nlp 과제

dataset.py

dataset.py에서는 텍스트 분류 모델에 따라 두 가지 형태의 데이터셋 클래스를 정의하고 있습니다.

SentimentDataset

사용 모델: BERT, GPT

- Hugging Face의 Tokenizer(BERT, GPT2 등)와 함께 사용되는 데이터셋 클래스입니다.
- `tokenizer`를 통해 입력 문장을 subword 단위로 토큰화하고 `[CLS]`, `[SEP]` 등의 특수 토큰을 포함한 후, padding 및 truncation을 적용합니다.
- 최종적으로 BERT/GPT 모델에 필요한 입력 형식인 `input_ids`, `attention_mask`, `token_type_ids` 등의 텐서를 자동 생성합니다.
- 각 아이템은 딕셔너리 형태로 반환되며, 키는 BERT/GPT 모델이 요구하는 포맷에 맞게 되어 있습니다.

```
{
  'input_ids': tensor([...]),
  'attention_mask': tensor([...]),
  'token_type_ids': tensor([...]),  # GPT는 없음
  'labels': tensor(0 or 1)
}
```

VocabDataset

사용 모델: RNN, LSTM

- 사전학습 토크나이저가 아닌, 사용자가 정의한 `vocab`과 간단한 `tokenizer` (예: 띄어쓰기 기준 split)를 사용한다.
- 단어 수준의 임베딩을 위해 고정된 vocabulary를 생성하며, 각 문장을 vocab index로 구성된 시퀀스로 변환된다.
- `<pad>`와 `<unk>` 토큰을 통해 길이 맞춤과 미등록 단어 처리도 수행한다.
- 각 아이템은 PyTorch 모델 입력을 위한 딕셔너리 형태로 반환되며, key는 `"input_ids"`와 `"labels"`만 포함된다.

```
{
  'input_ids': tensor([...]),  # vocab index 기반 시퀀스
  'labels': tensor(0 or 1)
}
```

정리

모델 종류	사용 Dataset 클래스	토크나이저	입력 형식
BERT	SentimentDataset	HuggingFace BERT tokenizer	input_ids, attention_mask, token_type_ids, labels
GPT	SentimentDataset	HuggingFace GPT2 tokenizer	input_ids, attention_mask, labels *(pad_token 설정 필요)*
RNN	VocabDataset	단순 tokenizer (띄어쓰기 등)	input_ids, labels
LSTM	VocabDataset	단순 tokenizer (띄어쓰기 등)	input_ids, labels

이처럼 Transformer 계열(BERT, GPT)은 Hugging Face tokenizer 기반의 SentimentDataset을, 순환신경망 계열(RNN, LSTM)은 직접 vocab을 구성하고 index로 변환하는 VocabDataset을 사용합니다.

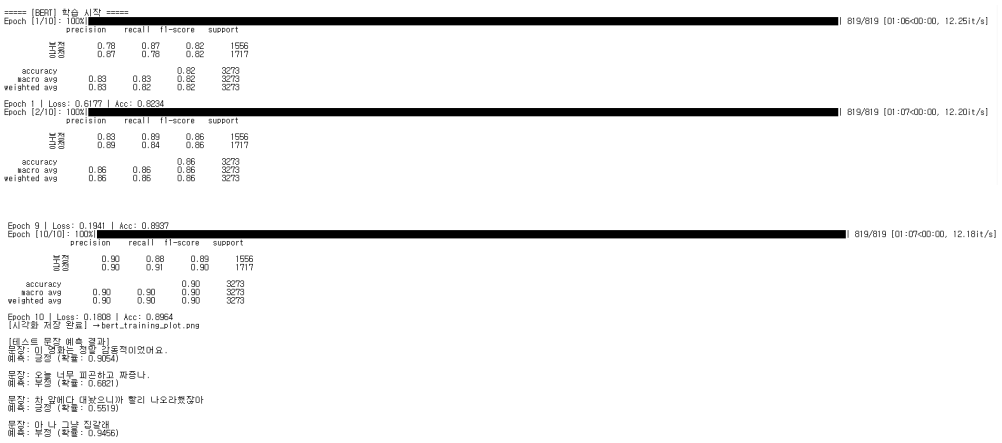
BERT

```
#  bert_classifier.py
import torch
import torch.nn as nn
import torch.nn.functional as F
from transformers import BertModel

class CustomBertClassifier(nn.Module):
    def __init__(self, dropout_rate=0.3, hidden_size=768, num_labels=2):
        super(CustomBertClassifier, self).__init__()
        self.bert = BertModel.from_pretrained("kykim/bert-kor-base")
        self.classifier = nn.Sequential(
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.Linear(hidden_size, num_labels)
        )

    def forward(self, input_ids, attention_mask=None, token_type_ids=None, labels=None):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)
        pooled_output = outputs.pooler_output
        logits = self.classifier(pooled_output)
        loss = None
        if labels is not None:
            loss = F.cross_entropy(logits, labels.to(logits.device))
        return type('Output', (), {'loss': loss, 'logits': logits})()
```

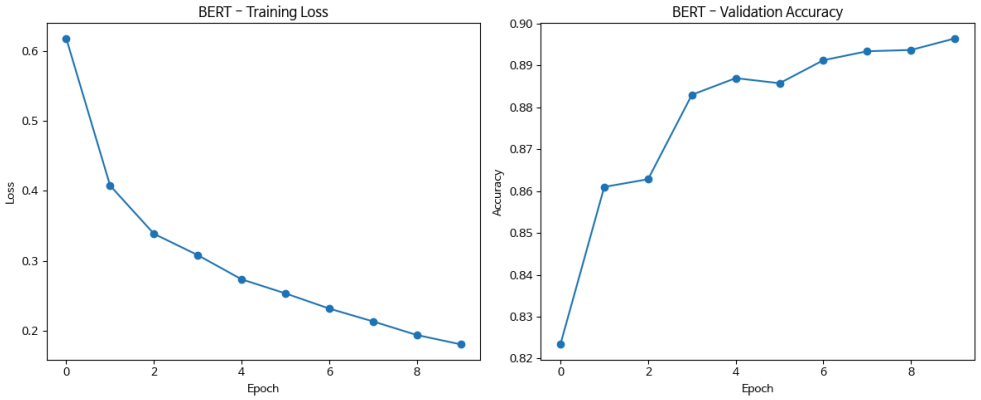
이 코드는 한국어 문장 분류를 위해 사전학습된 BERT 모델을 기반으로 한 커스텀 분류기 모델을 정의하고 있다. CustomBertClassifier 클래스는 PyTorch의 nn.Module을 상속받아 작성되었으며, 생성자에서는 huggingface의 kykim/bert-kor-base라는 한국어 BERT 모델을 불러와 self.bert로 초기화하고, 그 위에 두 개의 선형 계층과 ReLU, Dropout으로 구성된 간단한 분류기(self.classifier)를 쌓는다. 이 분류기는 BERT의 [CLS] 토큰에 해당하는 pooler_output을 입력으로 받아 최종적으로 이진 분류 결과를 출력하도록 설계되어 있다.

forward 함수에서는 입력으로 들어온 input_ids, attention_mask, token_type_ids를 BERT에 넣고, 그 결과로 얻은 pooler_output을 분류기에 통과시켜 logits를 얻는다. 만약 정답 레이블(labels)이 함께 주어진 경우에는 cross entropy loss를 계산하고, 그렇지 않다면 loss 없이 logits만 반환한다. 최종적으로 loss와 logits을 익명 객체로 묶어 반환하게 되어 있으며, 이 구조는 학습과 추론 모두에 사용 가능하다. 전체적으로 이 모델은 문장 수준의 감정 분류나 긍정/부정 분류 등 다양한 텍스트 분류 작업에 활용할 수 있다.

BERT 성능



BERT plot

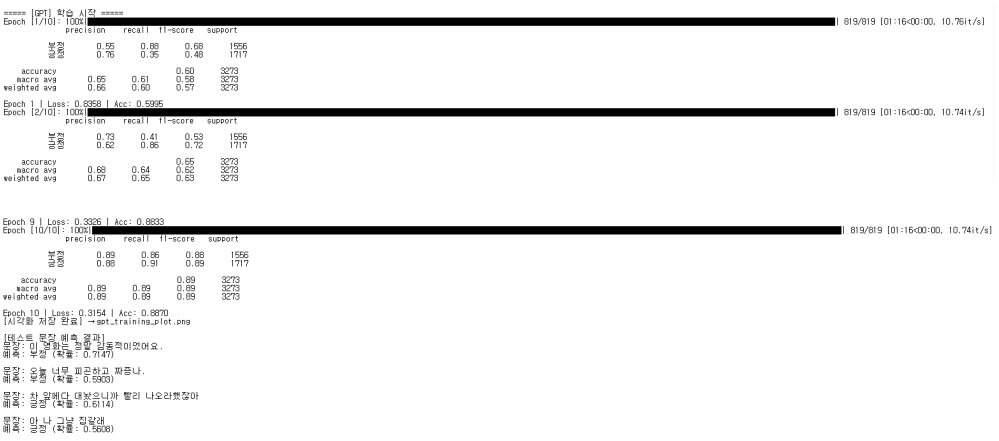


GPT2

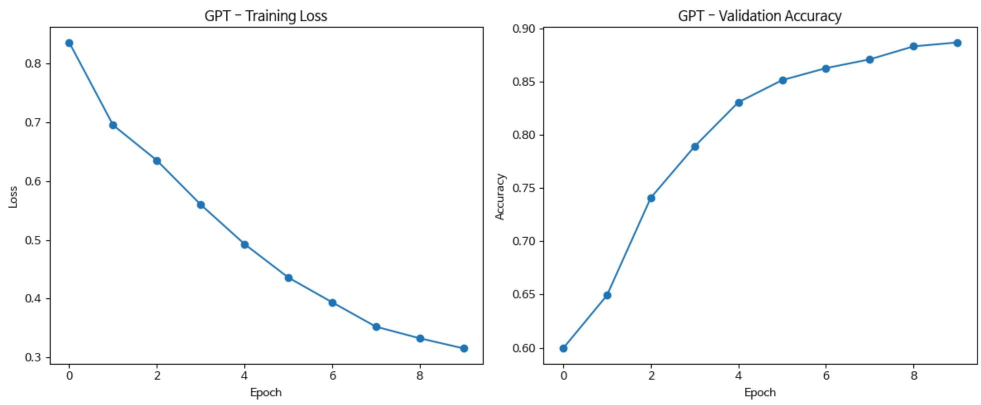
```
import torch
from transformers import GPT2Tokenizer, GPT2ForSequenceClassification

def get_gpt_model(num_labels=2):
    tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
    tokenizer.pad_token = tokenizer.eos_token  # GPT2는 pad_token이 없음 → eos_token 사용

    model = GPT2ForSequenceClassification.from_pretrained(
        "gpt2",
        num_labels=num_labels,
        pad_token_id=tokenizer.pad_token_id
    )
    return model, tokenizer
```

이 코드는 GPT-2 모델을 기반으로 한 문장 분류 모델을 설정하는 함수 `get_gpt_model`을 정의하고 있다. Hugging Face의 `GPT2ForSequenceClassification`을 사용하여 이진 분류를 위한 모델을 불러오며, tokenizer와 함께 반환된다. GPT-2는 기본적으로 pad token이 정의되어 있지 않기 때문에, 이를 해결하기 위해 tokenizer의 pad token을 eos token으로 설정하고 이를 모델에도 적용한다. 최종적으로 분류할 클래스 수를 지정할 수 있도록 `num_labels` 파라미터를 받아, 다양한 분류 문제에 사용할 수 있는 GPT-2 기반 모델과 tokenizer를 함께 반환한다.

GPT2 성능



GPT2 plot



LSTM

```
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim=128, hidden_dim=128, num_labels=2):
        super(LSTMClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, num_labels)

    def forward(self, input_ids, attention_mask=None, token_type_ids=None, labels=None):
        embedded = self.embedding(input_ids)
        _, (hidden, _) = self.lstm(embedded)
        logits = self.fc(hidden[-1])
        loss = None
        if labels is not None:
            loss = F.cross_entropy(logits, labels)
        return type('Output', (), {'loss': loss, 'logits': logits})()
```

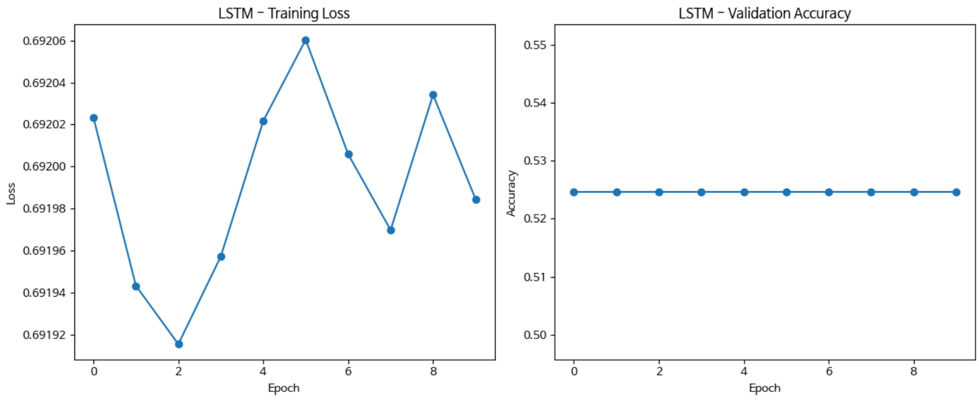
이 코드는 LSTM 기반 문장 분류 모델인 `LSTMClassifier` 를 정의한 것으로, 입력 문장을 정수 인덱스로 표현한 `input_ids` 를 받아 감정 분석이나 이진 분류 등의 작업을 수행할 수 있게 한다. 생성자에서는 단어 임베딩을 위한 `nn.Embedding` , 순차 데이터를 처리하는 `nn.LSTM` , 최종 분류를 위한 선형 계층 `nn.Linear` 를 차례로 초기화하며, LSTM은 입력 시퀀스를 처리한 뒤 마지막 hidden state를 추출해 분류기로 전달한다. `forward` 함수에서는 임베딩된 입력을 LSTM에 통과시킨 후, 마지막 hidden state를 기반으로 분류 결과 logits을 계산하고, 정답 레이블이 있을 경우 cross entropy loss도 함께 계산하여 반환한다. 이 모델은 사전학습 모델 없이 비교적 가볍게 구성된 순환신경망 기반 분류기로, 작은 규모의 텍스트 분류 작업에 적합하다.

LSTM 성능

```
===== [LSTM] 학습 시작 =====
Epoch [1/10]: 100%|████████████████████████████████████████| 819/819 [00:02<00:00, 350.63it/s]
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

          부정       0.00      0.00      0.00      1556
          긍정       0.52      1.00      0.69      1717

    accuracy                           0.52      3273
   macro avg       0.26      0.50      0.34      3273
weighted avg       0.28      0.52      0.36      3273
```

```
Epoch 9 | Loss: 0.6920 | Acc: 0.5246
Epoch [10/10]: 100%|██████████████████████████████████████| 819/819 [00:02<00:00, 344.35it/s]
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

          부정       0.00      0.00      0.00      1556
          긍정       0.52      1.00      0.69      1717

    accuracy                           0.52      3273
   macro avg       0.26      0.50      0.34      3273
weighted avg       0.28      0.52      0.36      3273

Epoch 10 | Loss: 0.6920 | Acc: 0.5246
[시각화 저장 완료] → lstm_training_plot.png

[테스트 문장 예측 결과]
문장: 이 영화는 정말 감동적이었어요.
예측: 긍정 (확률: 0.5241)

문장: 오늘 너무 피곤하고 짜증나.
예측: 긍정 (확률: 0.5241)

문장: 차 앞에다 대놨으니까 빨리 나오라했잖아
예측: 긍정 (확률: 0.5241)

문장: 아 나 그냥 집갈래
예측: 긍정 (확률: 0.5241)
```

LSTM plot

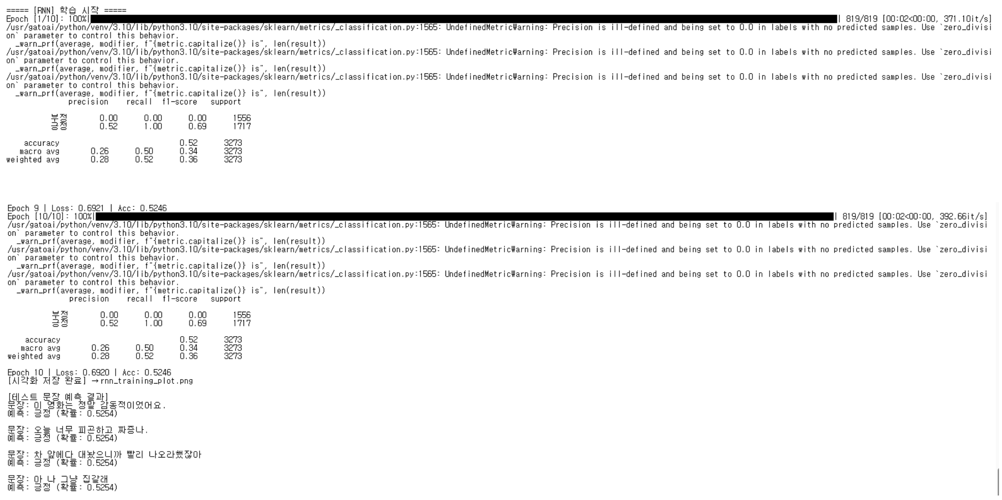


RNN

```
class RNNClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim=128, hidden_dim=128, num_labels=2):
        super(RNNClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.rnn = nn.RNN(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, num_labels)

    def forward(self, input_ids, attention_mask=None, token_type_ids=None, labels=None):
        embedded = self.embedding(input_ids)
        _, hidden = self.rnn(embedded)
        logits = self.fc(hidden[-1])
        loss = None
        if labels is not None:
            loss = F.cross_entropy(logits, labels)
        return type('Output', (), {'loss': loss, 'logits': logits})()
```

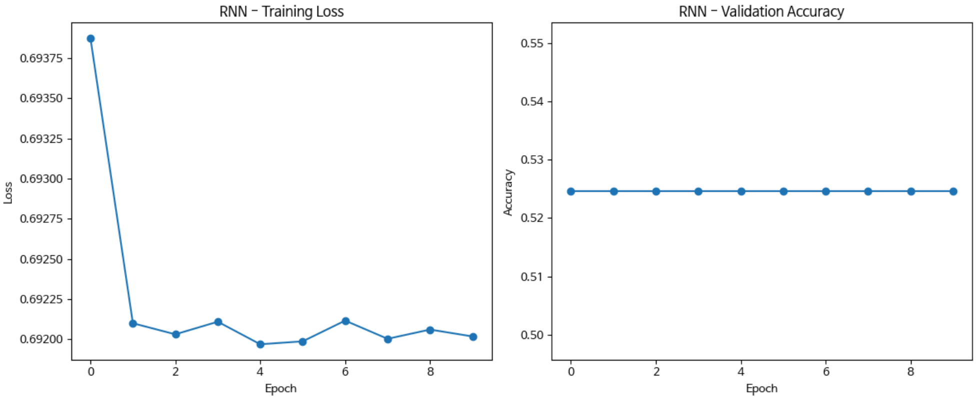
이 코드는 RNN 기반 문장 분류 모델인 `RNNClassifier` 를 정의한 것으로, 입력 문장을 정수 인덱스로 표현한 `input_ids` 를 받아 텍스트 분류를 수행한다. 모델은 단어 인덱스를 고정된 차원의 벡터로 변환하는 임베딩 층 `nn.Embedding` , 순차적인 문장 정보를 처리하는 순환 신경망 `nn.RNN` , 그리고 최종 분류를 위한 선형 계층 `nn.Linear` 로 구성된다. `forward` 함수에서는 입력을 임베딩한 뒤 RNN을 통과시키고, 마지막 hidden state를 추출해 분류기로 전달하여 logits을 계산한다. 레이블이 주어진 경우 cross entropy loss를 함께 계산하며, 학습과 추론 모두에 사용할 수 있도록 loss와 logits을 함께 반환한다. 이 모델은 구조적으로 LSTM보다 간단하며, 상대적으로 적은 파라미터로도 기본적인 문장 분류 작업을 수행할 수 있는 경량 모델이다.

RNN 성능



```
===== [RNN] 학습 시작 =====
Epoch [1/10]: 100%|██████████| 819/819 [00:02<00:00, 371.10it/s]
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

          부정       0.00      0.00      0.00      1556
          긍정       0.52      1.00      0.69      1717

    accuracy                           0.52      3273
   macro avg       0.26      0.50      0.34      3273
weighted avg       0.28      0.52      0.36      3273


Epoch 9 | Loss: 0.6921 | Acc: 0.5246
Epoch [10/10]: 100%|██████████| 819/819 [00:02<00:00, 392.66it/s]
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/gatoai/python/venv/3.10/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

          부정       0.00      0.00      0.00      1556
          긍정       0.52      1.00      0.69      1717

    accuracy                           0.52      3273
   macro avg       0.26      0.50      0.34      3273
weighted avg       0.28      0.52      0.36      3273

Epoch 10 | Loss: 0.6920 | Acc: 0.5246
[시각화 저장 완료] → rnn_training_plot.png

[테스트 문장 예측 결과]
문장: 이 영화는 정말 감동적이었어요.
예측: 긍정 (확률: 0.5254)

문장: 오늘 너무 피곤하고 짜증나.
예측: 긍정 (확률: 0.5254)

문장: 차 앞에다 대놨으니까 빨리 나오라했잖아
예측: 긍정 (확률: 0.5254)

문장: 아 나 그냥 집갈래
예측: 긍정 (확률: 0.5254)
```

RNN plot



---

main.py

이 코드는 감정 분석을 위해 BERT, GPT, LSTM, RNN 네 가지 모델을 한 번에 실험할 수 있게 구성한 통합 실행 스크립트다. 먼저 `train.csv` 파일에서 텍스트와 감정(label) 데이터를 불러오고, 중립(neutral) 감정은 제외해서 긍정과 부정만 분류하는 이진 분류 문제로 바꾼다. 그런 다음 데이터를 학습용과 검증용으로 나누고, 모델 학습에 사용할 디바이스도 GPU가 가능하면 GPU로 설정한다.

모델은 각각 구조에 맞게 정의되는데, BERT랑 GPT는 Hugging Face에서 제공하는 사전학습 모델을 불러오고 그에 맞는 tokenizer도 같이 설정한다. LSTM이랑 RNN은 직접 vocab을 구성해서 학습하는 방식이라 `VocabDataset`을 사용해서 단어 단위로 토큰화하고 인덱싱한다.

학습은 `train` 함수에서 진행되는데, 에폭마다 DataLoader로 데이터를 받아서 모델을 업데이트하고, `evaluate` 함수로 검증 정확도를 측정한다. 학습이 끝난 후엔 loss랑 정확도 추이를 그래프로 저장해주고, 마지막에는 예시 문장 몇 개를 직접 모델에 넣어서 예측 결과를 출력해준다.

`run` 함수가 전체 실행을 담당하는데, 여기에 모델 이름, 모델 객체, 학습용/검증용 DataLoader, 그리고 필요한 경우 tokenizer까지 넘겨주면 알아서 학습부터 평가, 시각화, 예측까지 전부 다 처리해준다. 실행 마지막 부분에서는 각 모델(BERT, GPT, LSTM, RNN)에 맞는 설정으로 `run` 함수를 네 번 호출해서 각각의 모델 실험을 순서대로 돌린다.

전체적으로 하나의 스크립트에서 다양한 모델을 통일된 방식으로 학습하고 비교할 수 있도록 잘 정리된 구조다.