Lab - 10 Fine-tuning BERT with PyTorch for Telecom Text Analysis

Objective

This laboratory session focuses on fine-tuning BERT (Bidirectional Encoder Representations from Transformers) using PyTorch for telecom-specific text analysis tasks. We'll work with a customer service conversations dataset to develop practical skills in applying transformer-based models to real-world telecom problems.

Dataset Description

We'll be working with an Email Spam Classification dataset containing:

`Text` - Email subject lines and body text

`Spam` - Binary classification labels (spam/ham)

Tasks

Task 1: Data Preparation

- Review data distribution
- Analyze text lengths
- Split data into train and test split

Task 2: Create data loaders

- Configure BERT tokenizer
- Build custom Dataset class
- Implement batch processing
- Set up train and test data loaders

Task 3: BERT model Configuration

- Initialize pre-trained BERT
- Configure classification head
- Set up optimizer and loss function

Task 4: Training Implementation

- Create training loop

Task 5: Model Evaluation

- Prediction on test dataset
- Calculate accuracy, precision, recall

Double-click (or enter) to edit

```
# ! pip install torch --quiet
# ! pip install transformers --quiet
# ! pip install pandas --quiet
# ! pip install scikit-learn --quiet
# ! pip install pyarrow --quiet
# ! pip install fastparquet --quiet
```

Importing libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
# from transformers import BertForSequenceClassification
from transformers import AutoTokenizer
import torch
import torch.nn as nn
```

```
from torch.utils.data import Dataset, DataLoader
from torch.optim import AdamW
import numpy as np
from transformers import BertTokenizer, BertModel, AdamW
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Loading dataset

```
data = pd.read_parquet(r'D:\Nokia_DL_L3_lab\NLP\email_spam_train.parquet')
data.head()
```

	Text	Spam
0	Subject: aiesec polska - eurolds 2000 jarek ,...	0
1	Subject: vince and stinson , i got this resum...	0
2	Subject: re : many helyette , sorry for not ...	0
3	Subject: re : replacement of stolen chairs fy...	0
4	Subject: fw : mark boland - cv vince : tony v...	0

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4556 entries, 0 to 4555
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Text    4556 non-null   object
 1   Spam    4556 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 71.3+ KB
```

Splitting it into train and test dataset

```
# train_X, test_X, train_Y, test_Y = train_test_split(data['Text'], data['Spam'], train_size = 0.7, random_state = 42)
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
```

bert model

```
model_checkpoint = "bert-base-uncased"
```

Making Pytorch dataset

```
class TokenData(Dataset):
    def __init__(self, data, tokenizer):
        self.data = data
        self.tokenizer = tokenizer
        self.max_length = 256

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

    def __getitem__(self, index):
        text = self.data.iloc[index]['Text']
        labels = self.data.iloc[index][['Spam']].values.astype(int)
        encoding = self.tokenizer(text, return_tensors='pt', padding=True, truncation=True, max_length=self.max_length)
        input_ids = encoding['input_ids'][0]
        attention_mask = encoding['attention_mask'][0]
        # resize the tensors to the same size
        input_ids = nn.functional.pad(input_ids, (0, self.max_length - input_ids.shape[0]), value=0)
        attention_mask = nn.functional.pad(attention_mask, (0, self.max_length - attention_mask.shape[0]), value=0)
        return input_ids, attention_mask, torch.tensor(labels)


batch_size = 64
tokenizer = BertTokenizer.from_pretrained(model_checkpoint)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
train_dataset = TokenData(train_data, tokenizer)
test_dataset = TokenData(test_data, tokenizer)
```

Train and test loader

```
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

```
# num_epochs = 3
device = "cuda" if torch.cuda.is_available() else "mps" if torch.backends.mps.is_available() else "cpu"
device
```

```
'cuda'
```

Creating BERT model

```
class BertClassifier(nn.Module):
    def __init__(self, num_labels):
        super(BertClassifier, self).__init__()
        self.bert = BertModel.from_pretrained(model_checkpoint)
        self.classifier = nn.Sequential(
            nn.Linear(self.bert.config.hidden_size, 300),
            nn.ReLU(),
            nn.Linear(300, 100),
            nn.ReLU(),
            nn.Linear(100, 50),
            nn.ReLU(),
            nn.Linear(50, num_labels)

        )

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        x = outputs['last_hidden_state'][:, 0, :]
        x = self.classifier(x)
        return x
```

Setting up some parameters

```
num_labels = 2
model = BertClassifier(num_labels).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 2e-5)
```

```
num_epochs = 3
n_total_steps = len(train_loader)
```

Model architecture

```
print(model)
```

```
BertClassifier(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
```

```
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (pooler): BertPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
)
(classifier): Sequential(
  (0): Linear(in_features=768, out_features=300, bias=True)
  (1): ReLU()
  (2): Linear(in_features=300, out_features=100, bias=True)
  (3): ReLU()
  (4): Linear(in_features=100, out_features=50, bias=True)
  (5): ReLU()
  (6): Linear(in_features=50, out_features=2, bias=True)
)
)
```

```
# len(train_loader)
```

```
57
```

Training loop

```
for epoch in range(num_epochs):

  for i, batch in enumerate(train_loader):

    input_ids, attention_mask, labels = batch
    input_ids = input_ids.to(device)

    attention_mask = attention_mask.to(device)
    labels = labels.type(torch.LongTensor)
    labels = labels.view(-1)
    labels = labels.to(device)

    optimizer.zero_grad()

    logits = model(input_ids, attention_mask)

    loss = criterion(logits, labels)
    loss.backward()
    optimizer.step()


    if (i+1) % 10 == 0:
        print(f'epoch {epoch + 1}/ {num_epochs}, batch {i+1}/{n_total_steps}, loss = {loss.item():.4f}')
```

```
epoch 1/ 3, batch 10/57, loss = 0.5535
epoch 1/ 3, batch 20/57, loss = 0.4974
epoch 1/ 3, batch 30/57, loss = 0.4135
epoch 1/ 3, batch 40/57, loss = 0.3820
epoch 1/ 3, batch 50/57, loss = 0.3428
epoch 2/ 3, batch 10/57, loss = 0.2331
epoch 2/ 3, batch 20/57, loss = 0.1798
epoch 2/ 3, batch 30/57, loss = 0.1532
epoch 2/ 3, batch 40/57, loss = 0.1364
epoch 2/ 3, batch 50/57, loss = 0.1209
epoch 3/ 3, batch 10/57, loss = 0.1023
epoch 3/ 3, batch 20/57, loss = 0.0706
epoch 3/ 3, batch 30/57, loss = 0.1050
epoch 3/ 3, batch 40/57, loss = 0.0517
epoch 3/ 3, batch 50/57, loss = 0.0438
```

Testing on test loader

```
all_labels = []
all_preds = []

with torch.no_grad():
  n_correct = 0
  n_samples = 0
  for i, batch in enumerate (test_loader):

    input_ids, attention_mask, labels = batch
    input_ids = input_ids.to(device)

    attention_mask = attention_mask.to(device)

    labels = labels.view(-1)
    labels = labels.to(device)

    outputs = model(input_ids, attention_mask)

    _, predictions = torch.max(outputs, 1)

    all_labels.append(labels.cpu().numpy())
    all_preds.append(predictions.cpu().numpy())
```

```
all_labels = np.concatenate(all_labels, axis=0)
all_preds = np.concatenate(all_preds, axis=0)
```

Classification report

```
print(classification_report(all_labels, all_preds))
print(accuracy_score(all_labels, all_preds))
```

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.98       676
           1       0.88      1.00      0.94       236

    accuracy                           0.97       912
   macro avg       0.94      0.98      0.96       912
weighted avg       0.97      0.97      0.97       912

0.9660087719298246
```

Start coding or generate with AI.