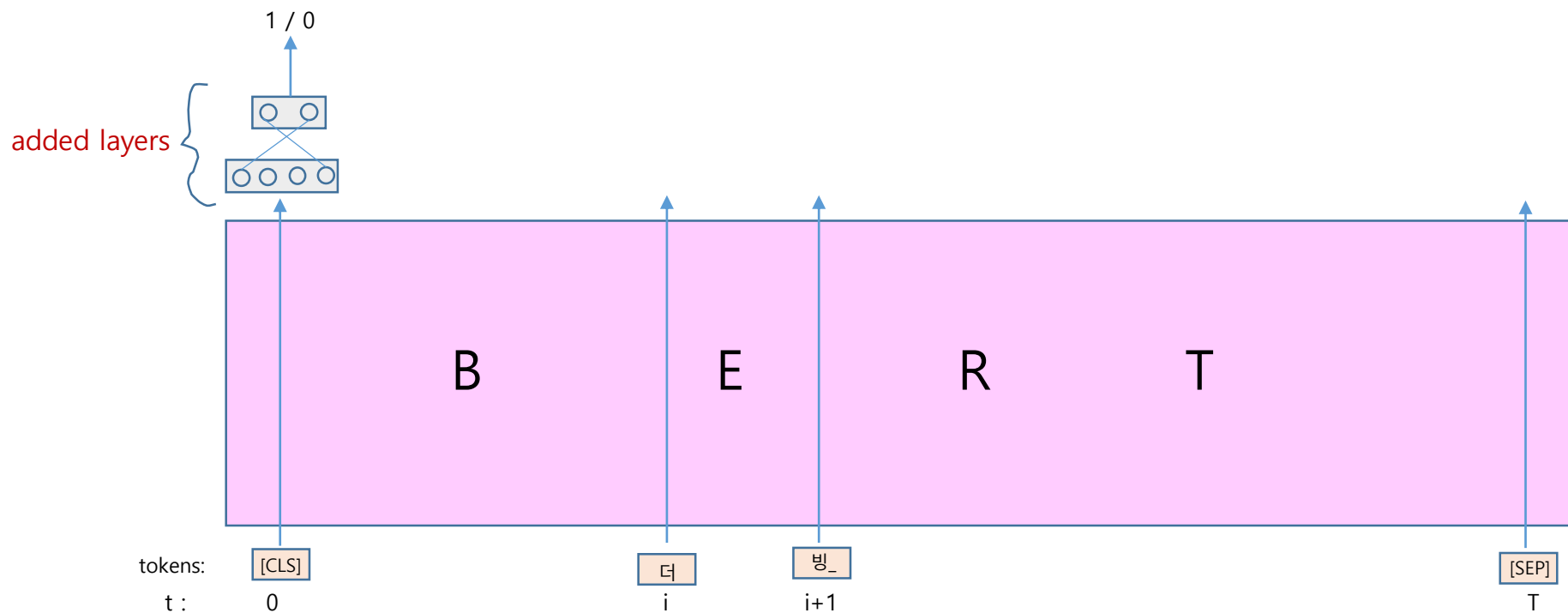


- 전반적인 내용은 다음 페이지부터 설명을 함.
- 과제에서 수행할 사항:
 - 빈 곳을 채워서 프로그램을 완성한다.
 - 실험을 통하여 가장 성능이 잘 나오는 **epoch** 를 찾는다.
 - 이 관찰에 기반하여 **number of epochs** 를 변경한다.
 - **test** 단계를 위한 **code**를 넣는다.
 - **training** 의 마지막 **epoch** 후의 **loss** 및 **validation accuracy** 에 대한 출력이 나오는 실행창을 캡처하여 보고서에 넣는다.
 - **test** 단계에서 측정한 **test accuracy** 의 출력이 나오는 실행창을 캡처하여 보고서에 넣는다.
- 제출물: 프로그램(.py 포맷), 결과보고서(.pdf 포맷).
 - **project** 디렉토리를 압축하여 제출하지 말 것.
 - 위 두 파일만 압축하지 않고 업로드 할 것.

- 목표: 감성분석
 - **Sentiment analysis**
- 이는 문서 또는 텍스트 분류의 대표적인 문제임
 - **text (document) classification** Supervised Learning, ()
- 우리의 방법:
 - **BERT** 를 사용하는 **deep learning** 모델 이용
Pre-Trained 가 Fine-Tuning
- 학습데이터:
 - 영어로 작성된 영화 감상평 (**movie review**)
- 본 과제에서 사용할 딥러닝 라이브러리: PYTORCH
- 우리는 English 에 대하여 pretrained 된 BERT 모델인 “bert-base-uncased” 를 사용함.
2018 google BERT
- 사용할 BERT 모델은 Hugging Face 사가 제공하는 transformers 패키지 안에 있는 AutoModel을 사용한다.
- 토큰나이저(tokenizer)로는 역시 transformers 안에 있는 AutoTokenizer 를 이용한다
- 이를 위해 transformers 를 설치해야 한다:
 - **"pip install transformers"** 명령을 이용한다.

- BERT 위에 FF 층을 올림.
- FF 층들과 BERT 사이에 LSTM 층을 넣을 수도 있음.
 - 보통 성능이 올라갈 것으로 예상됨.
- [CLS] 토큰을 input token sequence 의 첫 토큰으로 넣음.
- [CLS] 토큰이 입력되는 $t=0$ 의 출력만을 분류 예측(prediction)에 이용함.
 - $t=1\sim T$ 의 나머지 시간대의 출력들은 이용하지 않음. 즉 **loss** 의 결정과 **prediction** 에 이용되지 않음.



```
import numpy as np
import torch
from torch.utils.data import (DataLoader, RandomSampler, SequentialSampler,
                              TensorDataset)
from torch.utils.data.distributed import DistributedSampler
from tqdm import tqdm, trange

from torch import nn
import torch.nn.functional as F
import torch.optim
from torch.utils.data import Dataset, DataLoader
import time
from sklearn.metrics import classification_report
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler

from transformers import AutoModel, AutoTokenizer
```

- Pretrained BERT , tokenizer 객체 만들기

```
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")  
bert = AutoModel.from_pretrained("bert-base-uncased")
```

- GPU 계산장치 준비 (아래를 수정하여 device로 cpu 도 가능하게 할 것!)

```
device = torch.device("cuda")
```

- 상수 설정


```
batch_size = 8  
best_acc = 0.0           # 초기값  
num_epochs = 5  
max_seq_leng = 256
```

- 영화감상평에 긍정(1), 부정(0) 레이블을 tagging 한 labeled corpus

- 첫 줄: **comment**
- 각 줄은 3 개의 **field**이고 이들은 '\t' 로 구분되어 있음
- **id** 필드는 무시; **document** 필드에 감상평; **label** 필드의 1은 긍정, 0 은 부정을 나타냄.
 - Number of classes = 2.

- 화일명: “train_data_movie_reviews_imdb.txt”

- 각 줄에 정답레이블(1:긍정/0:부정) 과 영화감상평이 있다.
- 둘 사이는 '\t' 로 구분된다.
- 25,000 개의 감상평을 가짐 (이는 **IMDB** 데이터의 일부만 가져온 것임.)



```

1      Bromwell High is a cartoon comedy. It ran at the same time as some other programs about school life,
0      Story of a man who has unnatural feelings for a pig. Starts out with a opening scene that is a terrif
1      Homelessness (or Houselessness as George Carlin stated) has been an issue for years but never a plan
0      Airport '77 starts as a brand new luxury 747 plane is loaded up with valuable paintings & such belong
1      Brilliant over-acting by Lesley Ann Warren. Best dramatic hobo lady I have ever seen, and love scenes
0      This film lacked something I couldn't put my finger on at first: charisma on the part of the leading
1      This is easily the most underrated film inn the Brooks cannon. Sure, its flawed. It does not give a r
0      Sorry everyone,,, I know this is supposed to be an "art" film,, but wow, they should have handed out
1      This is not the typical Mel Brooks film. It was much less slapstick than most of his movies and actual
0      When I was little my parents took me along to the theater to see Interiors. It was one of many movies
  
```

- 한 감상평 text 에 대하여 하나의 훈련예제를 만들어야 함:
 - 1) Input token sequence of fixed length MSL:** 감상평에 해당하는 token sequence. Batch 를 만들 수 있도록 고정된 길이로 통일해야 함.
 - 2) attention mask:** input token sequence 의 각 token 마다 attention 에 참여할지 말지를 구분해 주어야 함
 - 3) target label :** input token sequence 마다 정답 label (index) 하나를 제공함.

한 훈련예제의 구성: 3

input token seq: (실제는 번호)	[CLS]	film	lacked	some	points	that	should	[SEP]	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]
attention mask seq:	1	1	1	1	1	1	1	1	0	0	0	0	0
target label:	1													

- input seq 에서 문장의 맨 앞과 뒤에 special token 을 넣는다:
 - [CLS], [SEP]
- [SEP] 까지의 토큰들에게는 attention mask 로 1 을 주고,
- 그 뒤의 [PAD] 토큰들에게는 attention mask 로 0 을 준다.
- target label: 1/0

training example list 구축 함수

```
def prepare_input_tok_seq_and_mask_seq_from_file (filename):
    sentences = []
    target_labels = []

    # assume that file is in the program directiory (files are naver sentiment data files).
    with open(filename, "r", encoding="utf-8") as fpr:
        i=0
        for line in fpr.readlines():
            line_splited = line.split('\t') # the result is a list of 3 strings: doc_id, sentence, label.

            sent = line_splited[1]
            sent = sent[:-1] #remove newline char
            sentences.append(sent)
            lab_str = line_splited[0] # the last char is a tab.
            lab_idx = int(lab_str)
            target_labels.append(lab_idx)
            i = i+1
            if i % 5000 == 0:
                print("number of lines read from file = ", i)

    # each sentence string in sentences is converted into a list of token ids (indices).
    sentences_id = [] # a list of sublists; a sublist is a list of indexes of tokens in a sentence.
    for i, sentence in enumerate(sentences):
        token_list = tokenizer.tokenize(sentence)
        token_id_list = tokenizer.convert_tokens_to_ids(token_list)
        sentences_id.append(token_id_list)

        if i % 5000 == 0:
            print("tokenizing sentences i", i)

    # prepare 2 separate lists having input token sequence and attention mask.
    list_tok_seq = []
    list_mask_seq = []
```



```

# 앞 페이지의 list_mask_seq 문장과 같은 깊이임.
for k, token_list in enumerate(sentences_id):
    if k % 5000 == 0:
        print("Number of reviews (for training examples) processed = ", k)

    tok_seq = np.zeros(max_seq_leng, dtype=np.int_) # this is for one sentence.
    mask_seq = np.zeros(max_seq_leng, dtype=np.int_)

    tok_seq[0] = 2 # add idx of [CLS] as first element
    mask_seq[0] = 1

    tk_leng = len(token_list)
    if tk_leng > max_seq_leng - 2:
        tk_leng1 = max_seq_leng - 2
    else:
        tk_leng1 = tk_leng

    place_sep = tk_leng1 + 1
    for i in range(tk_leng1):
        tok_seq[i + 1] = token_list[i]
        mask_seq[i + 1] = 1

    tok_seq[place_sep] = 3 # idx of token [SEP]
    mask_seq[place_sep] = 1

    if place_sep < max_seq_leng - 1:
        for j in range(place_sep + 1, max_seq_leng):
            tok_seq[j] = 0 # add pad.
            mask_seq[j] = 0 # add 0 mask.

    list_tok_seq.append(tok_seq)
    list_mask_seq.append(mask_seq)

list_tok_seq = np.array(list_tok_seq, np.int32)
list_mask_seq = np.array(list_mask_seq, np.int32)
target_labels = np.array(target_labels, np.int32)

return list_tok_seq, list_mask_seq, target_labels

```

```

list_tok_seq_all, list_mask_seq_all, target_labels_all = \
    prepare_data_loader(seq_from_file("./train_data_movie_reviews_imdb.txt"))
total_num_examples = list_tok_seq_all.shape[0]

num_train_examples = int(total_num_examples * 0.8)
num_val_examples = int(total_num_examples * 0.1)
num_test_examples = total_num_examples - num_train_examples - num_val_examples
sum_train_val = num_train_examples + num_val_examples

print("number of examples(total, validation, test): ", total_num_examples, num_train_examples, \
      num_val_examples, num_test_examples)

list_tok_seq_train, list_mask_seq_train, target_labels_train = list_tok_seq_all[:num_train_examples], \
    list_mask_seq_all[:num_train_examples], target_labels_all[:num_train_examples]

list_tok_seq_val, list_mask_seq_val, target_labels_val = list_tok_seq_all[num_train_examples:sum_train_val], \
    list_mask_seq_all[num_train_examples:sum_train_val], \
    target_labels_all[num_train_examples:sum_train_val]

list_tok_seq_test, list_mask_seq_test, target_labels_test = list_tok_seq_all[sum_train_val:], \
    list_mask_seq_all[sum_train_val:], target_labels_all[sum_train_val:]

train_tok = torch.tensor(list_tok_seq_train, dtype=torch.long)
train_mask = torch.tensor(list_mask_seq_train, dtype=torch.long)
train_y = torch.tensor(target_labels_train, dtype=torch.long)

val_tok = torch.tensor(list_tok_seq_val, dtype=torch.long)
val_mask = torch.tensor(list_mask_seq_val, dtype=torch.long)
val_y = torch.tensor(target_labels_val, dtype=torch.long)

test_tok = torch.tensor(list_tok_seq_test, dtype=torch.long)
test_mask = torch.tensor(list_mask_seq_test, dtype=torch.long)
test_y = torch.tensor(target_labels_test, dtype=torch.long)

```

```
train_data = TensorDataset(train_x, train_y)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=batch_size, drop_last=True)

val_data = TensorDataset(val_x, val_y)
val_sampler = SequentialSampler(val_data)
val_dataloader = DataLoader(val_data, sampler=val_sampler, batch_size=batch_size, drop_last=True)

test_data = TensorDataset(test_x, test_y)
test_sampler = SequentialSampler(test_data)
test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_size=batch_size, drop_last=True)
```

```
class BERT_Arch(nn.Module):

    def __init__(self, bert):
        super(BERT_Arch, self).__init__()
        self.bert = bert
        self.dropout = nn.Dropout(0.1)
        self.relu = nn.ReLU()
        self.fc1 = nn.Linear(768, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 2)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, sent_id, mask):
        outputs = self.bert(input_ids=sent_id, attention_mask=mask)
        cls_out = outputs.pooler_output → shape: (batch_sz, 768)
        x = self.fc1(cls_out)
        x1 = self.relu(x)
        x2 = self.dropout(x1)
        x3 = self.fc2(x2)
        x4 = self.relu(x3)
        x5 = self.dropout(x4)
        x6 = self.fc3(x5)
        x7 = self.softmax(x6)
        return x7

model = BERT_Arch(bert)
model = model.to(device)
```

```
optimizer = torch.optim.AdamW(model.parameters(), lr = 1e-5)  
loss_fn = nn.NLLLoss()
```

```
num_training_batches = len(train_dataloader)  
num_val_batches = len(val_dataloader)  
print("num of train and validation batches: ", num_training_batches, " , ", num_val_batches)
```

```

best_acc = 1.0
epochs = 7
num_training_batches = len(train_dataloader)
num_val_batches = len(val_dataloader)
print("num of train and validation batches: ", num_training_batches, num_val_batches)

for epoch in range(epochs):
    # Training is done for this epoch.
    model.train()
    total_loss, total_accuracy = 0, 0
    for step, batch in enumerate(train_dataloader):
        batch = [r.to(device) for r in batch]
        sent_id, mask, labels = batch
        model.zero_grad()
        preds = model(sent_id, mask)
        loss = loss_fn(preds, labels) # "reduce=True" is default.
        total_loss = total_loss + loss.item()

        loss.backward()

        optimizer.step()

        if step != 0 and step % 400 == 0:
            print("Epoch:", epoch, " batch:", step, " has finished.")

    avg_loss = total_loss / len(train_dataloader)

    # evaluate model after each epoch
    model.eval()
    hit_cnt = 0
    total_cnt = 0

    #with torch.no_grad():

```

```

with torch.no_grad():
    for step, batch in enumerate(val_dataloader):
        batch = [r.to(device) for r in batch]
        tok_part, mask_part, y_part = batch

        # model prediction
        preds = model(tok_part, mask_part)

        preds_label = torch.argmax(preds, dim=1)
        preds_label = preds_label.cpu()
        preds_label = preds_label.numpy()
        y_part = y_part.cpu()
        y_part = y_part.numpy()

        for i in range(y_part.shape[0]):
            if preds_label[i] == y_part[i]:
                hit_cnt += 1
        total_cnt += len(y_part)

acc = hit_cnt/total_cnt
print("At epoch:", epoch, " avg loss=", avg_loss, " Accuracy = ", acc)

# save the best model if the model has improved.
if acc < best_acc:
    best_acc = acc
    torch.save(model.state_dict(), 'saved_weights.pt')
    print("Model parameters are saved.")

print("Training ends.")

## 여기에 test 단계를 넣어야 한다.

print("Program ends.")

```