HW-3: RNN

姓名：罗威　学号：SA24218095

1. 整体定义及输出总结

1.1 整体定义

本次作业旨在使用`transformer`架构的BERT模型进行下游任务训练，以实现对IMDB影评数据集进行标签预测。通过调整训练参数和优化策略，提高模型的预测准确率，并观察学习曲线的变化。

1.2 实验环境

- **编程语言**：`Python`
- **PyTorch**：`2.5.0+cu124`
- **数据集**：`IMDB-v1`
- **硬件设备**：`GPU: RTX 4090`
- **设置环境变量以方便下载模型**：`export HF_ENDPOINT=https://hf-mirror.com`

1.3 程序输出及分析

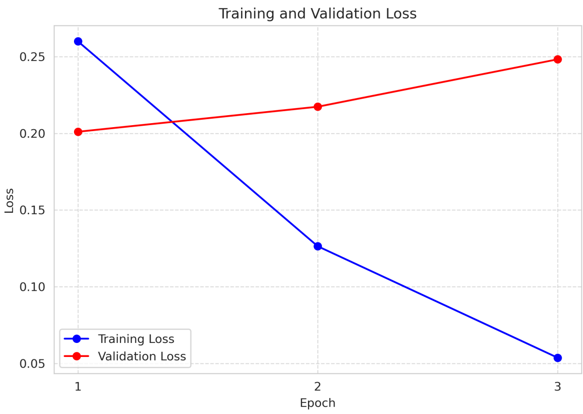
（1）程序输出：代码能够对`bert_base_uncased`模型进行下游任务微调训练，通过训练与验证交替的方式，最终在测试集上取了**92.44%**的准确率，具体输出如下：

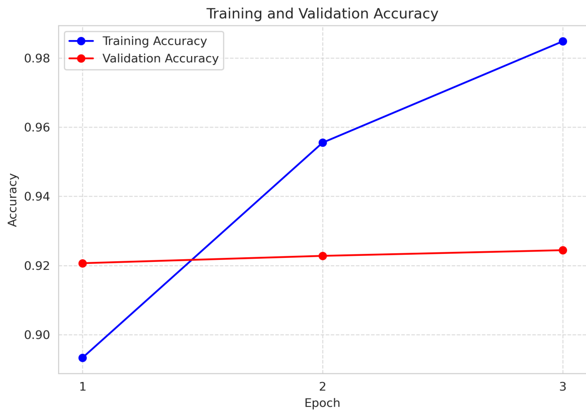
```
PyTorch Version: 2.5.0
CUDA Available : True
CUDA Version   : 12.4
Using device: cuda

1. Loading IMDB dataset and tokenizer:
Dataset loaded. Structure: DatasetDict(...)
Map: 100%|50000/50000

2. Loading pre-trained Transformer model: bert-base-uncased
Model loaded and moved to device.
Optimizer and scheduler seted.

3. Training start:
Training Progress:
--- Epoch 1/3 ---ss:   0%|| 0/4689 [00:00<?, ?it/s]
Training Progress:   1%|█ | 50/4689 [00:08<13:28,  5.74it/s]
    Epoch 1, Batch 50/1563, Loss: 0.3296
...
Training Progress:  33%|...| 1562/4689 [03:40<03:39, 14.25it/s]
  Average Training Loss in Epoch 1: 0.2600
  Average Training Loss: 0.2600, Training Accuracy: 0.8934
Training Progress:  33%|...| 1563/4689 [03:55<03:39, 14.25it/s
 Evaluation Loss: 0.2010, Evaluation Accuracy: 0.9207
            | 1561/4689 [04:12<01:04, 48.85it/s]

--- Epoch 2/3 ---ss:   0%|| 0/4689 [00:00<?, ?it/s]
Training Progress:  34%|...| 1612/4689 [04:16<03:38, 14.09it/s]
```

```
    Epoch 2, Batch 50/1563, Loss: 0.0186
...
  Average Training Loss: 0.1264, Training Accuracy: 0.9555
Training Progress:  67%|...| 3126/4689 [06:15<01:47, 14.52it/s
 Evaluation Loss: 0.2173, Evaluation Accuracy: 0.9228
            | 1561/4689 [02:21<01:04, 48.86it/s]

--- Epoch 3/3 ---ss:   0%|| 0/4689 [00:00<?, ?it/s]
Training Progress:  68%|...| 3175/4689 [06:37<01:47, 14.03it/s]
    Epoch 3, Batch 50/1563, Loss: 0.0141
...
Training Progress: 100%|...| 4689/4689 [08:24<00:00, 14.54it/s]
  Average Training Loss in Epoch 3: 0.0537
  Average Training Loss: 0.0537, Training Accuracy: 0.9848
Training Progress: 100%|...| 4689/4689 [08:35<00:00, 14.54it/s
 Evaluation Loss: 0.2482, Evaluation Accuracy: 0.9244

Training Progress: 100%|...| 4689/4689 [08:56<00:00,  8.75it/s]
Training complete.

4. Final evaluation on the test dataset:
Final Evaluation: 100%|...| 1563/1563 [00:31<00:00, 48.89it/s]
Test Accuracy: 0.9244

5. Saving model to ./model_for_imdb
Model and tokenizer saved.

6. Generating training visualization:
Training visualization saved to ./model_for_imdb/training_history.png
```

(2) 学习曲线：绘制训练损失和准确率随epoch的变化曲线，并保存为图像文件如下。

- 如下所示，训练损失随epoch的增加逐渐减小，最终降低到一个较低的水平。
- 训练准确率随epoch的增加逐渐提高，最终达到一个较高的水平。
- 验证集上的准确率与训练结果相似，表明模型具有较好的泛化能力。





2. 代码注释

2.1 导入相应库及参数定义

导入要使用的库，并检查PyTorch以及CUDA是否可用。

```python
import torch
from torch.utils.data import DataLoader
from transformers import AutoTokenizer, AutoModelForSequenceClassification, get_scheduler
from torch.optim import AdamW
from datasets import load_dataset
from sklearn.metrics import accuracy_score
from tqdm.auto import tqdm # For progress bars
import os
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')

# 0. 库版本、CUDA检查
print(f"PyTorch Version: {torch.__version__}")
print(f"CUDA Available : {torch.cuda.is_available()}")
print(f"CUDA Version   : {torch.version.cuda}")

# 1. 定义参数和DEVICE
MODEL_NAME = "bert-base-uncased" # 使用Hugging Face上的预训练模型BERT
MAX_LENGTH = 256                 # 定义分词器的最大序列长度
BATCH_SIZE = 16                  # 训练和验证的批次大小
LEARNING_RATE = 2e-5             # AdamW优化器的学习率
NUM_EPOCHS = 3                   # 训练轮数
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {DEVICE}")

OUTPUT_DIR = "./model_for_imdb" # 保存训练好的模型的路径
```

2.2 数据集加载和预处理

- 使用`Hugging Face`的`datasets`库加载`IMDB`影评数据集；
- 数据集包含三个部分：训练集 (25,000)、测试集 (25,000) 和无标签集 (50,000)；
- 只需要训练集和测试集进行情感分类任务，因此无标签集在后面被移除。
- 对应所有文本进行处理，短序列补0，长序列截断，统一到`MAX_LENGTH`。

```python
print("\nLoading IMDB dataset and tokenizer: ")
raw_datasets = load_dataset("imdb")

print(f"Dataset loaded. Structure: {raw_datasets}")

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True, max_length=MAX_LENGTH)
    # padding='max_length': 把短的序列扩展到MAX_LENTH
    # truncation=True: 把长的序列修剪到MAX_LENTH

```

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

# 移除"text"这栏，把数据转化为PyTorch张量
tokenized_datasets = tokenized_datasets.remove_columns(["text"])
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
tokenized_datasets.set_format("torch")

train_dataset = tokenized_datasets["train"]
eval_dataset = tokenized_datasets["test"]

# 创建DataLoader，按照定义的BATCH_SIZE分割
train_dataloader = DataLoader(train_dataset, shuffle=True, batch_size=BATCH_SIZE)
eval_dataloader = DataLoader(eval_dataset, batch_size=BATCH_SIZE)
```

2.3 加载模型、定义优化器和学习率调度器

- 使用 `AutoModelForSequenceClassification` 加载预训练 `BERT` 模型；
- 添加分类头（`num_labels=2`）用于二分类任务；
- 模型包含：BERT 基础架构（12 层 Transformer 编码器）、池化层（提取标记表示）、分类层（将特征映射到2个类别）；
- 模型参数通过微调进行更新。
- `AdamW` 优化器：适用于大型预训练语言模型的微调，结合了 `Adam` 优化器的自适应学习率和权重衰减正则化；
- 学习率调度器：学习率设置为线性衰减，不使用预热。

```python
print(f"\nLoading pre-trained Transformer model: {MODEL_NAME}")

# 标签分类为positive (1) and negative (0)，所以设置num_labels=2
model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=2)
model.to(DEVICE)
print("Model loaded and moved to device.")

optimizer = AdamW(model.parameters(), lr=LEARNING_RATE)

num_training_steps = NUM_EPOCHS * len(train_dataloader)
lr_scheduler = get_scheduler(
    name="linear", # 对transformer的常规配置
    optimizer=optimizer,
    num_warmup_steps=0, # (0,0.1)
    num_training_steps=num_training_steps
)
print("Optimizer and scheduler seted.")
```

2.4 模型训练流程

- 分为训练和评估两个阶段，每个 `EPOCH` 都要进行；
- 评估的时候会收集预测结果，计算准确率和损失；
- 训练期间会记录损失和准确率。

```python
print("\nTraining start:")
progress_bar_train = tqdm(range(num_training_steps), desc="Training Progress")
progress_bar_eval = tqdm(range(NUM_EPOCHS * len(eval_dataloader)), desc="Evaluation Progress", leave=False)

# 创建历史记录字典
history = {
    'train_loss': [],
    'val_loss': [],
    'train_acc': [],
    'val_acc': []
}

for epoch in range(NUM_EPOCHS):
    model.train()
    total_loss_train = 0
    train_preds = []
    train_labels = []
    print(f"\n--- Epoch {epoch + 1}/{NUM_EPOCHS} ---")

    for batch_num, batch in enumerate(train_dataloader):
        # (1)把数据移动到GPU
        batch = {k: v.to(DEVICE) for k, v in batch.items()}
        # (2)清除之前的梯度
        optimizer.zero_grad()
        # (3)前向传播
        outputs = model(**batch)
        loss = outputs.loss
        total_loss_train += loss.item()

        # (4)记录训练预测结果
        logits = outputs.logits
        preds = torch.argmax(logits, dim=-1)
        train_preds.extend(preds.cpu().numpy())
        train_labels.extend(batch["labels"].cpu().numpy())
        # (5)反向传播
        loss.backward()
        optimizer.step()
        # (6)更新学习率
        lr_scheduler.step()
        progress_bar_train.update(1)
        # (7)每50个batch记录Loss
        if (batch_num + 1) % 50 == 0:
            print(f"  Epoch {epoch+1}, Batch {batch_num+1}/{len(train_dataloader)}, Loss: {loss.item():.4f}")

    avg_train_loss = total_loss_train / len(train_dataloader)
    print(f"  Average Training Loss in Epoch {epoch + 1}: {avg_train_loss:.4f}")

    # 计算训练准确率
    train_accuracy = accuracy_score(train_labels, train_preds)
    avg_train_loss = total_loss_train / len(train_dataloader)
```

```python
    print(f"  Average Training Loss: {avg_train_loss:.4f}, Training Accuracy: {train_accuracy:.4f}")

    # 每训练一轮的同时验证一轮
    model.eval()
    total_loss_eval = 0
    val_preds = []
    val_labels = []

    with torch.no_grad(): # 测试的时候不需要计算梯度
        for batch in eval_dataloader:
            batch = {k: v.to(DEVICE) for k, v in batch.items()}
            outputs = model(**batch)
            loss = outputs.loss
            total_loss_eval += loss.item()

            logits = outputs.logits
            preds = torch.argmax(logits, dim=-1)
            val_preds.extend(preds.cpu().numpy())
            val_labels.extend(batch["labels"].cpu().numpy())
            progress_bar_eval.update(1)

    val_accuracy = accuracy_score(val_labels, val_preds)
    avg_val_loss = total_loss_eval / len(eval_dataloader)
    print(f"  Evaluation Loss: {avg_val_loss:.4f}, Evaluation Accuracy: {val_accuracy:.4f}")

    # 记录历史
    history['train_loss'].append(avg_train_loss)
    history['val_loss'].append(avg_val_loss)
    history['train_acc'].append(train_accuracy)
    history['val_acc'].append(val_accuracy)

    progress_bar_eval.reset() # 为下一轮验证重置

progress_bar_train.close()
progress_bar_eval.close()
print("Training complete.")
```

2.5 模型测试以及保存

- 按照评估流程进行测试，在测试集上进行全面的评估；
- 使用`save_pretrained()`方法保存模型权重和配置，同时也保存分词器配置。

```python
print("\nFinal evaluation on the test dataset: ")
model.eval()
all_preds_final = []
all_labels_final = []
final_eval_progress = tqdm(eval_dataloader, desc="Final Evaluation")

with torch.no_grad():
    for batch in final_eval_progress:
        batch = {k: v.to(DEVICE) for k, v in batch.items()}
        outputs = model(**batch)
```

```
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)
        all_preds_final.extend(predictions.cpu().numpy())
        all_labels_final.extend(batch["labels"].cpu().numpy())

final_accuracy = accuracy_score(all_labels_final, all_preds_final)
print(f"Test Accuracy: {final_accuracy:.4f}")

# 保存微调的模型以及分词配置
print(f"\nSaving model to {OUTPUT_DIR}")
if not os.path.exists(OUTPUT_DIR):
    os.makedirs(OUTPUT_DIR)
model.save_pretrained(OUTPUT_DIR)
tokenizer.save_pretrained(OUTPUT_DIR)
print("Model and tokenizer saved.")
```

2.6 绘制损失曲线

- 利用`matplotlib`库绘制训练损失和准确率随epoch的变化曲线，并保存为图像文件。

```
print("\nGenerating training visualization:")

# 创建图表
plt.figure(figsize=(14, 5))

# 绘制损失曲线
plt.subplot(1, 2, 1)
plt.plot(history['train_loss'], label='Training Loss', marker='o', color='blue')
plt.plot(history['val_loss'], label='Validation Loss', marker='o', color='red')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.xticks(range(NUM_EPOCHS), [str(i+1) for i in range(NUM_EPOCHS)])
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)

# 绘制准确率曲线
plt.subplot(1, 2, 2)
plt.plot(history['train_acc'], label='Training Accuracy', marker='o', color='blue')
plt.plot(history['val_acc'], label='Validation Accuracy', marker='o', color='red')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.xticks(range(NUM_EPOCHS), [str(i+1) for i in range(NUM_EPOCHS)])
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)

# 调整布局并保存图片
plt.tight_layout()
plt.savefig(f"{OUTPUT_DIR}/training_history.png", dpi=300, bbox_inches='tight')
```

```
print(f"Training visualization saved to {OUTPUT_DIR}/training_history.png")
```