山东大学计算机科学与技术学院

可视化技术课程实验报告

学号：202300130220	姓名：刘傲宇	班级：数据科学与大数据技术班
实验题目：BERT 实践		
实验学时：2	实验日期：2025/	

实验目标：

对动手实践利用机器学习方法分析大规模数据有进一步了解，并学习如何利用远程环境进行工程代码的调试

实验步骤与内容：

自定义 MRPCDataset 类，对数据集进行加载和预处理：

在数据加载时，首先需要处理这些文本数据，使用 BERT 的 Tokenizer 将句子转化为 token IDs，并构造 input_ids、attention_mask 等输入格式。

```
class MRPCDataset(Dataset):
    def __init__(self, file_path, tokenizer=None, max_length=128):
        """
        初始化数据集，加载数据和准备预处理。
        :param file_path: 数据集文件路径
        :param tokenizer: BERT tokenizer
        :param max_length: 最大输入序列长度
        """
        local_model_path = "./bert-base-uncased"
        self.tokenizer = tokenizer if tokenizer else BertTokenizer.from_pretrained(local_model_path)
        self.max_length = max_length

        # 加载数据
        self.data = []
        with open(file_path, 'r', encoding='utf-8-sig') as f:  # 使用 'utf-8-sig' 以跳过 BOM
            next(f)  # 跳过表头
            for line in f:
                parts = line.strip().split('\t')  # 使用制表符（tab）分隔字段
                if len(parts) < 5:  # 防止格式错误的行
                    continue
                label = int(parts[0])  # 标签为第一列
                sentence1 = parts[3]  # 第一个句子为第四列
                sentence2 = parts[4]  # 第二个句子为第五列
                self.data.append((sentence1, sentence2, label))

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

```python
def __getitem__(self, idx):
    # 获取每个样本
    sentence1, sentence2, label = self.data[idx]

    # 确保 sentence1 和 sentence2 都是字符串
    assert isinstance(sentence1, str), f"Expected string but got {type(sentence1)}"
    assert isinstance(sentence2, str), f"Expected string but got {type(sentence2)}"

    # 对句子进行编码
    encoding = self.tokenizer(
        sentence1, sentence2,
        truncation=True,
        padding='max_length',
        max_length=self.max_length,
        return_tensors='pt'
    )

    # 获取输入数据并转换为tensor
    input_ids = encoding['input_ids'].squeeze(0)  # 去掉batch维度
    attention_mask = encoding['attention_mask'].squeeze(0)

    return {
        'input_ids': input_ids,
        'attention_mask': attention_mask,
        'label': torch.tensor(label, dtype=torch.float)
    }
```

模型构建：

采用了 HuggingFace 的 transformers 库，加载预训练模型 BERT-base-uncased。该模型经过预训练，可直接用于下游任务的微调。首先，利用 BERT 模型获取句子对的特征表示，具体使用的是其池化层的输出（即 [CLS] token 的特征）。然后，将该特征输入到一个全连接层（FCModel）中，实现二分类任务的预测。

root > FCModel.py

```python
import torch
import torch.nn as nn


class FCModel(nn.Module):
    def __init__(self):
        """
        初始化全连接层模型
        """
        super(FCModel, self).__init__()

        # 定义一个包含两个全连接层的简单网络
        self.fc1 = nn.Linear(768, 256)  # BERT输出的维度为768
        self.fc2 = nn.Linear(256, 1)  # 输出一个标量，表示二分类
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        """
        前向传播函数
        :param x: BERT模型的池化输出，形状为(batch_size, 768)
        :return: 预测的概率值
        """
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.sigmoid(x)  # 输出一个0到1之间的概率值
        return x
```

训练：

```python
def train():
    epoch_loss, epoch_acc = 0., 0.
    total_len = 0

    for i, data in enumerate(train_loader):
        print("当前显存使用情况: ", torch.cuda.memory_allocated())

        bert_model.train()
        model.train()

        input_ids = data['input_ids'].to(device)  # 直接使用 tokenized 的 input_ids
        attention_mask = data['attention_mask'].to(device)  # 输入句子的attention_mask
        label = data['label'].to(device)  # 标签

        # 检查 label 是否在 [0, 1] 范围内
        #print(f"标签值的范围: min={label.min().item()}, max={label.max().item()}")

        # 确保标签在 [0, 1] 之间
        label = torch.clamp(label, 0, 1)

        encoding = {
            'input_ids': input_ids,
            'attention_mask': attention_mask
        }

        # 传递给 BERT 模型进行前向传播
        bert_output = bert_model(**encoding)
        pooler_output = bert_output.pooler_output  # 获取BERT模型池化后的输出

        # 通过全连接层模型进行预测
        predict = model(pooler_output).squeeze()  # [batch_size, 1] -> [batch_size]

        # 计算损失和准确率
        loss = crit(predict, label.float())
        acc = binary_accuracy(predict, label)

        # 梯度清零、反向传播、优化
        optimizer.zero_grad()
        bert_optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        bert_optimizer.step()

        epoch_loss += loss.item() * len(label)
        epoch_acc += acc.item() * len(label)
        total_len += len(label)

        # 打印每个batch的loss和准确率
```

结果：

……

结论分析与体会：

本次实验完成了 MRPC 数据集同义预测全流程，自定义数据集类保障了数据质量，BERT 与全连接层配合实现了有效预测，训练显存稳定，最终模型准确率达 67.54%。深刻感受到数据预处理是基础，“预训练模型 + 自定义分类头” 架构适配性强，训练中实时监控状态、把控细节很重要，同时也认识到模型仍有优化空间，后续可通过调参、扩充数据等方式进一步提升性能，积累了端到端机器学习任务的实践经验。