中国科学技术大学　微电子学院

《神经网络及其应用》实验报告三

姓名　杨翊麟　　**学号**　BC24219010

时间　2025/05/11　　**指导教师**　徐奇

实验名称：Transformers 实现 IMDB 影评数据集情感识别

一、实验目的

1、训练 Transformers 序列神经网络，利用预训练的 BERT 模型对 IMDB 影评数据集进行情感分类任务；
2、熟悉 Transformers 的 Self-Attention 自注意力机制；
3、熟悉 BERT 预训练模型双向编码、双向理解的特性以及 Tokenizer 分词器将原始文本分为 token（子词）ID 的方法。

二、实验原理

1、BERT 预训练模型特点：

• 双向编码：相比传统从左到右或从右到左的语言模型，BERT 在预训练时同时考虑词汇左边和右边的上下文，实现了真正的深层双向理解。
• 只使用 Transformer 编码器：BERT 只用 Transformer 的编码器部分来处理输入（不包括解码器）。
• 通过“预训练 + 微调”的方式使用：BERT 先在大规模语料上预训练，然后针对具体任务进行微调（fine-tuning）。
• 掩码语言模型：随机遮盖输入中的一些词（如将 15% 的 token 替换为 [MASK]），训练模型预测被遮盖的词。

2、 Tokenizer 分词器特点：

• Wordpiece 子词分词算法：不直接按词（word）分，而是将罕见词拆分为常见的子词单位。例如“unhappiness” 可以被拆成 ["un", "##happi", "##ness"]，## 表示这是前面词的一部分。
• 分词器的输出：分词器通常将文本转为以下几项：
1) *input_ids*：每个 token 对应的整数 ID。
2) *token_type_ids*：用于 NSP 任务，区分第一句和第二句。
3) *attention_mask*：标记哪些 token 是实际输入，哪些是填充（padding）。
4) *tokens*：原始的 token 列表（可以用于可视化）。

三、实验步骤概述（transformers_imdb_sentiment_classification.py）

（1）引入所需的库和定义参数

```
# 引入核心库
import torch
from torch.utils.data import DataLoader
from transformers import BertTokenizer,
BertForSequenceClassification, get_scheduler
from torch.optim import AdamW
from datasets import load_dataset
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
from tqdm import tqdm
import matplotlib.pyplot as plt
import os

# 设置模型和训练的关键参数
MODEL_NAME = "bert-base-uncased"  # 使用 BERT 英文基础模型
MAX_LENGTH = 256                  # 最大文本长度（截断处理）
BATCH_SIZE = 32                   # 每个 batch 包含的样本数量（适配本机 8GB
GPU显存）
NUM_EPOCHS = 3                    # 训练轮数
LEARNING_RATE = 2e-5               # 学习率
DEVICE = torch.device("cuda" if torch.cuda.is_available() else
"cpu")    # 使用 GPU 加速训练速度
print(f"[INFO] 当前使用设备: {DEVICE}")

# 创建目录用于保存模型和图片
os.makedirs("output", exist_ok=True)
```

（2）加载 IMDB 数据集与预训练分词器

```
# 下载并加载 IMDB 数据集，包含 train/test 两部分
print("\n[INFO] 加载 IMDB 数据集...")
dataset = load_dataset("imdb")
# 加载预训练分词器
tokenizer = BertTokenizer.from_pretrained(MODEL_NAME)
print("[INFO] 对文本进行分词和编码...")

# 定义分词函数，对输入文本进行截断和填充
def tokenize_function(example):
    return tokenizer(
        example["text"],
        padding="max_length",
```

```
        truncation=True,
        max_length=MAX_LENGTH
    )
# 批量处理整个训练集和测试集（提高效率）
tokenized_datasets = dataset.map(tokenize_function, batched=True)
# 设置数据格式为 PyTorch 可接受的格式
tokenized_datasets.set_format(type="torch",    columns=["input_ids",
"attention_mask", "label"])
```

（3）构建 DataLoader 进行批量训练，加载预训练 BERT 模型并设置优化器

```
# 使用 PyTorch 的 DataLoader 封装训练和测试数据
print("[INFO] 构建 DataLoader...")
train_loader = DataLoader(tokenized_datasets["train"],
batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(tokenized_datasets["test"],
batch_size=BATCH_SIZE)

# 加载预训练 BERT 模型
print("[INFO] 加载预训练 BERT 模型...")
model = BertForSequenceClassification.from_pretrained(MODEL_NAME,
num_labels=2)
model.to(DEVICE)
optimizer = AdamW(model.parameters(), lr=LEARNING_RATE)

# 设置学习率调度器
num_training_steps = NUM_EPOCHS * len(train_loader)
scheduler = get_scheduler("linear", optimizer=optimizer,
num_warmup_steps=0, num_training_steps=num_training_steps)
```

（4）训练模型（支持混合精度 AMP）

```
print("[INFO] 开始训练模型...")
model.train()
scaler = torch.amp.GradScaler(device="cuda")
train_losses = []

for epoch in range(NUM_EPOCHS):
    total_loss = 0
    loop = tqdm(train_loader, desc=f"Epoch {epoch+1}/{NUM_EPOCHS}")
    for batch in loop:
        batch["labels"] = batch.pop("label")
        batch = {k: v.to(DEVICE) for k, v in batch.items()}

        optimizer.zero_grad()
```

```
        with torch.amp.autocast(device_type="cuda"):
            outputs = model(**batch)
            loss = outputs.loss

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        scheduler.step()  # 更新学习率

        total_loss += loss.item()
        loop.set_postfix(loss=loss.item())

    avg_loss = total_loss / len(train_loader)
    train_losses.append(avg_loss)
    print(f"Epoch {epoch+1} 平均损失: {avg_loss:.4f}\n")

# 保存训练后的模型
model.save_pretrained("output/bert_imdb_model")
tokenizer.save_pretrained("output/bert_imdb_model")
```

（5）可视化训练损失曲线，并在测试集上进行模型评估，显示混淆矩阵

```
plt.figure(figsize=(8, 4))
plt.plot(range(1, NUM_EPOCHS + 1), train_losses, marker='o',
color='b')
plt.title("Training loss changes with Epoch")
plt.xlabel("Epoch")
plt.ylabel("Average training loss")
plt.grid(True)
plt.tight_layout()
plt.savefig("output/training_loss_curve.png")
plt.show()

# 模型评估
print("[INFO] 开始模型测试...")
model.eval()
preds, labels = [], []

with torch.no_grad():
    for batch in tqdm(test_loader, desc="Evaluating"):
        batch["labels"] = batch.pop("label")
        batch = {k: v.to(DEVICE) for k, v in batch.items()}
        outputs = model(**batch)
        predictions = torch.argmax(outputs.logits, dim=-1)
        preds.extend(predictions.cpu().numpy())
```

```
        labels.extend(batch["labels"].cpu().numpy())

# 计算准确率
acc = accuracy_score(labels, preds)
print(f"\n[RESULT] 测试集准确率: {acc * 100:.2f}%")

# 显示混淆矩阵
cm = confusion_matrix(labels, preds)
cmd = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=["negative", "positive"])
cmd.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.savefig("output/confusion_matrix.png")
plt.show()
```

四、实验结果分析

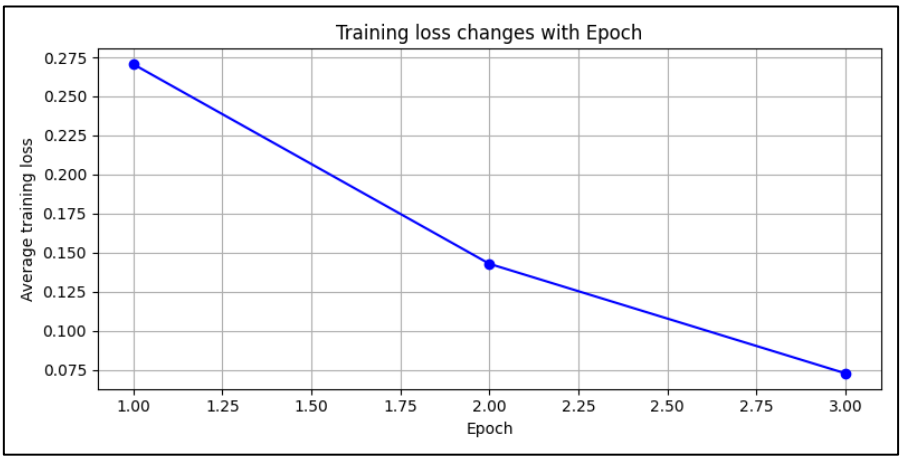
（1）运行结果图

这段运行结果展示了使用 BERT 模型在 IMDB 影评情感分类任务中的完整训练与评估过程。

在训练阶段，模型共迭代 3 个 epoch，训练损失从第 1 个 epoch 的 0.2706 逐步下降到第 3 个 epoch 的 0.0727，表明模型在学习过程中逐渐收敛，性能持续提升。分词与编码速度稳定，且整个训练过程在 GPU (cuda) 上运行，加快了运算效率。

最终，在测试集上的准确率达到了 **92.22%**，说明该模型具备很强的情感识别能力，能有效区分正面与负面影评。

（2）训练损失随 Epoch 变化图

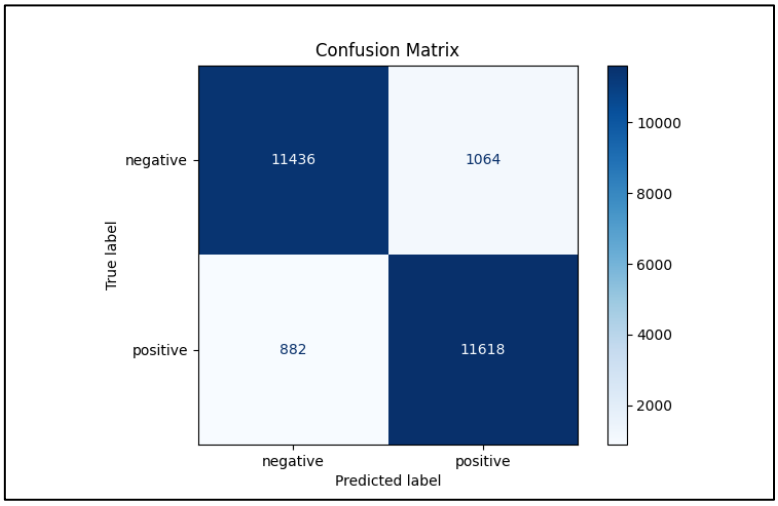


该图展示了模型在 3 个训练周期中平均损失值的变化：

- 第 1 个 epoch：平均损失约为 0.2706
- 第 2 个 epoch：迅速下降至 0.1428
- 第 3 个 epoch：进一步下降至 0.0727

曲线呈明显下降趋势，表明随着训练的进行，模型对训练数据的拟合程度逐步增强，损失函数有效优化。训练过程稳定、无过拟合迹象，是良好训练效果的体现。

（3）混淆矩阵图



该混淆矩阵用于评估模型在 IMDB 测试集上的分类性能。结果显示：

- **真正例（True Negative）**：模型正确地将 11,436 条负面影评预测为负面。
- **真反例（True Positive）**：模型正确地将 11,618 条正面影评预测为正面。
- **假正例（False Positive）**：模型错误地将 1,064 条负面影评预测为正面。
- **假反例（False Negative）**：模型错误地将 882 条正面影评预测为负面。

模型在两个类别上的表现较为平衡，误判率较低，体现出良好的二分类能力。这与测试集准确率 92.22% 的整体评估结果一致。

五、实验总结

本实验基于预训练的 BERT 模型，完成了 IMDB 英文影评情感分类任务。通过加载 bert-base-uncased 模型，并对影评文本进行编码、训练和评估，实验成功实现了高精度的二分类任务。

训练过程中模型损失函数值显著下降，从初始的 0.2706 降至最后一个 epoch 的 0.0727，训练曲线呈现稳定、持续下降的趋势，说明模型收敛良好、训练有效，并未出现明显的过拟合。

在测试集上的准确率达到了 **92.22%**，表明模型对情感分类任务具有良好的泛化能力。

通过本实验，使我对于 Transformers 自注意力机制对序列文本处理高有效性的特点有了更深入的认识，同时也让我对机器学习中自然语言处理 (NLP) 的分支产生了更多兴趣，希望未来出现更优秀的模型让更多丰富的应用得以实现和落地。