基于IMDB的情感分类

数据加载与预处理

使用Keras 的内置 IMDB 数据集，将评论转换为固定长度的词索引序列，并构造 PyTorch 的 Dataset 和 DataLoader 以便后续训练。

In [1]:
```python
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
import torch
from torch.utils.data import Dataset, DataLoader
```

设置序列填充参数

In [2]:
```python
vocab_size = 10000      # 词汇表大小（考虑常见的10000个词）
maxlen = 200            # 文本序列最大长度
```

加载数据集

In [3]:
```python
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=
```

将所有序列填充到固定长度，便于批量处理

```
In [4]: train_data = pad_sequences(train_data, maxlen=maxlen, padding='post') # 填充训练
        test_data = pad_sequences(test_data, maxlen=maxlen, padding='post') # 填充测试集
```

构建pytorch数据集

```
In [5]: class MovieDataset(Dataset):
            def __init__(self, data, labels):
                self.data = data
                self.labels = labels

            def __len__(self):
                return len(self.labels) # 获取数据集大小

            def __getitem__(self, idx):
                # 获取单条样本的序列张量和标签张量
                x = torch.tensor(self.data[idx], dtype=torch.long) # 使用 long 类型以配合
                y = torch.tensor(self.labels[idx], dtype=torch.float)  # 使用 float 类型
                return x, y
```

实例化数据集以及设置批量处理加载器

```
In [6]: train_dataset = MovieDataset(train_data, train_labels) # 将训练集数据封装为MovieD
        test_dataset  = MovieDataset(test_data, test_labels) # 将测试集数据封装为MovieDat
        batch_size = 64 # 批大小
```

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

print(f"Number of training samples: {len(train_dataset)}, testing samples: {len(
```

```
Number of training samples: 25000, testing samples: 25000
```

BERT模型

使用BERT模型进行情感分类。

In [7]:
```
import torch.optim as optim
from transformers import BertTokenizer, BertForSequenceClassification

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') # 设备选择
```

将数字序列还原为英文单词序列

In [8]:
```
word_index = imdb.get_word_index() # 获取词汇表映射
reverse_word_index = {value+3: key for key, value in word_index.items()} # keras
reverse_word_index[0] = "<PAD>" # keras默认预留了0作为填充符
reverse_word_index[1] = "<START>" # keras默认预留了1作为起始符
reverse_word_index[2] = "<UNK>" # keras默认预留了2作为未知符
```

将序列还原为英文句子

In [11]:

```
texts_train = [] # 用于存储训练集文本
for seq in train_data:
    words = [reverse_word_index.get(idx, "<PAD>") for idx in seq] # 将索引转换为
    words = [w for w in words if w not in ["<PAD>", "<START>", "<UNK>"]] # 去除
    texts_train.append(" ".join(words)) # 将单词列表拼接为文本字符串

texts_test = [] # 用于存储测试集文本
for seq in test_data:
    words = [reverse_word_index.get(idx, "<PAD>") for idx in seq] # 将索引转换为
    words = [w for w in words if w not in ["<PAD>", "<START>", "<UNK>"]] # 去除
    texts_test.append(" ".join(words)) # 将单词列表拼接为文本字符串
```

使用BERT分词器对文本数据进行编码

In [12]:

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased') # 加载一个预训练好
max_len = 128 # 设置模型最大输入长度为128
train_encodings = tokenizer(texts_train, truncation=True, padding=True, max_leng
test_encodings  = tokenizer(texts_test,  truncation=True, padding=True, max_leng

train_labels_list = train_labels.tolist() # 将标签数组转换为列表
test_labels_list  = test_labels.tolist()
```

将bert编码后的数据转换为pytorch可处理的数据格式

In [13]:

```
class IMDBDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings # bert编码后的数据字典
        self.labels = labels # 标签列表
    def __len__(self):
        return len(self.labels) # 数据集长度
    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items
        item['labels'] = torch.tensor(self.labels[idx], dtype=torch.long) # 设置
        return item

train_dataset_bert = IMDBDataset(train_encodings, train_labels_list) # 创建bert编
test_dataset_bert  = IMDBDataset(test_encodings,  test_labels_list) # 创建bert编
train_loader_bert  = DataLoader(train_dataset_bert, batch_size=16, shuffle=True)
test_loader_bert   = DataLoader(test_dataset_bert,  batch_size=16, shuffle=False
```

准备bert模型和优化器

In [14]:
```
model_bert = BertForSequenceClassification.from_pretrained('bert-base-uncased',
optimizer = optim.Adam(model_bert.parameters(), lr=2e-5) # 定义优化器
```

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

训练bert模型

In [16]:
```
epochs = 2 # 训练轮次2轮
for epoch in range(epochs):
    model_bert.train() # 设置模型为训练模式
    total_loss = 0.0 # 初始化总损失
    for batch in train_loader_bert: # 对于每个批次的训练
        optimizer.zero_grad() # 梯度清零
        input_ids = batch['input_ids'].to(device) # 将bert编码后的ID数据移到设备
        attention_mask = batch['attention_mask'].to(device) # 将bert编码后的有效
        labels = batch['labels'].to(device) # 将bert编码后的标签数据移到设备
        outputs = model_bert(input_ids=input_ids, attention_mask=attention_mask,
        loss = outputs.loss # 计算损失
        loss.backward() # 反向传播计算梯度
        optimizer.step() # 更新参数
        total_loss += loss.item() # 累加损失
    avg_loss = total_loss / len(train_loader_bert) # 平均损失
    print(f"Epoch {epoch+1}/{epochs}, Loss: {avg_loss:.4f}") # 打印对应轮次的损失
```

```
Epoch 1/2, Loss: 0.0919
Epoch 2/2, Loss: 0.0506
```

在测试集上对模型进行评估

In [17]:
```
model_bert.eval() # 将模型切换为评估模式
correct = 0 # 初始化预测正确的样本数
total = 0 # 初始化总样本数
with torch.no_grad(): # 关闭梯度计算
    for batch in test_loader_bert: # 对于测试集的每个批次
        input_ids = batch['input_ids'].to(device)# 将当前批次的ID数据移到设备
        attention_mask = batch['attention_mask'].to(device)# 将当前批次的有效性参
        labels = batch['labels'].to(device) # 将当前批次的标签数据移到设备
        outputs = model_bert(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits # 提取样本对不同类别的预测分数
        preds = torch.argmax(logits, dim=1) # 找出最大值的索引
        correct += (preds == labels).sum().item() # 将预测结果与真是标签比较
        total += labels.size(0) # 样本总数
accuracy = correct / total # 计算准确率
print(f"BERT模型测试准确率： {accuracy*100:.2f}%") # 打印准确率
```

```
BERT模型测试准确率： 89.24%
```