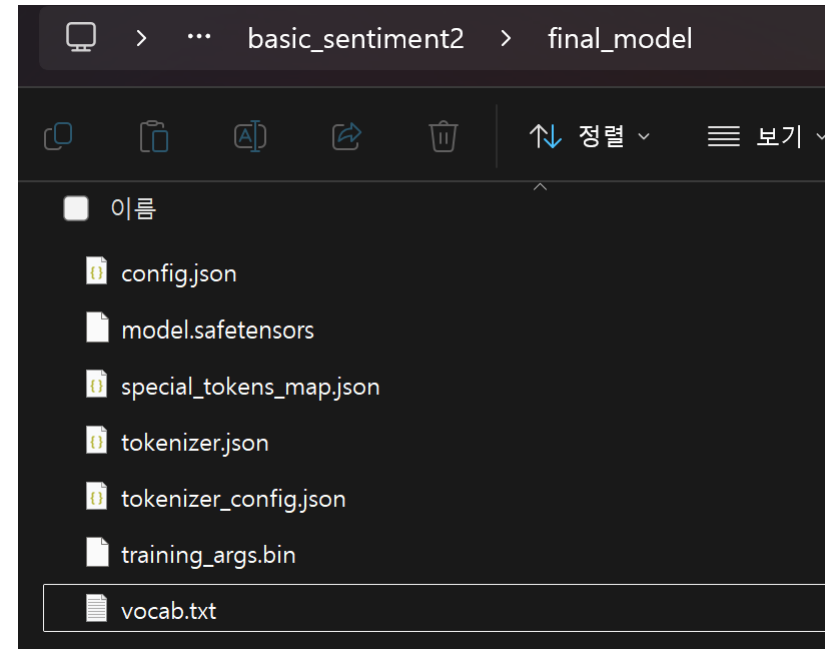
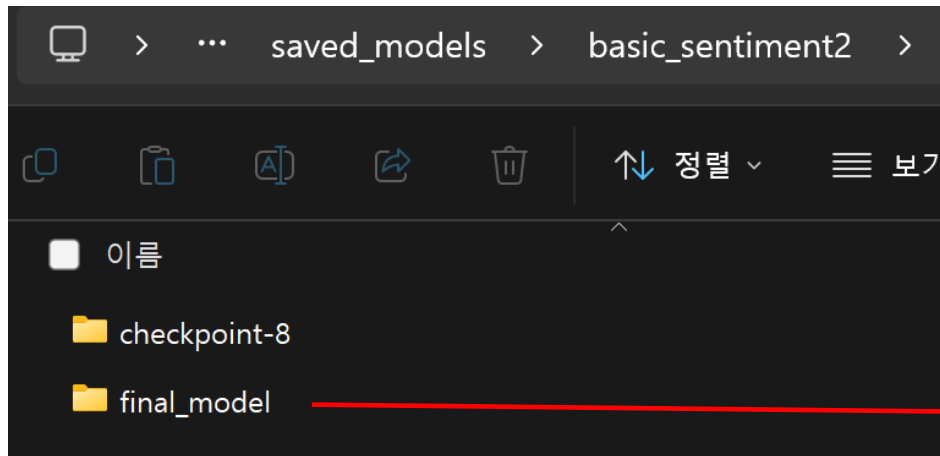
기존 모델이 에포크 2에서 저장된 모델이라서 에포크 3부터 시작

# 최종 모델 저장

# 모델 저장

```
trainer.save_model("./saved_models/basic_sentiment2/final_model")  
tokenizer.save_pretrained("./saved_models/basic_sentiment2/final_model")
```

```
('./saved_models/basic_sentiment2/final_model\\tokenizer_config.json',  
 './saved_models/basic_sentiment2/final_model\\special_tokens_map.json',  
 './saved_models/basic_sentiment2/final_model\\vocab.txt',  
 './saved_models/basic_sentiment2/final_model\\added_tokens.json',  
 './saved_models/basic_sentiment2/final_model\\tokenizer.json')
```



## Section

긍부정 분석 풀 모델 파인튜닝(베스트 모델 저장)

# 최종 모델 불러와서 예측

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

# 저장된 모델과 토크나이저 불러오기
model = AutoModelForSequenceClassification.from_pretrained("./saved_models/basic_sentiment2/final_model")
tokenizer = AutoTokenizer.from_pretrained("./saved_models/basic_sentiment2/final_model")

# 예측 텍스트 준비
test_texts = ["이 제품 너무 좋아요!", "별로예요. 추천 안함."]

# 토크나이징 및 텐서화
inputs = tokenizer(test_texts, return_tensors="pt", padding=True, truncation=True, max_length=64)

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

preds = torch.argmax(outputs.logits, dim=1)
print("예측 결과:", preds.tolist())
```

예측 결과: [1, 0]

# GEN AI 인텐시브 과정

Section 1. 파인튜닝

---

Section 1-3. LoRA 실습



# import 라이브러리

```
[1]: import numpy as np
import pandas as pd
import torch
from datasets import Dataset
from sklearn.model_selection import train_test_split
from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments
from peft import get_peft_model, LoraConfig, TaskType
```

transformers 모델에 PEFT 기법을 적용하는 함수

LoRA 설정

Task(작업 종류) 설정

# 데이터 불러오기 모델 설정

```
[2]: df = pd.read_csv("./data/review_data.csv", encoding='cp949')  
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['labels'], random_state=0)
```

```
[3]: train_dataset = Dataset.from_pandas(train_df)  
test_dataset = Dataset.from_pandas(test_df)
```

```
[4]: model_name = "beomi/kcbert-base"  
tokenizer = AutoTokenizer.from_pretrained(model_name)  
base_model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at r.bias', 'classifier.weight']  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions

# LoRA 설정 및 적용

```
[5]: # PEFT 설정 (LoRA)
peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS, # 시퀀스 분류 작업
    r=8,
    lora_alpha=16,
    lora_dropout=0.1,
    bias="none",
    target_modules=["query", "value"],
)
```

```
•[6]: # LoRA 적용
model = get_peft_model(base_model, peft_config)
```

 class `peft.TaskType`

[<source>](#)

```
( value, names = None, module = None, qualname = None,
type = None, start = 1 )
```

Enum class for the different types of tasks supported by PEFT.

Overview of the supported task types:

- SEQ\_CLS: Text classification. 텍스트 분류
- SEQ\_2\_SEQ\_LM: Sequence-to-sequence language modeling.
- CAUSAL\_LM: Causal language modeling.
- TOKEN\_CLS: Token classification.
- QUESTION\_ANS: Question answering.
- FEATURE\_EXTRACTION: Feature extraction. Provides the hidden states which can be used as embeddings or features for downstream tasks.

# LoRA 설정 및 적용

```
[5]: # PEFT 설정 (LoRA)
peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS, # 시퀀스 분류 작업
    r=8,
    lora_alpha=16, # LoRA에서 사용하는 low-rank 행렬의 rank 설정
    lora_dropout=0.1,
    bias="none",
    target_modules=["query", "value"],
)
```

```
•[6]: # LoRA 적용
model = get_peft_model(base_model, peft_config)
```

$$\begin{array}{ccc} n \times d & d \times (h \cdot d_h) & d \times (h \cdot d_h) \\ X(W_Q + \Delta W) = Q \\ \downarrow \\ W_Q + BA & , r \ll \min(d, h \cdot d_h) \end{array}$$

$W_Q$  (4x8 grid of blue squares) +  $B$  (4x2 grid of red squares)  $\times$   $A$  (2x8 grid of yellow squares)

$d \times (h \cdot d_h)$        $d \times r$        $r \times (h \cdot d_h)$

# LoRA 설정 및 적용

```
[5]: # PEFT 설정 (LoRA)
peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS, # 시퀀스 분류 작업
    r=8,
    lora_alpha=16,
    lora_dropout=0.1, # 학습 안정성 조절 파라미터
    bias="none",
    target_modules=["query", "value"],
)
```

```
•[6]: # LoRA 적용
model = get_peft_model(base_model, peft_config)
```

$$W_Q + BA$$

안정적인 학습을 위해 스케일러 적용

$$W_Q + \frac{lora\_alpha}{r} BA$$

LoRA에서 사용하는 low-rank 행렬의 rank 설정

lora\_alpha를 분모가 아닌 분자에 곱하는 이유

- **B, A** 행렬은 일반적으로 초기값이 거의 0에 가깝도록 초기화 되므로 매우 작은 값을 가짐.
- 따라서 **BA** 자체가 작아지고, 이를 기존 W에 더해도 값이 너무 작아서 학습이 매우 느리거나 영향력이 없을 수 있음
- 따라서 스케일링을 통해 값을 크게 만들어주는 계수 필요

# LoRA 설정 및 적용

```
[5]: # PEFT 설정 (LoRA)
peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS, # 시퀀스 분류 작업
    r=8,
    lora_alpha=16,
    lora_dropout=0.1,
    bias="none",
    target_modules=["query", "value"],
)
```

LoRA 모듈에만 적용되는 드롭아웃 확률, 0.05 ~ 0.1 사이가 자주 사용됨

기존 모델의 bias 파라미터를 학습에 포함시킬 것인지 결정

```
•[6]: # LoRA 적용
model = get_peft_model(base_model, peft_config)
```

"none": bias 파라미터 학습 안함(일반적으로 이 옵션 사용)

"all": 기존 모델의 bias까지 학습에 포함

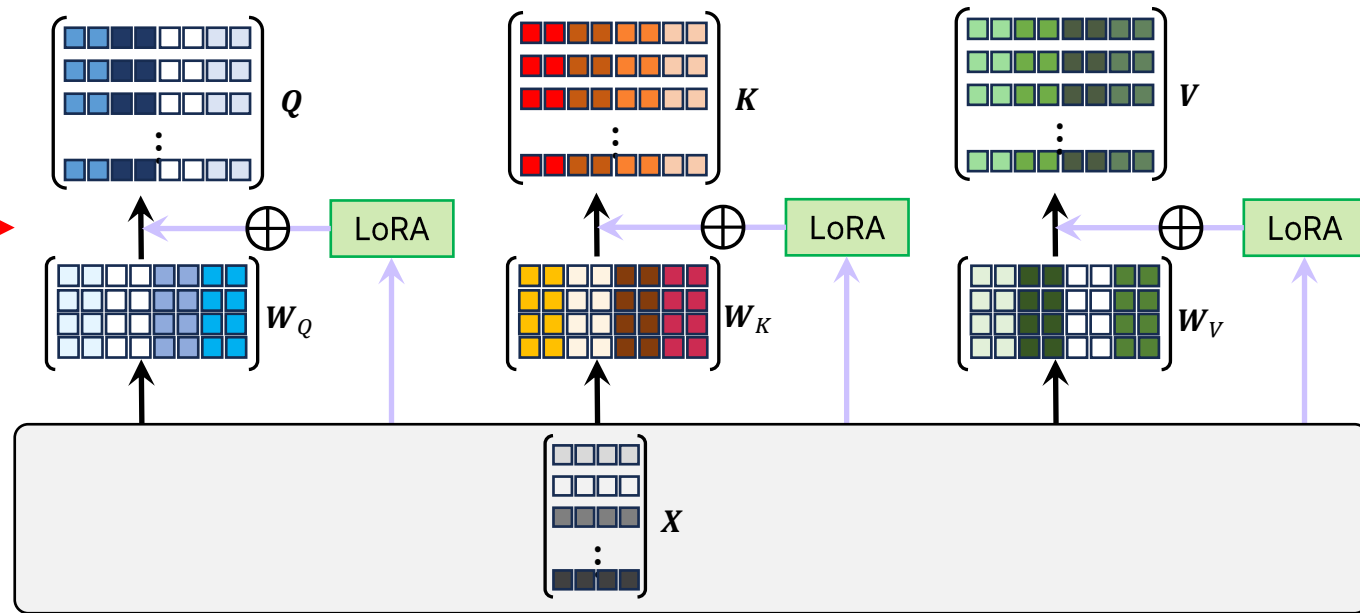
"lora\_only": LoRA가 삽입된 모듈의 bias만 학습

# LoRA 설정 및 적용

```
[5]: # PEFT 설정 (LoRA)
peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS, # 시퀀스 분류 작업
    r=8,
    lora_alpha=16,
    lora_dropout=0.1,
    bias="none",
    target_modules=["query", "value"],
)
```

Query, Key, Value 중 어디에 LoRA 적용할 지 설정

```
•[6]: # LoRA 적용
model = get_peft_model(base_model, peft_config)
```



# 데이터 전처리

```
[7]: # 전처리 함수
def preprocess(data):
    res = tokenizer(data["text"], truncation=True, padding="max_length", max_length=64)
    return res
```

```
[8]: train_dataset = train_dataset.map(preprocess, batched=True)
test_dataset = test_dataset.map(preprocess, batched=True)
```

Map: 100%  16/16 [00:00<00:00, 394.91 examples/s]

Map: 100%  4/4 [00:00<00:00, 210.45 examples/s]



# Trainer 설정

```
[9]: # Trainer 설정
training_args = TrainingArguments(
    output_dir="./saved_models/peft_lora_sentiment",
    eval_strategy="epoch",
    save_strategy="epoch",
    save_total_limit=1,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    greater_is_better=True,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    logging_dir="./logs",
    logging_steps=10,
    label_names=["labels"],
    use_cpu=True, # GPU 사용 시 False로
)
```

# 학습

```
[10]: # accuracy
def compute_metrics(predict):
    preds = np.argmax(predict.predictions, axis=1)
    acc = (preds == predict.label_ids).mean()
    return {"accuracy": acc}
```

```
[11]: trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics
)
```

```
[12]: trainer.train()
```

 [12/12 00:18, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.704548	0.500000
2	No log	0.702713	0.500000
3	0.749500	0.700424	0.500000

# 테스트

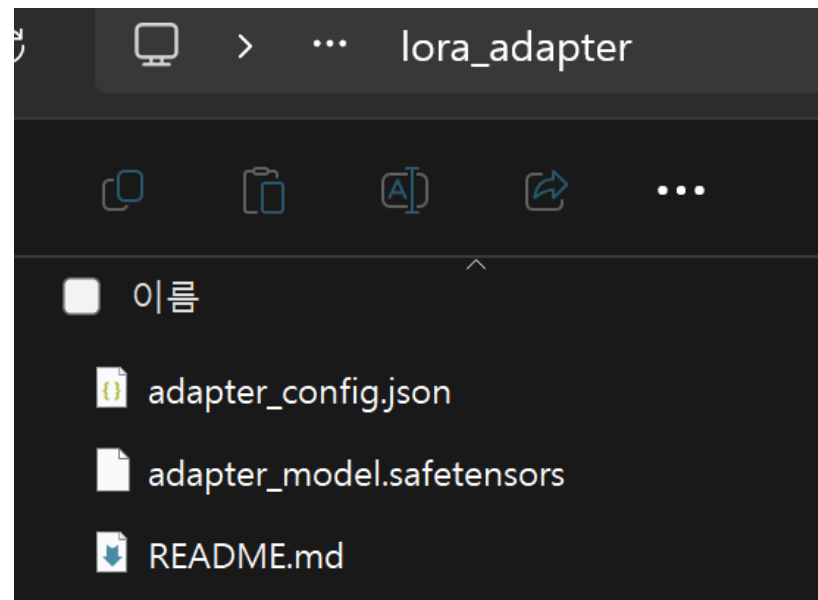
```
[13]: # 테스트 예측
test_texts = ["이 제품 너무 좋아요!", "별로예요. 추천 안함."]
inputs = tokenizer(test_texts, return_tensors="pt", padding=True, truncation=True, max_length=64)
```

```
[14]: model.eval()
with torch.no_grad():
    outputs = model(**inputs)
preds = torch.argmax(outputs.logits, dim=1)
print("예측 결과:", preds.tolist())
```

예측 결과: [1, 1]

# LoRA 어댑터만 저장하기

```
[15]: model.save_pretrained("./lora_adapter")
```



# 저장된 LoRA 어댑터 불러와서 사용

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from peft import PeftModel, PeftConfig
```

```
peft_config = PeftConfig.from_pretrained("./lora_adapter")
```

```
model_name = "beomi/kcbert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
base_model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

Some weights of BertForSequenceClassification were not initialized from the model checkpoint (beomi/kcbert-base). You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

```
model = PeftModel.from_pretrained(base_model, "./lora_adapter")
tokenizer = AutoTokenizer.from_pretrained(peft_config.base_model_name_or_path)
```

```
texts = ["이 제품 정말 좋아요!", "별로네요... 다시는 안 사요."]
inputs = tokenizer(texts, return_tensors="pt", padding=True, truncation=True, max_length=64)
```

```
model.eval()
with torch.no_grad():
    outputs = model(**inputs)
preds = torch.argmax(outputs.logits, dim=1)

print("예측 결과:", preds.tolist())
```

예측 결과: [1, 1]

감사합니다.

Q & A