Outline

- Fine-tuning a BERT model in PyTorch
  - Loading the IMDb movie review dataset
  - Tokenizing the dataset
  - Loading and fine-tuning a pre-trained BERT model
  - Fine-tuning a transformer more conveniently using the Trainer API
- Summary

Quote from https://huggingface.co/transformers/custom_datasets.html:

> DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters than bert-base-uncased , runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark.

Fine-tuning a BERT model in PyTorch

Loading the IMDb movie review dataset

[1]:
```
import gzip
import shutil
import time

import pandas as pd
import requests
import torch
import torch.nn.functional as F
import torchtext

import transformers
from transformers import DistilBertTokenizerFast
from transformers import DistilBertForSequenceClassification
```

General Settings

[2]:
```
torch.backends.cudnn.deterministic = True
RANDOM_SEED = 123
torch.manual_seed(RANDOM_SEED)
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

NUM_EPOCHS = 3
```

Download Dataset

The following cells will download the IMDB movie review dataset (http://ai.stanford.edu/~amaas/data/sentiment/) for positive-negative sentiment classification in

as CSV-formatted file:

[3]:
```
url = "https://github.com/rasbt/machine-learning-book/raw/main/ch08/movie_data.
 ↪csv.gz"
filename = url.split("/")[-1]

with open(filename, "wb") as f:
    r = requests.get(url)
    f.write(r.content)

with gzip.open('movie_data.csv.gz', 'rb') as f_in:
    with open('movie_data.csv', 'wb') as f_out:
        shutil.copyfileobj(f_in, f_out)
```

Check that the dataset looks okay:

[4]:
```
df = pd.read_csv('movie_data.csv')
df.head()
```

[4]:

	review	sentiment
0	In 1974, the teenager Martha Moxley (Maggie Gr…	1
1	OK… so… I really like Kris Kristofferson a…	0
2	***SPOILER*** Do not read this, if you think a…	0
3	hi for all the people who have seen this wonde…	1
4	I recently bought the DVD, forgetting just how…	0

[5]:
```
df.shape
```

[5]:
```
(50000, 2)
```

Split Dataset into Train/Validation/Test

[6]:
```
train_texts = df.iloc[:35000]['review'].values
train_labels = df.iloc[:35000]['sentiment'].values

valid_texts = df.iloc[35000:40000]['review'].values
valid_labels = df.iloc[35000:40000]['sentiment'].values

test_texts = df.iloc[40000:]['review'].values
test_labels = df.iloc[40000:]['sentiment'].values
```

Tokenizing the dataset

[7]:
```
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
```

```
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
```

```
tokenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]
```

```
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

```
config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
```

[8]:
```
train_encodings = tokenizer(list(train_texts), truncation=True, padding=True)
valid_encodings = tokenizer(list(valid_texts), truncation=True, padding=True)
test_encodings = tokenizer(list(test_texts), truncation=True, padding=True)
```

[9]:
```
train_encodings[0]
```

[9]:
```
Encoding(num_tokens=512, attributes=[ids, type_ids, tokens, offsets,
attention_mask, special_tokens_mask, overflowing])
```

Dataset Class and Loaders

[10]:
```
class IMDbDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.
 ↪items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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


train_dataset = IMDbDataset(train_encodings, train_labels)
valid_dataset = IMDbDataset(valid_encodings, valid_labels)
test_dataset = IMDbDataset(test_encodings, test_labels)
```

[11]:
```
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=16,␣
 ↪shuffle=True)
valid_loader = torch.utils.data.DataLoader(valid_dataset, batch_size=16,␣
 ↪shuffle=False)
```

```
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=16,␣
 ↪shuffle=False)
```

Loading and fine-tuning a pre-trained BERT model

[12]:
```
model = DistilBertForSequenceClassification.
 ↪from_pretrained('distilbert-base-uncased')
model.to(DEVICE)
model.train()

optim = torch.optim.Adam(model.parameters(), lr=5e-5)
```

```
model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
```

```
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

Train Model – Manual Training Loop

[13]:
```
def compute_accuracy(model, data_loader, device):
    with torch.no_grad():
        correct_pred, num_examples = 0, 0

        for batch_idx, batch in enumerate(data_loader):

        ### Prepare data
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs['logits']
            predicted_labels = torch.argmax(logits, 1)
            num_examples += labels.size(0)
            correct_pred += (predicted_labels == labels).sum()

        return correct_pred.float()/num_examples * 100
```

[14]:
```
start_time = time.time()

for epoch in range(NUM_EPOCHS):

    model.train()

    for batch_idx, batch in enumerate(train_loader):
```

```
        ### Prepare data
        input_ids = batch['input_ids'].to(DEVICE)
        attention_mask = batch['attention_mask'].to(DEVICE)
        labels = batch['labels'].to(DEVICE)

        ### Forward
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss, logits = outputs['loss'], outputs['logits']

        ### Backward
        optim.zero_grad()
        loss.backward()
        optim.step()

        ### Logging
        if not batch_idx % 250:
            print (f'Epoch: {epoch+1:04d}/{NUM_EPOCHS:04d} | '
                   f'Batch {batch_idx:04d}/{len(train_loader):04d} | '
                   f'Loss: {loss:.4f}')

    model.eval()

    with torch.set_grad_enabled(False):
        print(f'Training accuracy: '
              f'{compute_accuracy(model, train_loader, DEVICE):.2f}%'
              f'\nValid accuracy: '
              f'{compute_accuracy(model, valid_loader, DEVICE):.2f}%')

    print(f'Time elapsed: {(time.time() - start_time)/60:.2f} min')

print(f'Total Training Time: {(time.time() - start_time)/60:.2f} min')
print(f'Test accuracy: {compute_accuracy(model, test_loader, DEVICE):.2f}%')
```

```
Epoch: 0001/0003 | Batch 0000/2188 | Loss: 0.6894
Epoch: 0001/0003 | Batch 0250/2188 | Loss: 0.1266
Epoch: 0001/0003 | Batch 0500/2188 | Loss: 0.3208
Epoch: 0001/0003 | Batch 0750/2188 | Loss: 0.0238
Epoch: 0001/0003 | Batch 1000/2188 | Loss: 0.0983
Epoch: 0001/0003 | Batch 1250/2188 | Loss: 0.0143
Epoch: 0001/0003 | Batch 1500/2188 | Loss: 0.1931
Epoch: 0001/0003 | Batch 1750/2188 | Loss: 0.0866
Epoch: 0001/0003 | Batch 2000/2188 | Loss: 0.2064
Training accuracy: 96.18%
Valid accuracy: 92.02%
Time elapsed: 37.72 min
Epoch: 0002/0003 | Batch 0000/2188 | Loss: 0.0897
Epoch: 0002/0003 | Batch 0250/2188 | Loss: 0.0510
```

```
Epoch: 0002/0003 | Batch 0500/2188 | Loss: 0.0180
Epoch: 0002/0003 | Batch 0750/2188 | Loss: 0.0066
Epoch: 0002/0003 | Batch 1000/2188 | Loss: 0.0996
Epoch: 0002/0003 | Batch 1250/2188 | Loss: 0.2691
Epoch: 0002/0003 | Batch 1500/2188 | Loss: 0.0728
Epoch: 0002/0003 | Batch 1750/2188 | Loss: 0.1633
Epoch: 0002/0003 | Batch 2000/2188 | Loss: 0.2918
Training accuracy: 98.83%
Valid accuracy: 93.00%
Time elapsed: 75.54 min
Epoch: 0003/0003 | Batch 0000/2188 | Loss: 0.1123
Epoch: 0003/0003 | Batch 0250/2188 | Loss: 0.0622
Epoch: 0003/0003 | Batch 0500/2188 | Loss: 0.0071
Epoch: 0003/0003 | Batch 0750/2188 | Loss: 0.0141
Epoch: 0003/0003 | Batch 1000/2188 | Loss: 0.1116
Epoch: 0003/0003 | Batch 1250/2188 | Loss: 0.0994
Epoch: 0003/0003 | Batch 1500/2188 | Loss: 0.0238
Epoch: 0003/0003 | Batch 1750/2188 | Loss: 0.0119
Epoch: 0003/0003 | Batch 2000/2188 | Loss: 0.0690
Training accuracy: 99.14%
Valid accuracy: 92.24%
Time elapsed: 113.34 min
Total Training Time: 113.34 min
Test accuracy: 92.19%
```

[15]:
```
del model # free memory
```