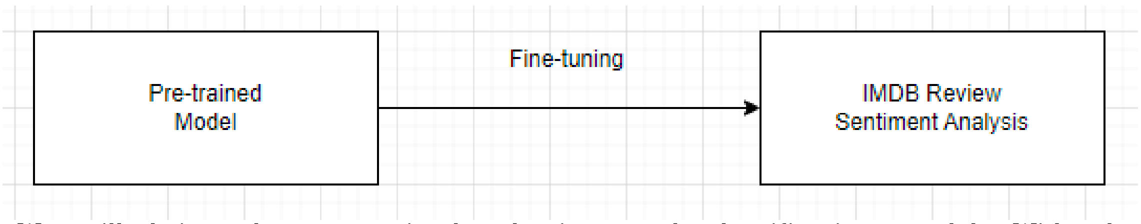
Bert Fine-Tuning

Sentiment Analysis of IMDB Movie Reviews

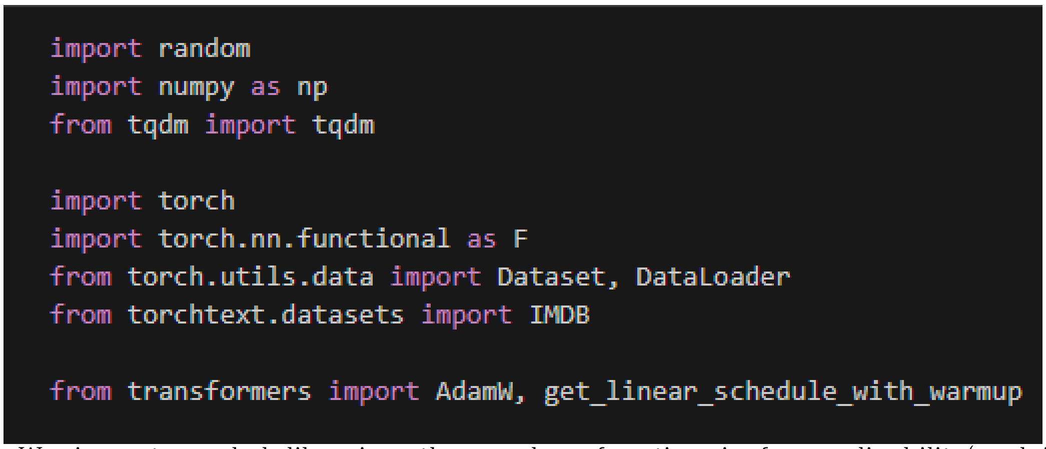
과목명	딥러닝	**학과**	컴퓨터학부
교수명	정우환	**학번**	20200052551
제출일	2024.12.19	**이름**	성주원

1. Bert fine-tuning Concept



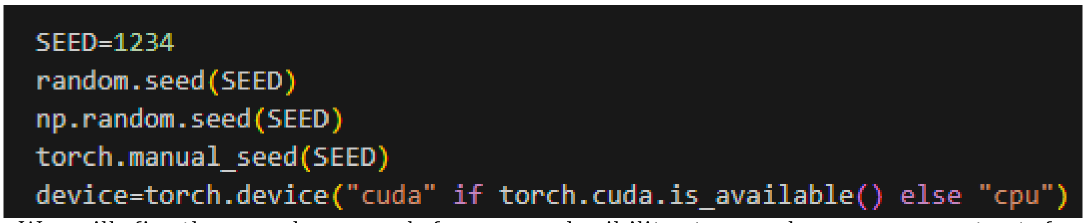
-We will bring the pre-trained tokenizer and classification model. With the fine-tuning, we will make our model classify IMDB review with sentiment analysis.

2. Source Code Analysis



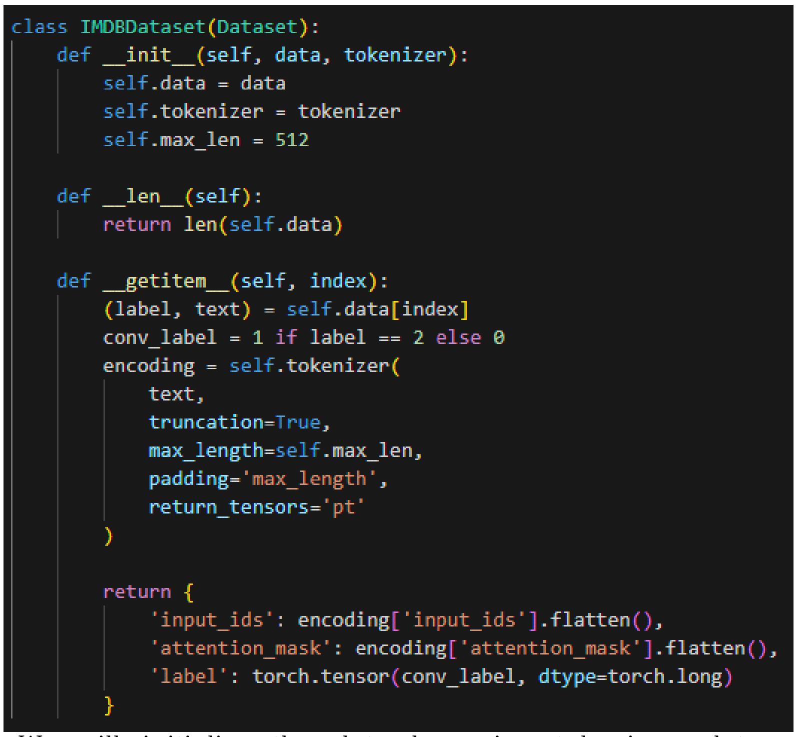
```
import random
import numpy as np
from tqdm import tqdm

import torch
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchtext.datasets import IMDB

from transformers import AdamW, get_linear_schedule_with_warmup
```

-We import needed libraries, the random function is for replicability(explained in detail on next code). We bring numpy, torch for data analysis. tqdm is for visualizing progress. Dataset and Dataloader is for bringing the datas. IMDB will be the training and validation data. AdamW is loss function and get_linear_schedule_with_warmup is for adjusting the learning rate dynamically.



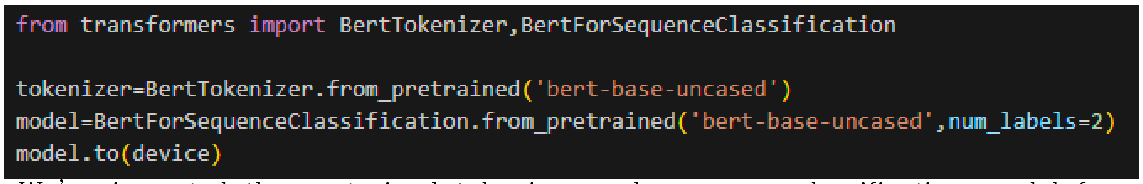
```
SEED=1234
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

-We will fix the random seed for reproducibility to produce same output for the same input. We will use GPU if available. Else, we will use CPU instead.



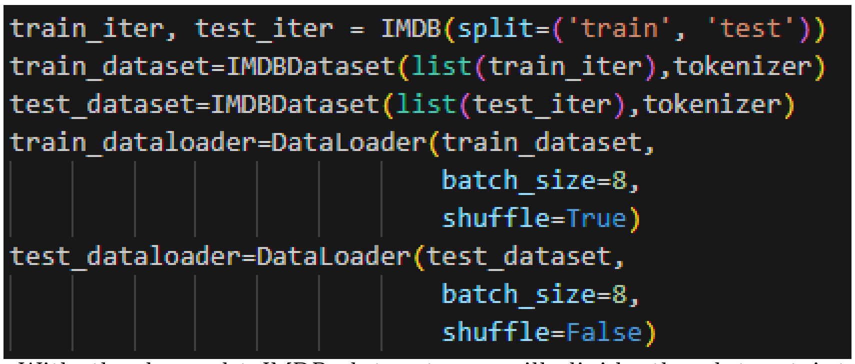
```python
class IMDBDataset(Dataset):
    def __init__(self, data, tokenizer):
        self.data = data
        self.tokenizer = tokenizer
        self.max_len = 512

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

    def __getitem__(self, index):
        (label, text) = self.data[index]
        conv_label = 1 if label == 2 else 0
        encoding = self.tokenizer(
            text,
            truncation=True,
            max_length=self.max_len,
            padding='max_length',
            return_tensors='pt'
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'label': torch.tensor(conv_label, dtype=torch.long)
        }
```

-We will initialize the data by using tokenizer, data, and setting max_len(max length to 512 due to positional embedding). For the __len__ function, it will return length. In the _getitem__ function, since IMDB dataset set 2 as positive and 1 as negative, we will change it into 0, 1 respectively for the label. By using the huggingface tokenizer, we will tokenize the datasets. We will return data into token id tensor, attention_mask, and label.



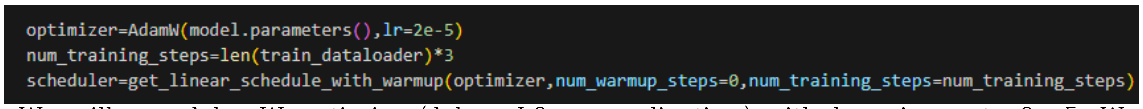
```python
from transformers import BertTokenizer,BertForSequenceClassification

tokenizer=BertTokenizer.from_pretrained('bert-base-uncased')
model=BertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=2)
model.to(device)
```

-We've imported the pretrained tokenizer and sequence classification model from the transformers.



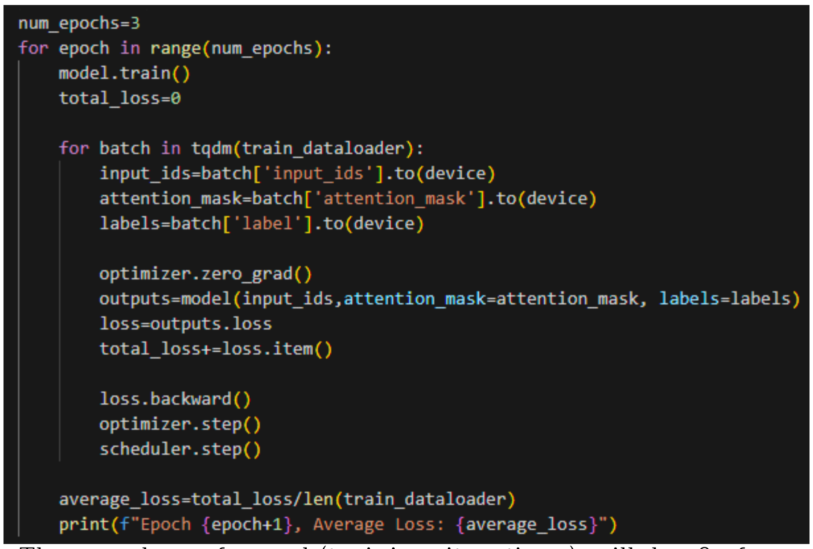
```python
train_iter, test_iter = IMDB(split=('train', 'test'))
train_dataset=IMDBDataset(list(train_iter),tokenizer)
test_dataset=IMDBDataset(list(test_iter),tokenizer)
train_dataloader=DataLoader(train_dataset,
                            batch_size=8,
                            shuffle=True)
test_dataloader=DataLoader(test_dataset,
                            batch_size=8,
                            shuffle=False)
```

-With the brought IMDB dataset, we will divide the dataset into train and test data. We will tokenize the datasets. After, we will shuffle the train data for the generalization and set the batch size into 8 to for the mini batch.



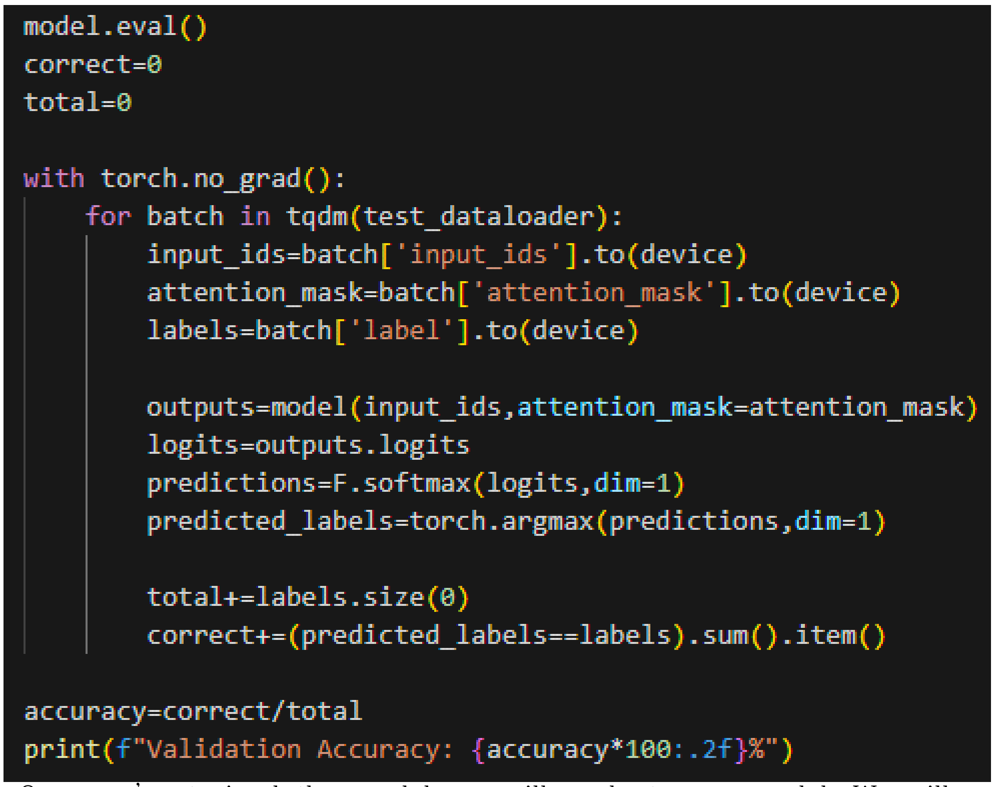
```python
optimizer=AdamW(model.parameters(),lr=2e-5)
num_training_steps=len(train_dataloader)*3
scheduler=get_linear_schedule_with_warmup(optimizer,num_warmup_steps=0,num_training_steps=num_training_steps)
```

-We will use AdamW optimizer(Adam+L2 generalization) with learning rate 2e-5. We will use learning rate scheduler with get_linear_schedule_with_warmup. This scheduler linearly decreases the learning. The total number of training_steps will be 3 because our training number of epoch will be 3.



```python
num_epochs=3
for epoch in range(num_epochs):
    model.train()
    total_loss=0

    for batch in tqdm(train_dataloader):
        input_ids=batch['input_ids'].to(device)
        attention_mask=batch['attention_mask'].to(device)
        labels=batch['label'].to(device)

        optimizer.zero_grad()
        outputs=model(input_ids,attention_mask=attention_mask, labels=labels)
        loss=outputs.loss
        total_loss+=loss.item()

        loss.backward()
        optimizer.step()
        scheduler.step()

    average_loss=total_loss/len(train_dataloader)
    print(f"Epoch {epoch+1}, Average Loss: {average_loss}")
```

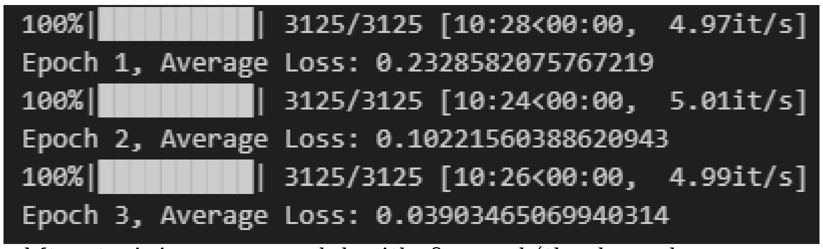
-The number of epoch(training iterations) will be 3. for each epoch, we will train our model with batch size(8). We will train the dataset with our setted tensor, attention_mask, and labels. We will print the status as we iterate the training.



```python
model.eval()
correct=0
total=0

with torch.no_grad():
    for batch in tqdm(test_dataloader):
        input_ids=batch['input_ids'].to(device)
        attention_mask=batch['attention_mask'].to(device)
        labels=batch['label'].to(device)

        outputs=model(input_ids,attention_mask=attention_mask)
        logits=outputs.logits
        predictions=F.softmax(logits,dim=1)
        predicted_labels=torch.argmax(predictions,dim=1)

        total+=labels.size(0)
        correct+=(predicted_labels==labels).sum().item()

accuracy=correct/total
print(f"Validation Accuracy: {accuracy*100:.2f}%")
```

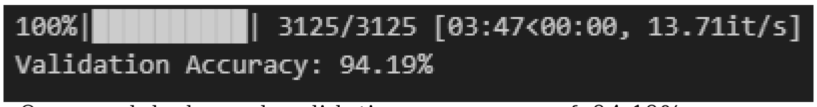
-Once we've trained the model, we will evaluate our model. We will calculate the prediction using softmax function and select the most high probability class as an prediction result. If the prediction and label is the same, we will increment the correct and calculate the accuracy based on number of successful prediction.

3. Result



```
100%|██████████| 3125/3125 [10:28<00:00,  4.97it/s]
Epoch 1, Average Loss: 0.2328582075767219
100%|██████████| 3125/3125 [10:24<00:00,  5.01it/s]
Epoch 2, Average Loss: 0.10221560388620943
100%|██████████| 3125/3125 [10:26<00:00,  4.99it/s]
Epoch 3, Average Loss: 0.03903465069940314
```

-After training our model with 3 epoch(the loss decreases as epoch iterates)



```
100%|██████████| 3125/3125 [03:47<00:00, 13.71it/s]
Validation Accuracy: 94.19%
```

-Our model showed validation accuracy of 94.19%