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：202300130262	姓名： 何青青	班级： 23 数据
实验题目：bert 环境配置+bert 实践		
实验学时：4	实验日期：2025/11/7-11/21	

实验目标：

1、深入理解 PyTorch 框架下利用预训练 BERT 模型对 MRPC 数据集进行同义预测的完整 pipeline，包括数据预处理逻辑、模型加载与微调机制、推理过程及性能评价方法。

2、动手实践机器学习方法分析大规模文本数据，通过训练过程中损失值与准确率的变化，验证模型优化效果，提升对 Transformer 类模型（BERT）在自然语言处理（文本语义匹配）任务中应用的认知。

实验环境：

硬件环境：带 GPU 主机，CUDA 版本 12.7

软件环境：VSCode、Anaconda 、 Python 3.8、 PyTorch 1.7.0、Transformers 4.18.0

实验步骤与内容：

1、配置环境

①检查 Anaconda 是否安装:如下图所示,使用 conda 命令查看版本信息时可以看到 Anaconda 已经成功安装

```
C:\Users\13223>conda --version
conda 24.11.3
```

②创建虚拟环境：为隔离实验依赖，满足本次实验所需要的环境，考虑创建虚拟环境

```
C:\Users\13223>conda create -n ml-lab python=3.8
Retrieving notices: done
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: E:\Anaconda\envs\ml-lab

  added / updated specs:
    - python=3.8
```

可以成功激活该虚拟环境

```
(base) PS C:\Users\13223> conda activate ml-lab
```

③安装 PyTorch

a)首先查询本地 GPU 版本及 CUDA 版本，从而确定需要安装的 PyTorch 版本，如下图，CUDA 版本为 12.7，符合本次实验的“CUDA 版本大于 10.0”要求（由于始终无法找到可用且可连接的远程 GPU 服务器，且本地 GPU 符合实验要求，故使用本地 GPU 完成本次实验）

```
C:\Users\13223>nvidia-smi
Thu Nov 20 23:46:16 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 566.07                 Driver Version: 566.07         CUDA Version: 12.7     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                  Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf           Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 3060 ...  WDDM  |   00000000:01:00.0 Off |                  N/A |
| N/A   47C    P0             24W /   80W |       0MiB /   6144MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

b)接下来在虚拟环境中完成 PyTorch 的下载：

```
(ml-lab) PS D:\大三\大数据分析实践\实验\bert\BDA_bert> pip install torch==2.3.1 torch
vision==0.18.1 torchaudio==2.3.1 --index-url https://download.pytorch.org/whl/cu121
```

③安装剩余依赖包

在 VSCode 中激活虚拟环境后，在 VSCode 终端中执行 pip 命令安装依赖包，安装 BERT 模型依赖、数据处理与可视化工具。

```
(base) PS D:\大三\大数据分析实践\实验\bert\BDA_bert> conda activate ml-lab
```

```
(ml-lab) PS D:\大三\大数据分析实践\实验\bert\BDA_bert> pip install transformers
datasets tqdm numpy
```

2、编写项目文件

①test_gpu.py 文件：测试项目的环境配置，本地 GPU、CUDA 版本及可用情况，验证 GPU 设备被 PyTorch 识别，为后续实验开展做准备。

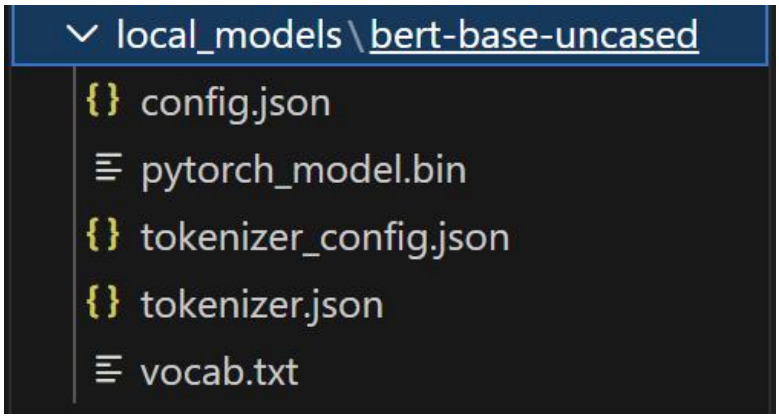
```
(ml-lab) PS D:\大三\大数据分析实践\实验\bert\BDA_bert> python test_gpu.py
=== 环境测试 ===
PyTorch版本: 2.3.1+cu121
Transformers版本: 4.46.3
Datasets版本: 3.1.0

=== GPU信息 ===
CUDA可用: True
GPU名称: NVIDIA GeForce RTX 3060 Laptop GPU
CUDA版本: 12.1
GPU矩阵乘法测试成功!
```

②加载 BERT 模型：由于使用命令行运行代码自动下载时，始终存在代理问题或网络问题：

```
开始加载BERT模型...
'(MaxRetryError("HTTPSConnectionPool(host='huggingface.co', port=443)
: Max retries exceeded with url: /bert-base-uncased/resolve/main/toke
nizer_config.json (Caused by ProxyError('Unable to connect to proxy',
 SSLError(SSLZeroReturnError(6, 'TLS/SSL connection has been closed (
EOF) (_ssl.c:1149)'))))"), '(Request ID: cb283cae-6041-4773-8113-a561
8d574642)')' thrown while requesting HEAD https://huggingface.co/bert
-base-uncased/resolve/main/tokenizer_config.json
Retrying in 1s [Retry 1/5].
```

因此在多次尝试后最终选择在网站手动下载 BERT 模型，如下图：



③verify_model.py 文件：验证手动下载的本地模型是否能够成功加载使用，检查所有必需的模型文件是否存在，尝试加载 tokenizer 和模型参数，执行样例推理测试，并验证 GPU 设备的可用性。

```
(ml-lab) D:\大三\大数据分析实践\实验\bert\BDA_bert>python verify_model.py
=== 验证本地BERT模型 ===
模型路径: ./local_models/bert-base-uncased

检查模型文件:
  config.json: ✓ 存在
  pytorch_model.bin: ✓ 存在
  vocab.txt: ✓ 存在
  tokenizer_config.json: ✓ 存在

尝试加载模型...
✓ Tokenizer加载成功
✓ BERT模型加载成功
✓ 模型推理测试成功
  输入形状: torch.Size([1, 10])
  输出pooler形状: torch.Size([1, 768])
✓ 设备检测: cuda
✓ 模型已移动到GPU

 本地模型验证成功！可以运行main.py进行训练。
```

④data_processor.py 文件：完成本地 MRPC 数据集的预处理和加载工作。它解析原始的数据文件格式，首先从 msr_paraphrase_data.txt 构建句子 ID 到文本内容的映射关系，然后读取训练集和测试集文件，将数据整理成标准化的字典格式。

```
def load_sentence_mapping(self):
    """加载句子ID到文本的映射"""
    print("加载句子映射...")
    df_data = pd.read_csv(self.data_file, sep='\t', encoding='utf-8', quoting=3)
    sentence_map = {}
    for _, row in df_data.iterrows():
        sentence_id = row['Sentence ID']
        sentence_text = row['String']
        sentence_map[sentence_id] = sentence_text
```

```
def load_dataset(self, split='train'):
    """加载训练集或测试集"""
    if split == 'train':
        file_path = self.train_file
    else:
        file_path = self.test_file

    print(f"加载{split}数据集: {file_path}")

    # 读取数据文件
    df = pd.read_csv(file_path, sep='\t', encoding='utf-8', quoting=3, skiprows=1,
                     names=['Quality', '#1 ID', '#2 ID', '#1 String', '#2 String'])

    # 加载句子映射
    sentence_map = self.load_sentence_mapping()
```

⑤MRPCDataset.py 文件：实现 PyTorch 数据集类的封装，将处理好的数据包装成 PyTorch 框架可识别的 Dataset 格式。通过实现__len__和__getitem__方法，它使得数据可以被 DataLoader 批量加载，为模型训练提供标准化的数据迭代接口，支持随机打乱和批量处理等功能。

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

def __getitem__(self, idx):
    sample = self.samples[idx]
    sentence1 = sample['sentence1']
    sentence2 = sample['sentence2']
    label = sample['label']
    return sentence1, sentence2, torch.tensor(label, dtype=torch.float)
```

⑥FCModel.py 文件：定义了全连接分类器模型，作为 BERT 模型顶部的分类层。它接收 BERT 输出的 768 维特征向量，通过一个包含两个全连接层、ReLU 激活函数和 Dropout 层的神经网络，将高维特征映射到二分类输出，最终使用 Sigmoid 函数输出 0-1 之间的概率值，用于

判断句子对是否同义。

```
def __init__(self, input_dim=768, hidden_dim=256, output_dim=1):
    super(FCModel, self).__init__()
    self.classifier = nn.Sequential(
        nn.Linear(input_dim, hidden_dim),
        nn.ReLU(),
        nn.Dropout(0.1),
        nn.Linear(hidden_dim, output_dim),
        nn.Sigmoid()
    )
```

⑦main.py 文件：作为项目的主程序，整合了整个 BERT 实验的完整流程。完成的操作主要有配置实验环境、检测 GPU 设备、加载数据和模型、执行训练循环，包括前向传播、损失计算、反向传播和参数更新等步骤，同时监控训练过程中的损失和准确率变化，确保实验顺利进行。

3、运行主程序

完成项目搭建和环境配置后，运行主程序，如下图，可以看到本地设备使用情况，并且成功加载下载到本地的 MPRC 数据集和 BERT 模型，能够执行训练步骤。

```
(ml-lab) D:\大三\大数据分析实践\实验\bert\BDA_bert>python main.py
=== 基于本地MRPC数据集和本地BERT模型的实验 ===
使用设备: cuda
GPU名称: NVIDIA GeForce RTX 3060 Laptop GPU
CUDA版本: 12.1

开始加载本地MRPC数据集...
加载train数据集: data\msr_paraphrase_train.txt
加载句子映射...
加载了 10948 个句子映射
train数据集加载完成，共 4076 个样本
train数据集: 共4076个样本
数据载入完成，共4076个训练样本

开始加载本地BERT模型...
找到所有必要的模型文件，开始加载...
Tokenizer加载成功!
BERT模型加载完成
全连接层模型创建完成

开始训练...
```

在每个训练轮次中，能够看到每个批次的损失值和准确率，完成一个轮次训练后，输出平均损失和准确率：

```
Batch 500/510, Loss: 0.4012, Acc: 0.8750
Epoch 1 完成:
平均损失: 0.5540
平均准确率: 0.7154
耗时: 56.19秒
```

```
Batch 500/510, Loss: 0.2792, Acc: 0.7500
Epoch 2 完成:
平均损失: 0.3445
平均准确率: 0.8489
耗时: 66.21秒
```

```
Batch 500/510, Loss: 0.0191, Acc: 1.0000
Epoch 3 完成:
平均损失: 0.1509
平均准确率: 0.9470
耗时: 115.52秒
```

在经过三轮训练后，可以看到损失值明显降低，准确率显著提升，说明 BERT 预训练模型在 MRPC 同义预测任务上的微调效果显著：模型通过学习训练集中句子对的语义关联，逐步优化了特征提取能力与分类决策逻辑，对“句子对是否同义”的判断精度不断提高，同时损失值的下降表明模型预测结果与真实标签的偏差持续减小，整体训练过程收敛有效。

结论分析与体会：

1、本次实验成功完成本地 GPU 运行环境搭建(Anaconda+虚拟环境+ PyTorch+Transformer)，解决了模型自动下载的网络问题，验证了 GPU 加速的可行性（模型训练全程使用 cuda 设备），为机器学习工程实践奠定了环境基础。

2、基于 BERT-base-uncased 与全连接分类器的同义预测模型，在 MRPC 训练集上经过 3 轮微调后，平均准确率从 71.54% 提升至 94.70%，损失值从 0.5540 降至 0.1509，说明预训练 BERT 模型具备强大的文本语义提取能力，能够有效捕捉句子对的深层语义关联；全连接分类器能够将 BERT 输出的 768 维特征精准映射到二分类结果，且通过 Adam 优化器的参数更新，模型逐步收敛到较优状态。

3、通过编写数据集封装(MRPCDataset)、数据预处理(data_processor)、分类器(FCModel)与训练逻辑（main.py），深入理解了 BERT 微调的完整流程：

①数据层面：将原始文本解析为“ID-文本映射”，再封装为 PyTorch 可迭代的 Dataset 格式；

②模型层面：BERT 的 pooler 输出能够作为语义特征，通过全连接层转换为分类结果；

③训练层面：需要分别优化 BERT 与分类器参数，通过损失函数与准确率监控模型状态，实现“预训练模型微调”。