

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: 202300130003	姓名: 肖皓天	班级: 数据 23		
实验题目: bert 实践				
实验学时: 32	实验日期: 2025/11/26			
实验目标:				
基于 BERT 的 MRPC 数据集微调实验步骤				
流程描述:				
<p>1. 实验环境配置与设备选择 检查 <code>torch.cuda</code> 可用性，并将计算设备对象赋值给 <code>device</code> 变量。</p> <pre>device = torch.device("cuda" if torch.cuda.is_available() else "cpu") print(f"使用设备: {device}")</pre>				
<p>2. 预训练模型与分词器的加载 本实验选用 Google 发布的 <code>bert-base-uncased</code> (不区分大小写的英文基础版 BERT) 作为预训练底座。 实例化 <code>BertTokenizer</code>, 用于将文本转换为模型可读的 Token ID。 实例化 <code>BertForSequenceClassification</code>, 该模型在 BERT 主干后自动添加了一个全连接层用于分类。 设置 <code>num_labels=2</code>, 因为 MRPC 任务是判断两个句子是否语义相同 (是/否) 的二分类任务。 将模型加载至计算设备 (GPU)。</p> <pre>model_name = "bert-base-uncased" tokenizer = BertTokenizer.from_pretrained(model_name) model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2) model.to(device)</pre>				
<p>3. 数据集加载 实验采用 GLUE 基准测试中的 MRPC (Microsoft Research Paraphrase Corpus) 数据集, 该数据集包含成对的句子及其语义等价性标签。</p> <p>使用 Hugging Face 的 `datasets` 库在线加载 MRPC 数据集。</p> <pre>dataset_name = "glue" subset_name = "mrpc" raw_datasets = load_dataset(dataset_name, subset_name)</pre>				
<p>4. 数据预处理 (分词与编码) 原始文本无法直接输入模型, 需经过分词、截断和填充处理。 定义 <code>tokenize_function</code>, 同时输入两个句子 (`sentence1` 和 `sentence2`)。 设置 <code>truncation=True</code> 和 <code>padding="max_length"</code>, 并将最大长度统一限制为 128, 确保 Batch 内张量维度一致。</p>				

使用 `.map()` 方法以批处理方式 (`batched=True`) 对整个数据集进行快速映射。

```
def tokenize_function(examples):
    return tokenizer(examples["sentence1"], examples["sentence2"], truncation=True, padding="max_length", max_length=128)

tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
```

5. 数据格式化与列清洗

为了适配 PyTorch 模型和训练循环，需要对数据列进行清洗和重命名。

移除冗余列：删除原始文本列 `sentence1`、`sentence2` 和索引列 `idx`，只保留模型需要的 `input_ids`、`attention_mask` 等。

重命名标签列：将数据集中的 `label` 重命名为 `labels`，这是 Hugging Face `BertForSequenceClassification` 计算 Loss 时默认寻找的参数名。

设置张量格式：将数据格式转换为 PyTorch Tensor。

```
tokenized_datasets = tokenized_datasets.remove_columns(["sentence1", "sentence2", "idx"])
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
tokenized_datasets.set_format("torch")
```

6. 数据加载器 (DataLoader) 构建

为了实现小批量随机梯度下降 (Mini-batch SGD)，需要构建数据加载器。

将处理后的数据集划分为训练集 (train) 和验证集 (validation)。

训练集采样：使用 `RandomSampler` 打乱数据顺序，防止模型记忆数据顺序，增强泛化能力。

验证集采样：使用 `SequentialSampler` 按顺序采样，便于评估。

设置批量大小 (`batch_size`) 为 16。

```
batch_size = 16
train_dataset = tokenized_datasets["train"]
eval_dataset = tokenized_datasets["validation"]

train_dataloader = DataLoader(train_dataset, sampler=RandomSampler(train_dataset), batch_size=batch_size)
eval_dataloader = DataLoader(eval_dataset, sampler=SequentialSampler(eval_dataset), batch_size=batch_size)
```

7. 优化器与评估指标配置

优化器：选用 PyTorch 原生的 AdamW (Adam with Weight Decay Fix) 优化器，学习率设为 `5e-5`，这是 BERT 微调的经典参数。

评估指标：使用 evaluate 库加载 GLUE-MRPC 的标准评估指标（包含 Accuracy 和 F1 Score）。

```
# 6. 设置优化器
optimizer = AdamW(model.parameters(), lr=5e-5)

# 7. 加载评估指标 (修复点：使用 evaluate.load)
metric = evaluate.load("glue", subset_name)
```

8. 微调循环逻辑实现

实验设置训练轮次 (epochs) 为 5 轮。每一轮包含“训练”和“评估”两个阶段。

```

epochs = 5
for epoch in range(epochs):
    print(f"\n===== Epoch {epoch + 1} / {epochs} =====")
    model.train()
    total_loss = 0

    for batch in tqdm(train_dataloader, desc="训练"):
        optimizer.zero_grad()
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["labels"].to(device)

        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        total_loss += loss.item()

        loss.backward()
        optimizer.step()

    avg_train_loss = total_loss / len(train_dataloader)
    print(f" 训练平均损失: {avg_train_loss:.4f}")

    model.eval()
    eval_loss = 0
    predictions = []
    references = []

    for batch in tqdm(eval_dataloader, desc="评估"):
        with torch.no_grad():
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["labels"].to(device)

            outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
            loss = outputs.loss
            eval_loss += loss.item()
            logits = outputs.logits

            preds = torch.argmax(logits, dim=-1)
            predictions.extend(preds.cpu().numpy())
            references.extend(labels.cpu().numpy())

    avg_eval_loss = eval_loss / len(eval_dataloader)
    print(f" 评估平均损失: {avg_eval_loss:.4f}")

    # 计算准确率
    results = metric.compute(predictions=predictions, references=references)
    print(f" 评估准确率: {results['accuracy']:.4f}")

```

9. 微调 bert 结果

```

===== Epoch 1 / 5 =====
训练: 100% | 230/230 [00:11<00:00, 19.86it/s]
训练平均损失: 0.5444
评估: 100% | 26/26 [00:00<00:00, 84.71it/s]
评估平均损失: 0.4190
评估准确率: 0.8162

===== Epoch 2 / 5 =====
训练: 100% | 230/230 [00:11<00:00, 20.16it/s]
训练平均损失: 0.3377
评估: 100% | 26/26 [00:00<00:00, 87.87it/s]
评估平均损失: 0.3492
评估准确率: 0.8431

===== Epoch 3 / 5 =====
训练: 100% | 230/230 [00:10<00:00, 21.00it/s]
训练平均损失: 0.1703
评估: 100% | 26/26 [00:00<00:00, 87.16it/s]
评估平均损失: 0.4104
评估准确率: 0.8431

===== Epoch 4 / 5 =====
训练: 100% | 230/230 [00:11<00:00, 20.45it/s]
训练平均损失: 0.0872
评估: 100% | 26/26 [00:00<00:00, 87.83it/s]
评估平均损失: 0.6301
评估准确率: 0.8505

===== Epoch 5 / 5 =====
训练: 100% | 230/230 [00:11<00:00, 20.39it/s]
训练平均损失: 0.0620
评估: 100% | 26/26 [00:00<00:00, 87.61it/s]
评估平均损失: 0.5188
评估准确率: 0.8333

微调完成!
(bert_finetune) root@autodl-container-5776409e39-5f15fc22:~/autodl-tmp/lab5# 

```

结论分析与体会：

本次实验成功基于 BERT 预训练模型 (bert-base-uncased) 在 MRPC 数据集上完成了微调任务。本次实验不仅加深了我对 Transformer 架构和 BERT 模型的理解，更让我在深度学习工程实践方面积累了宝贵经验：