실습 코드를 기반으로 똑같이 LSTM 기반 학습 코드를 구현하여 10 epoch training에 대한 train / test 결과를 확인하세요. (20)

```
 Epoch 1/10
Training: 100%|██████████| 391/391 [00:08<00:00, 43.87it/s, loss=0.706]
Epoch 1 Summary - Avg Loss: 0.6932, Train Acc: 0.5064, Test Acc: 0.5038

 Epoch 2/10
Training: 100%|██████████| 391/391 [00:08<00:00, 45.05it/s, loss=0.676]
Epoch 2 Summary - Avg Loss: 0.6579, Train Acc: 0.6098, Test Acc: 0.5276

 Epoch 3/10
Training: 100%|██████████| 391/391 [00:08<00:00, 47.38it/s, loss=0.515]
Epoch 3 Summary - Avg Loss: 0.6119, Train Acc: 0.6470, Test Acc: 0.6026

 Epoch 4/10
Training: 100%|██████████| 391/391 [00:09<00:00, 40.24it/s, loss=0.574]
Epoch 4 Summary - Avg Loss: 0.5527, Train Acc: 0.7104, Test Acc: 0.5299

 Epoch 5/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.97it/s, loss=0.49]
Epoch 5 Summary - Avg Loss: 0.5075, Train Acc: 0.7400, Test Acc: 0.6811

 Epoch 6/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.26it/s, loss=0.562]
Epoch 6 Summary - Avg Loss: 0.4660, Train Acc: 0.7675, Test Acc: 0.7085

 Epoch 7/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.57it/s, loss=0.291]
Epoch 7 Summary - Avg Loss: 0.4576, Train Acc: 0.7524, Test Acc: 0.7157

 Epoch 8/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.85it/s, loss=0.277]
Epoch 8 Summary - Avg Loss: 0.3553, Train Acc: 0.8464, Test Acc: 0.7652

 Epoch 9/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.17it/s, loss=0.445]
Epoch 9 Summary - Avg Loss: 0.3093, Train Acc: 0.8748, Test Acc: 0.7005

 Epoch 10/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.23it/s, loss=0.346]
Epoch 10 Summary - Avg Loss: 0.2594, Train Acc: 0.9032, Test Acc: 0.7810
```

Glove (6B, 300dim) pretrained word embedding 모델을 적용하여 성능을 확인하세요. (20)

```python
import torchtext
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleLSTMClassifier(vocab_size=len(vocab), embedding_dim = 300, hidden_dim =
128).to(device)
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
vectors = torchtext.vocab.GloVe(name='6B', dim=300,cache='~/.vector_cache')
pretrained_embedding = vectors.get_vecs_by_tokens(vocab.get_itos())
model.embedding.weight.data = pretrained_embedding
```

```
~/.vector_cache/glove.6B.zip: 862MB [02:39, 5.41MB/s]
100%|██████████| 399999/400000 [00:49<00:00, 8095.18it/s]
```

```python
model = train_loop(
model=model,
train_loader=train_loader,
test_loader=test_loader,
optimizer=optimizer,
criterion=criterion,
device=device,
epochs=10
)
```

```
 Epoch 1/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.87it/s, loss=0.691]
Epoch 1 Summary - Avg Loss: 0.6858, Train Acc: 0.5363, Test Acc: 0.5538

 Epoch 2/10
Training: 100%|██████████| 391/391 [00:08<00:00, 47.42it/s, loss=0.677]
Epoch 2 Summary - Avg Loss: 0.6916, Train Acc: 0.5420, Test Acc: 0.5197

 Epoch 3/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.98it/s, loss=0.312]
Epoch 3 Summary - Avg Loss: 0.5330, Train Acc: 0.7126, Test Acc: 0.8471

 Epoch 4/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.21it/s, loss=0.301]
Epoch 4 Summary - Avg Loss: 0.2391, Train Acc: 0.9114, Test Acc: 0.8580

 Epoch 5/10
Training: 100%|██████████| 391/391 [00:08<00:00, 45.92it/s, loss=0.0623]
Epoch 5 Summary - Avg Loss: 0.1016, Train Acc: 0.9684, Test Acc: 0.8412
```


```
 Epoch 5/10
Training: 100%|██████████| 391/391 [00:08<00:00, 45.92it/s, loss=0.0623]
Epoch 5 Summary - Avg Loss: 0.1016, Train Acc: 0.9684, Test Acc: 0.8412
```


```
 Epoch 6/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.30it/s, loss=0.00606]
Epoch 6 Summary - Avg Loss: 0.0443, Train Acc: 0.9883, Test Acc: 0.8361

 Epoch 7/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.34it/s, loss=0.00195]
Epoch 7 Summary - Avg Loss: 0.0198, Train Acc: 0.9958, Test Acc: 0.8232

 Epoch 8/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.56it/s, loss=0.00196]
Epoch 8 Summary - Avg Loss: 0.0110, Train Acc: 0.9979, Test Acc: 0.8211

 Epoch 9/10
Training: 100%|██████████| 391/391 [00:08<00:00, 46.02it/s, loss=0.000847]
Epoch 9 Summary - Avg Loss: 0.0064, Train Acc: 0.9988, Test Acc: 0.8178

 Epoch 10/10
Training: 100%|██████████| 391/391 [00:08<00:00, 45.79it/s, loss=0.000526]
Epoch 10 Summary - Avg Loss: 0.0040, Train Acc: 0.9992, Test Acc: 0.8159
```

– epoch, learning_rate, batch_size를 고정한채 LSTM 기반 모델의 test set 성능을 87% 이상으로 올려보고 어떻게 성능을 향상시켰는지 작 성하세요. (20)

GloVe 임베딩을 학습(fine-tuning)하지 않도록 설정한다 - 핵심아이디어

```
15]  model.embedding.weight.requires_grad = False
```

```
Epoch 1/10
Training: 100%|██████████| 391/391 [00:04<00:00, 83.49it/s, loss=0.685]
Epoch 1 Summary - Avg Loss: 0.6868, Train Acc: 0.5282, Test Acc: 0.5090

Epoch 2/10
Training: 100%|██████████| 391/391 [00:04<00:00, 83.67it/s, loss=0.64]
Epoch 2 Summary - Avg Loss: 0.6722, Train Acc: 0.5817, Test Acc: 0.6434

Epoch 3/10
Training: 100%|██████████| 391/391 [00:04<00:00, 82.54it/s, loss=0.586]
Epoch 3 Summary - Avg Loss: 0.6148, Train Acc: 0.6869, Test Acc: 0.7507

Epoch 4/10
Training: 100%|██████████| 391/391 [00:04<00:00, 80.66it/s, loss=0.608]
Epoch 4 Summary - Avg Loss: 0.6000, Train Acc: 0.6981, Test Acc: 0.7208

Epoch 5/10
Training: 100%|██████████| 391/391 [00:04<00:00, 82.58it/s, loss=0.313]
Epoch 5 Summary - Avg Loss: 0.4310, Train Acc: 0.8100, Test Acc: 0.8476

Epoch 6/10
Training: 100%|██████████| 391/391 [00:04<00:00, 81.96it/s, loss=0.442]
Epoch 6 Summary - Avg Loss: 0.3340, Train Acc: 0.8591, Test Acc: 0.8600

Epoch 7/10
Training: 100%|██████████| 391/391 [00:04<00:00, 83.56it/s, loss=0.358]
Epoch 7 Summary - Avg Loss: 0.3058, Train Acc: 0.8713, Test Acc: 0.8686

Epoch 8/10
Training: 100%|██████████| 391/391 [00:04<00:00, 83.20it/s, loss=0.292]
Epoch 8 Summary - Avg Loss: 0.2835, Train Acc: 0.8819, Test Acc: 0.8684

Epoch 9/10
Training: 100%|██████████| 391/391 [00:04<00:00, 84.81it/s, loss=0.0967]
Epoch 9 Summary - Avg Loss: 0.2609, Train Acc: 0.8922, Test Acc: 0.8731

Epoch 10/10
Training: 100%|██████████| 391/391 [00:04<00:00, 83.83it/s, loss=0.129]
Epoch 10 Summary - Avg Loss: 0.2378, Train Acc: 0.9045, Test Acc: 0.8726
```

실습코드를 활용하여 naïve_transformer_model을 training 하려면 되지 않는다. training 되도록 코드를 수정하세요. (20)

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-17-464526274> in <cell line: 0>()
     75 optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
     76
---> 77 naive_transformer_model = TransformerClassifier(
     78     vocab_size=len(vocab),
     79     embedding_dim=300,

2 frames
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/activation.py in __init__(self, embed_dim, num_heads, dropout, bias, add_bias_kv, add_zero_attn, kdim, vdim, batch_first, device, dtype)
    989         self.batch_first = batch_first
    990         self.head_dim = embed_dim // num_heads
--> 991         assert self.head_dim * num_heads == self.embed_dim, "embed_dim must be divisible by num_heads"
    992
    993         if not self._qkv_same_embed_dim:

AssertionError: embed_dim must be divisible by num_heads
```

문제점-

IndexError: Dimension out of range (expected to be in range of [-1, 0], but got 1

각 헤드는 embed_dim / num_head 차원의 서브공간을 사용해야 하기 때문에, embed_dim은 num_head로 **나누어떨어져야 함**.

+ IndexError: Dimension out of range (expected to be in range of [-1, 0], but got 1) 에러는 **.squeeze(1)을 호출했는데, 해당 시점의 텐서가 1차원이라서 .squeeze(1)이 불가능해서** 생긴 오류

torch.sigmoid(self.classifier(pooled) 와 같이 squeeze를 제거 및 sigmoid

training이 되도록 고치면 성능이 형편없다. epoch, learning_rate, batch_size를 고정한채 naïve_taransformer_model 기반 모델의 test set 성능을 82% 이상으로 올려보고 어떻게 성능을 향상 시켰는지 작성하세요. (20

실행 시켜보니 다음과 같은 결과를 얻음

```
Training: 100%|██████████| 391/391 [02:40<00:00,  2.44it/s, loss=0.295]
Epoch 5 Summary - Avg Loss: 0.3117, Train Acc: 0.8606, Test Acc: 0.8364

 Epoch 6/10
Training: 100%|██████████| 391/391 [02:42<00:00,  2.41it/s, loss=0.34]
Epoch 6 Summary - Avg Loss: 0.2791, Train Acc: 0.8825, Test Acc: 0.8314

 Epoch 7/10
Training: 100%|██████████| 391/391 [02:39<00:00,  2.46it/s, loss=0.215]
Epoch 7 Summary - Avg Loss: 0.2532, Train Acc: 0.8926, Test Acc: 0.8090
```

위 방법 외에 어떤 방법을 쓰던 sentiment classification을 하여 test set 성능을 높이기 위한 모델을 개발하고 본인의 최종 모델은 무엇이 고 성능은 얼마이며 성능을 올리기 위해 어떤 것들을 수행하였는지 정리하세요. (20)

```python
# 2. BERT 토크나이저
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# 3. Dataset 클래스 정의
class IMDBDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=256):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __getitem__(self, idx):
        encoding = self.tokenizer(
            self.texts[idx],
            padding='max_length',
            truncation=True,
            max_length=self.max_len,
            return_tensors='pt'
        )
        input_ids = encoding['input_ids'].squeeze(0)
        attention_mask = encoding['attention_mask'].squeeze(0)
        label = torch.tensor(self.labels[idx], dtype=torch.float)
        return input_ids, attention_mask, label

    def __len__(self):
        return len(self.texts)

# 4. DataLoader 준비
train_dataset = IMDBDataset(train_df['text'].tolist(), train_df['label'].tolist(), tokenizer)
test_dataset = IMDBDataset(test_df['text'].tolist(), test_df['label'].tolist(), tokenizer)
```

BERT 토크나이저를 불러와 텍스트를 토큰화할 준비를 하고,MDBDataset 클래스는 입력 텍스트와 라벨을 받아 토큰화하고 텐서로 변환하며, getitem__에서는 각 인덱스별로 input_ids, attention_mask, label을 반환하고, 이렇게 만든 train/test Dataset은 DataLoader에 넣어 학습에 사용할 수 있게 된다. + 아래와 같이 모델 정의한다.

```python
# 5. BERT 기반 분류 모델 정의
class BertClassifier(nn.Module):
    def __init__(self, dropout=0.3):
        super().__init__()
        self.bert = BertModel.from_pretrained("bert-base-uncased")
        if freeze_bert:
            for param in self.bert.parameters():
                param.requires_grad = False
        self.dropout = nn.Dropout(dropout)
        self.classifier = nn.Linear(768, 1)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        last_hidden_state = outputs.last_hidden_state
        pooled = last_hidden_state.mean(dim=1)
        logits = self.classifier(self.dropout(pooled))
        return logits.squeeze(1)
```

```python
# 7. 학습 실행
if __name__ == "__main__":
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = BertClassifier(dropout=0.3)
    criterion = nn.BCEWithLogitsLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
    mymodel=train_loop(model, train_loader, test_loader, optimizer, criterion, device, epochs=3)
```

```
model.safetensors: 100%  440M/440M [00:06<00:00, 53.0MB/s]

Epoch 1/3
Training: 100%|██████████| 391/391 [19:52<00:00,  3.05s/it, loss=0.065]
Epoch 1 Summary - Avg Loss: 0.2753, Train Acc: 0.8838, Test Acc: 0.9168

Epoch 2/3
Training: 100%|██████████| 391/391 [19:48<00:00,  3.04s/it, loss=0.0819]
Epoch 2 Summary - Avg Loss: 0.1633, Train Acc: 0.9414, Test Acc: 0.9208

Epoch 3/3
Training: 100%|██████████| 391/391 [19:54<00:00,  3.06s/it, loss=0.0893]
Epoch 3 Summary - Avg Loss: 0.0935, Train Acc: 0.9688, Test Acc: 0.9175
```

약간의 과적합이 일어났지만 91%로 처음으로 90%가 넘어가는 성능을 보였다.

본인의 모델 코드 및 가중치를 저장하여 python test.py [data_dir] [model name] 으로 실행하면 해당 모델의 가중치를 불러와서 test set 으로 inference를 수행 후 test set 문장 index와 pos neg 결과를 tab으로 구분하여 result.tsv로 출력하도록 작성하세요. (40)

Model.pt 다운로드 링크

https://drive.google.com/file/d/11dCrfGgpf1TIOvd2W3duG-7eWvy0yHK2/view

	A	B
1	Index	Prediction
2	0	positive
3	1	positive
4	2	positive
5	3	positive
6	4	positive
7	5	negative
8	6	positive
[illegible]	7	positive
10	8	positive
11	9	negative
12	10	positive
13	11	positive
14	12	positive
15	13	positive
16	14	positive
17	15	positive
18	16	positive
19	17	positive
20	[illegible]	[illegible]

```
# 정확도 출력
accuracy = correct / total
print(f"Test Accuracy: {accuracy:.4f}")

# 4. 결과를 result.tsv 파일로 저장
result_df = pd.DataFrame(results, columns=["Index", "Prediction"])
result_df.to_csv("result.tsv", sep="₩t", index=False)


print("Results have been saved to result.tsv")
```