# 山东大学计算机科学与技术学院

# 大数据分析实践课程实验报告

| 学号：202300300198 | 姓名：朱宣玮 | 班级：数据 23 |
|---|---|---|
| 实验题目：BERT 实践 | | |
| 实验学时：2 | 实验日期：2025.11.7 | |

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.

实验环境：

实验 4 配置的 BERT 服务器环境

操作系统

Linux (AutoDL 容器环境)

Python 版本：3.8

深度学习框架

PyTorch 2.4.1+cu121 (CUDA 12.1)

Transformers 4.46.3

数据处理库

Datasets 3.1.0

实验步骤与结果：

实验步骤

1. 配置环境与参数

在 VS CODE 中通过 Remote - SSH 插件连接实验服务器。

导入需要的库，定义训练过程中所需的超参数。

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from transformers import BertTokenizer, BertModel
from datasets import load_dataset
import os

# 设置网络代理，以直接从Hugging Face上下载数据集
os.environ["http_proxy"] = "http://127.0.0.1:7890"
os.environ["https_proxy"] = "http://127.0.0.1:7890"

BATCH_SIZE = 8
LEARNING_RATE = 1e-5
NUM_EPOCHS = 3
MAX_LENGTH = 128  # 句子最大长度
```

2. 定义模型与工具函数

创建用于分类的全连接层模型 FCModel，以及用于计算准确率的 binary_accuracy 函数、处理文本数据的 preprocess_function 和在验证集上评估模型的 evaluate 函数。

```python
# 全连接层模型
class FCModel(nn.Module):
    def __init__(self, input_dim=768, hidden_dim=256, output_dim=1):
        super(FCModel, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return x

# 工具函数
def binary_accuracy(predictions, labels):
    """计算二分类准确率"""
    rounded_preds = torch.round(predictions)
    correct = (rounded_preds == labels).float()
    accuracy = correct.sum() / len(correct)
    return accuracy
def preprocess_function(examples, tokenizer):
    """预处理函数：将文本转换为模型输入"""
    return tokenizer(
        examples["sentence1"],
        examples["sentence2"],
        truncation=True,
        padding="max_length",
        max_length=MAX_LENGTH
    )

def evaluate(model, bert_model, eval_loader, criterion, device):
    """在验证集上评估模型性能"""
    model.eval()  # 切换到评估模式
    bert_model.eval()

    total_loss = 0.0
    total_acc = 0.0
    total_samples = 0
```

3. 加载数据和模型

使用 datasets 库加载 MRPC 数据集，自动分割为训练集和验证集。随后加载预训练的 BERT 模型和分词器，并将模型移动到可用的计算设备（在服务器上使用 GPU）。

```python
# 主训练流程
def main():
    # 1. 自动载入MRPC数据集
    print("正在载入MRPC数据集...")
    dataset = load_dataset("glue", "mrpc")
    train_dataset = dataset["train"]
    eval_dataset = dataset["validation"] # 加载验证集
    print(f"数据集载入完成，训练集：{len(train_dataset)} 个样本，验证集：{len(eval_dataset)} 个样本")

    # 2. 设置运行设备
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f"使用设备：{device}")

    # 3. 加载BERT模型和Tokenizer
    print("正在加载BERT模型...")
    tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
    bert_model = BertModel.from_pretrained("bert-base-uncased")
    bert_model.to(device)
    print("BERT模型加载完成")

    # 4. 创建全连接层模型
    print("正在创建全连接层模型...")
    model = FCModel()
    model.to(device)
    print("全连接层模型创建完成")
```

4. 配置训练组件

定义优化器（Adam）、损失函数（二元交叉熵损失 BCELoss）。对训练集和验证集进行预处理，创建 DataLoader 以支持批量数据加载。

```
# 5. 定义优化器和损失函数
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
bert_optimizer = torch.optim.Adam(bert_model.parameters(), lr=LEARNING_RATE)
criterion = nn.BCELoss()  # 二分类交叉熵损失

# 6. 预处理数据集
print("正在预处理训练集...")
tokenized_train_dataset = train_dataset.map(
    lambda examples: preprocess_function(examples, tokenizer),
    batched=True
)
tokenized_train_dataset.set_format(
    type="torch",
    columns=["input_ids", "attention_mask", "token_type_ids", "label"]
)

print("正在预处理验证集...")
tokenized_eval_dataset = eval_dataset.map( # 预处理验证集
    lambda examples: preprocess_function(examples, tokenizer),
    batched=True
)
tokenized_eval_dataset.set_format(
    type="torch",
    columns=["input_ids", "attention_mask", "token_type_ids", "label"]
)
print("数据集预处理完成")

# 7. 创建DataLoader
train_loader = DataLoader(
    tokenized_train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True
)
eval_loader = DataLoader( # 创建验证集DataLoader
    tokenized_eval_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False # 验证时不需要打乱
)
```

5. 执行训练与验证

循环执行训练过程。在每个 Epoch 中，首先在训练集上进行模型训练，计算并打印每批数据的损失和准确率。训练结束后，立即在验证集上评估模型当前的泛化能力，并输出验证结果。

```
for epoch in range(NUM_EPOCHS):
    print(f"\n===== Epoch {epoch + 1}/{NUM_EPOCHS} =====")

    # --- 训练阶段 ---
    model.train()
    bert_model.train()
    epoch_loss = 0.0
    epoch_acc = 0.0
    total_samples = 0

    for batch_idx, batch in enumerate(train_loader):
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        token_type_ids = batch["token_type_ids"].to(device)
        labels = batch["label"].float().to(device)

        optimizer.zero_grad()
        bert_optimizer.zero_grad()

        bert_output = bert_model(
            input_ids=input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids
        )
        pooler_output = bert_output.pooler_output
        predictions = model(pooler_output).squeeze()

        loss = criterion(predictions, labels)
        acc = binary_accuracy(predictions, labels)

        loss.backward()
        optimizer.step()
        bert_optimizer.step()

        batch_size = labels.size(0)
        epoch_loss += loss.item() * batch_size
        epoch_acc += acc.item() * batch_size
        total_samples += batch_size

        if (batch_idx + 1) % 10 == 0:
            print(f"Batch {batch_idx + 1}/{len(train_loader)} - Loss: {loss.item():.4f}, Acc: {acc.item():.4f}")

    avg_train_loss = epoch_loss / total_samples
    avg_train_acc = epoch_acc / total_samples
    print(f"训练集 - 平均损失: {avg_train_loss:.4f}, 平均准确率: {avg_train_acc:.4f}")
```

训练过程：

```
正在载入MRPC数据集...
数据集载入完成，训练集：3668 个样本，验证集：408 个样本
使用设备：cuda
正在加载BERT模型...
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertModel: ['cls.predictions.transform.LayerNorm.bi
as', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.dense.bias', 'cls.predictions.bias', 'cls.predictions.transform
.dense.weight', 'cls.seq_relationship.bias', 'cls.predictions.decoder.weight', 'cls.seq_relationship.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture (e.g
. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical (initializing
 a BertForSequenceClassification model from a BertForSequenceClassification model).
BERT模型加载完成
正在创建全连接层模型...
全连接层模型创建完成
正在预处理训练集...
正在预处理验证集...
数据集预处理完成
开始训练...

===== Epoch 1/3 =====
Batch 10/459 - Loss: 0.6471, Acc: 0.7500
Batch 20/459 - Loss: 0.6702, Acc: 0.6250
Batch 30/459 - Loss: 0.6307, Acc: 0.7500
Batch 40/459 - Loss: 0.6393, Acc: 0.7500
Batch 50/459 - Loss: 0.7845, Acc: 0.3750
Batch 60/459 - Loss: 0.5426, Acc: 0.8750
Batch 70/459 - Loss: 0.6685, Acc: 0.6250
Batch 80/459 - Loss: 0.6139, Acc: 0.6250
Batch 90/459 - Loss: 0.5774, Acc: 0.7500
Batch 100/459 - Loss: 0.5376, Acc: 0.7500
Batch 110/459 - Loss: 0.4992, Acc: 0.8750


Batch 300/459 - Loss: 0.3628, Acc: 0.8750
Batch 310/459 - Loss: 0.7217, Acc: 0.7500
Batch 320/459 - Loss: 0.0638, Acc: 1.0000
Batch 330/459 - Loss: 0.0609, Acc: 1.0000
Batch 340/459 - Loss: 0.0514, Acc: 1.0000
Batch 350/459 - Loss: 0.0556, Acc: 1.0000
Batch 360/459 - Loss: 0.2865, Acc: 0.8750
Batch 370/459 - Loss: 0.0719, Acc: 1.0000
Batch 380/459 - Loss: 0.0598, Acc: 1.0000
Batch 390/459 - Loss: 0.0666, Acc: 1.0000
Batch 400/459 - Loss: 0.0886, Acc: 1.0000
Batch 410/459 - Loss: 0.0723, Acc: 1.0000
Batch 420/459 - Loss: 0.1163, Acc: 1.0000
Batch 430/459 - Loss: 0.9571, Acc: 0.7500
Batch 440/459 - Loss: 0.3091, Acc: 0.8750
Batch 450/459 - Loss: 0.0969, Acc: 1.0000
训练集 - 平均损失：0.2115，平均准确率：0.9286
正在进行验证...
验证集 - 平均损失：0.3591，平均准确率：0.8578
```

每轮的训练结果如下

Epoch 1：

训练集 - 平均损失: 0.5736, 平均准确率: 0.7069

验证集 - 平均损失: 0.4306, 平均准确率: 0.8088

Epoch 2：

训练集 - 平均损失: 0.3693, 平均准确率: 0.8484

验证集 - 平均损失: 0.3455, 平均准确率: 0.8505

Epoch 3：

训练集 - 平均损失: 0.2115, 平均准确率: 0.9286

验证集 - 平均损失: 0.3591, 平均准确率: 0.8578

结论分析与体会：

结论分析

模型训练效果显著，训练集准确率从 70.69%升至 92.86%，验证集达 85.78%，证明 BERT 微调能有效捕捉语义匹配特征。

第 3 轮验证集损失略升，因数据集小、模型复杂，出现轻微过拟合，整体性能符合任务预期。

体会

通过本次实验，实现了利用机器学习方法（BERT 微调）分析 MRPC 文本数据的目标，深入理解了数据预处理、模型训练与评估的全流程。同时，掌握了通过 VS CODE 连接服务器，实现在远程环境下调试代码。