实验报告：基于BERT的文本分类

实验目的

熟悉和掌握基于BERT在文本分类任务中的应用

对比分析：

- 端到端微调
- 先预训练后微调
- 蒸馏模型微调

硬件环境

- GPU：8张RTX 4090
- 显存： 48GB
- Python: 3.12 cuda:12.4

一、端到端模型

```
from transformers import BertTokenizer,
BertForSequenceClassification, AdamW,
get_linear_schedule_with_warmup
from torch.utils.data import DataLoader, Dataset
import torch
import torch.nn as nn
import pandas as pd
from tqdm.auto import tqdm
from sklearn.metrics import accuracy_score, f1_score,
precision_score, recall_score
import os
import seaborn as sns
sns.set_style("whitegrid")
```

```python
tokenizer_folder = 'model'
tokenizer = BertTokenizer.from_pretrained(tokenizer_folder)
model = 
BertForSequenceClassification.from_pretrained(tokenizer_folder, 
num_labels=2)
device = torch.device("cuda" if torch.cuda.is_available() else 
"cpu")
model.to(device)


def print_data_info(train, val, test, n=3):
    for name, data in zip(['训练集', '验证集', '测试集'], [train, 
val, test]):
        print(f"{name}大小: {len(data)}\n列名:
{data.columns.tolist()}\n示例:\n{data.sample(n)}\n")



class SentimentDataset(Dataset):
    def __init__(self, data_list, tokenizer, max_length=128):
        self.data = data_list
        self.tokenizer = tokenizer
        self.max_length = max_length
    def __len__(self):
        return len(self.data)
    def __getitem__(self, idx):
        sentense, label = self.data.iloc[idx]['sentence'], 
self.data.iloc[idx]['label']
        inputs = self.tokenizer(sentense, 
add_special_tokens=True, max_length=self.max_length, 
padding='max_length', truncation=True, return_tensors='pt')
        input_ids = inputs['input_ids'].squeeze()
        attention_mask = inputs['attention_mask'].squeeze()
        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'labels': torch.tensor(label, dtype=torch.long)
        }



train_data = pd.read_parquet("data/sst2/data/train-00000-of-
00001.parquet")
val_data = pd.read_parquet("data/sst2/data/validation-00000-of-
00001.parquet")
test_data_r  = pd.read_parquet("data/sst2/data/test-00000-of-
00001.parquet")
print('改进前的test_data\n')
print_data_info(train_data,val_data,test_data_r)
```

```python
'''
由于sst-2数据集的test集没有label，所以考虑对数据集进行处理，将train_data做split
error:test_data = pd.read_parquet("data/sst2/data/test-00000-of-00001.parquet")
# 测试集示例：
        idx                                            sentence  label
541     541  warmed-over tarantino by way of wannabe elmore ...     -1
1577   1577  lacking gravitas , macdowell is a placeholder  ...     -1
531     531  i regret to report that these ops are just not ...     -1
481     481  when your leading ladies are a couple of scree ...     -1
40       40  it is a kickass , dense sci-fi action thriller ...     -1
'''

# 划分训练集为新的训练集和测试集
from sklearn.model_selection import train_test_split
train_data_new, test_data = train_test_split(train_data, test_size=0.2, random_state=42)

print(train_data_new.columns)  # 确认列名是 'sentence' 还是 'text'

train_data_new = train_data_new.reset_index(drop=True)
test_data = test_data.reset_index(drop=True)
val_data = val_data.reset_index(drop=True)

# 创建数据集
train_dataset = SentimentDataset(train_data_new, tokenizer)
val_dataset = SentimentDataset(val_data, tokenizer)
test_dataset = SentimentDataset(test_data, tokenizer)

# 创建数据加载器
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)


print("改进后的test_data\n")
print_data_info(train_data_new,val_data,test_data)

'''
学习率调度器，分层学习率
'''
```

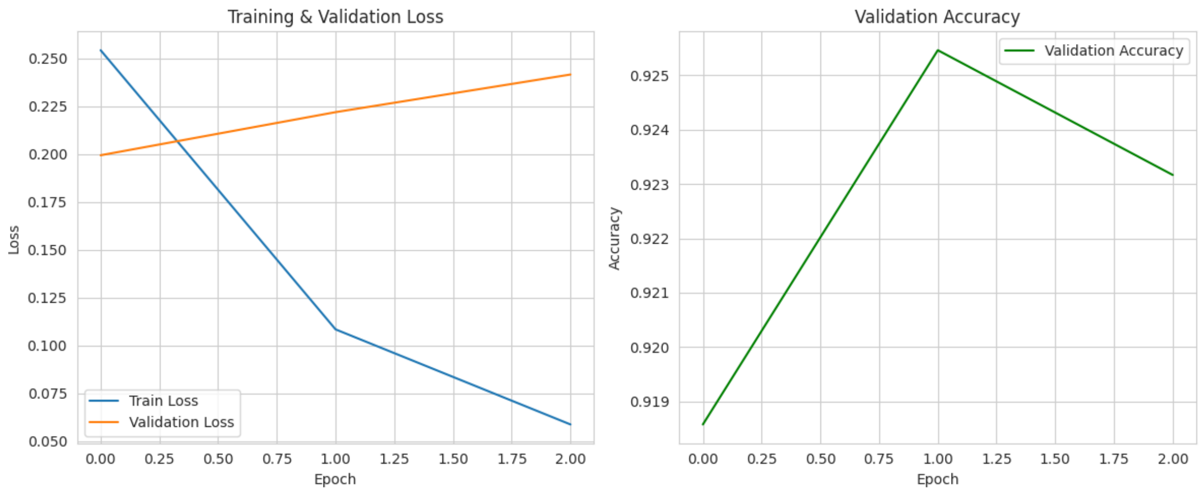
```python
no_decay = ['bias', 'LayerNorm.weight']
optimizer_parameters = [
    {'params': [p for n, p in model.named_parameters() if not
any(nd in n for nd in no_decay)], 'weight_decay': 0.01},
    {'params': [p for n, p in model.named_parameters() if any(nd
in n for nd in no_decay)], 'weight_decay': 0.0}
]
optimizer = AdamW(optimizer_parameters, lr=3e-5)

num_epochs = 3

total_steps = len(train_loader) * num_epochs
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0.1*total_steps,
    num_training_steps=total_steps
)


best_val_accuracy = 0.0
train_losses = []
val_losses = []
val_accuracies = []
global_step = 0
OUTPUT_DIR = "./results" # 模型和训练结果输出目录
PLOTS_DIR = "./plots"   # 图表保存目录

os.makedirs(OUTPUT_DIR, exist_ok=True)
os.makedirs(PLOTS_DIR, exist_ok=True)

'''
BertForSequenceClassification 模型默认使用的是交叉熵损失函数
（CrossEntropyLoss）

'''

for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    progress_bar = tqdm(train_loader, desc=f"Epoch
{epoch+1}/{num_epochs}")

    for batch in progress_bar:
        optimizer.zero_grad()
        batch = {k: v.to(device) for k, v in batch.items()}
        labels = batch['labels']
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()
        total_loss += loss.item()
```

```
        optimizer.step()
        scheduler.step()
        progress_bar.set_postfix({'training_loss':
f'{loss.item():.3f}', 'lr': f'{scheduler.get_last_lr()
[0]:.2e}'})

    avg_loss = total_loss / len(train_loader) # 计算平均损失
    print(f"Epoch {epoch + 1}/{num_epochs}, Loss:
{avg_loss:.4f}")

    train_losses.append(avg_loss) # 记录训练损失

    # 验证阶段
    model.eval()
    total_val_loss = 0
    all_val_preds = []
    all_val_labels = []

    with torch.no_grad():
        for batch in val_loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            val_loss = outputs.loss
            total_val_loss += val_loss.item()

            val_preds = torch.argmax(outputs.logits,
dim=1).cpu().numpy()
            val_labels = batch['labels'].cpu().numpy()
            all_val_preds.extend(val_preds)
            all_val_labels.extend(val_labels)

    avg_val_loss = total_val_loss / len(val_loader)
    val_accuracy = accuracy_score(all_val_labels, all_val_preds)
    val_accuracies.append(val_accuracy)
    val_precision = precision_score(all_val_labels,
all_val_preds, average='weighted')
    val_recall = recall_score(all_val_labels, all_val_preds,
average='weighted')
    val_f1 = f1_score(all_val_labels, all_val_preds,
average='weighted')

    val_losses.append(avg_val_loss)

    print(f"Epoch {epoch + 1}/{num_epochs}, Val Loss:
{avg_val_loss:.4f}, Val Accuracy: {val_accuracy:.4f}, "
          f"Val Precision: {val_precision:.4f}, Val Recall:
{val_recall:.4f}, Val F1: {val_f1:.4f}")

    if val_accuracy > best_val_accuracy:
        print(f" best accuracy {val_accuracy:.4f})")
```

```
            best_val_accuracy = val_accuracy
            model.save_pretrained(OUTPUT_DIR,
safe_serialization=False)


model =
BertForSequenceClassification.from_pretrained(OUTPUT_DIR)
model.to(device)
model.eval()  # 切换到评估模式

import matplotlib.pyplot as plt
import numpy as np

def evaluate(model, data_loader):
    all_preds = []
    all_labels = []
    incorrect_samples = []

    with torch.no_grad():
        for batch in tqdm(data_loader, desc="Testing"):
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            preds = torch.argmax(outputs.logits, dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(batch['labels'].cpu().numpy())

            incorrect_mask = (preds !=
batch['labels']).cpu().numpy()
            incorrect_indices = np.where(incorrect_mask)[0]

            # 解码并保存部分错误样本（最多5个）
            for i in incorrect_indices:
                if len(incorrect_samples) >= 5:
                    break
                incorrect_samples.append({
                    'sentence':
tokenizer.decode(batch['input_ids'][i],
skip_special_tokens=True),
                    'true_label': batch['labels'][i].item(),
                    'pred_label': preds[i].item()
                })

    # 计算指标
    accuracy = accuracy_score(all_labels, all_preds)
    precision = precision_score(all_labels, all_preds,
average='weighted')
    recall = recall_score(all_labels, all_preds,
average='weighted')
    f1 = f1_score(all_labels, all_preds, average='weighted')
```

```
    print("随机输出5个错误案例:")
    for case in incorrect_samples[:5]:
        print(f"Sentence: {case['sentence']}")
        print(f"True Label: {case['true_label']}, Predicted
Label: {case['pred_label']}\n")

    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'preds': all_preds,
        'labels': all_labels,
        # 'incorrect_samples' : incorrect_samples
    }

def plot_metrics(train_losses, val_losses, val_accuracies):
    plt.figure(figsize=(12, 5))

    # Loss曲线
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Train Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training & Validation Loss')
    plt.legend()

    # Accuracy曲线
    plt.subplot(1, 2, 2)
    plt.plot(val_accuracies, label='Validation Accuracy',
color='green')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Validation Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.savefig(f"{PLOTS_DIR}/training_metrics.png")  # 保存图表
    plt.show()


plot_metrics(train_losses, val_losses, val_accuracies)



# 运行测试
test_results = evaluate(model, test_loader)
print(f"""
测试集结果:
```

```
准确率: {test_results['accuracy']:.4f}
精确率: {test_results['precision']:.4f}
召回率: {test_results['recall']:.4f}
F1分数: {test_results['f1']:.4f}
""")
```




```
改进前的test_data

训练集大小: 67349
列名: ['idx', 'sentence', 'label']
示例:
         idx                                              sentence
label
59113  59113          that the new film is a lame kiddie flick
    0
56034  56034  reflect that its visual imagination is breatht...
     1
35121  35121                 most multilayered and sympathetic
    1

验证集大小: 872
列名: ['idx', 'sentence', 'label']
示例:
     idx                                              sentence
label
601  601                              fancy a real downer ?
0
844  844  given how heavy-handed and portent-heavy it is...
0
349  349   ... turns so unforgivably trite in its last 10...
0
```

```
测试集大小: 1821
列名: ['idx', 'sentence', 'label']
示例:
       idx                                           sentence
label
1525  1525  all the well-meaningness in the world ca n't e...
  -1
1392  1392                              go see it and enjoy .
  -1
867    867  though the controversial korean filmmaker 's l...
  -1

Index(['idx', 'sentence', 'label'], dtype='object')
改进后的test_data

训练集大小: 53879
列名: ['idx', 'sentence', 'label']
示例:
         idx                                           sentence
label
23050  14928  his penchant for tearing up on cue -- things t...
     1
41506  65216                                    expanded vision
    1
5581   27335                          its own languorous charm
    1

验证集大小: 872
列名: ['idx', 'sentence', 'label']
示例:
     idx                                           sentence
label
745  745  made with no discernible craft and monstrously ...
0
172  172  it seems like i have been waiting my whole lif...
1
237  237  a by-the-numbers effort that wo n't do much to ...
0

测试集大小: 13470
列名: ['idx', 'sentence', 'label']
示例:
         idx                                           sentence
label
12694  29282                             stunningly unoriginal
    0
6969   13624            that embraces its old-fashioned themes
    1
```

```
2108   61308  to speak about other than the fact that it is ...
     0
```

由于sst-2数据集的test集没有label，所以考虑对数据集进行处理，将train_data做split

随机输出5个错误案例:

```
Sentence: is just the point
True Label: 1, Predicted Label: 0

Sentence: sexy, violent, self - indulgent and maddening
True Label: 0, Predicted Label: 1

Sentence: you'd swear you
True Label: 1, Predicted Label: 0

Sentence: walks a tricky tightrope between being wickedly funny
and just plain wicked
True Label: 0, Predicted Label: 1

Sentence: all but spits out denzel washington's fine performance
in the title role.
True Label: 1, Predicted Label: 0
```

混淆矩阵 confusion matrix: [[386 42] [32 412]]： 可见正负样本的分类效果差距不大

测试结果

```
测试集结果:
准确率: 0.9532
精确率: 0.9534
召回率: 0.9532
F1分数: 0.9532
```

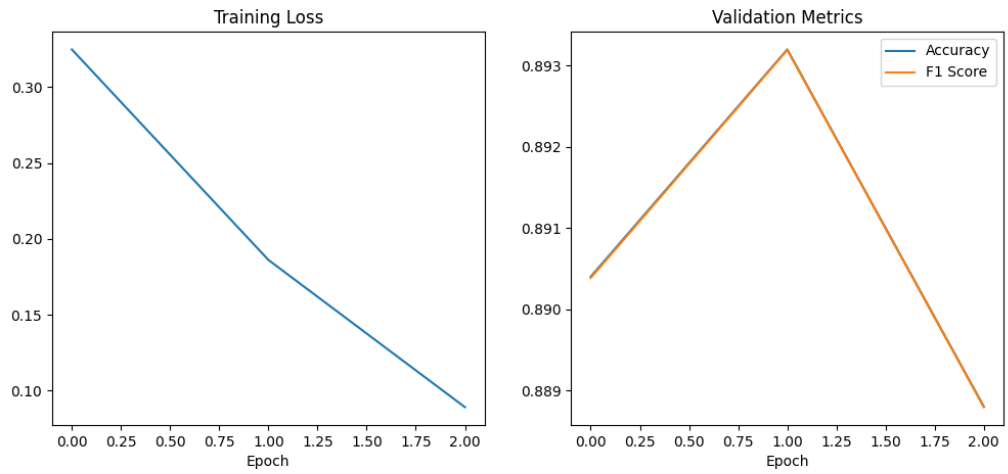
二、训练后微调

原先使用Trainer,但消耗较多显存，故抛弃该做法。

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
```

```python
from transformers import BertTokenizer, BertForMaskedLM, BertModel, get_linear_schedule_with_warmup, DataCollatorForLanguageModeling
from transformers import DataCollatorForLanguageModeling, Trainer, TrainingArguments
from torch.optim import AdamW
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import os
from tqdm.auto import tqdm
from datasets import load_dataset
import pandas as pd
import pynvml

tokenizer_folder = 'model'
tokenizer = BertTokenizer.from_pretrained(tokenizer_folder)
# 这个模型在预训练时使用
mlm_model = BertForMaskedLM.from_pretrained(tokenizer_folder)
device = torch.device('cuda:6')
mlm_model.to(device)

'''
BertForMaskedLM主要由BERT编码器+MaskedLM预测头组成:
可以使用Trainer进行微调,让BERT适应特定领域的词汇预测
'''

dataset = load_dataset(path="data/imdb")["unsupervised"]
data_unsupervised = dataset.train_test_split(test_size=0.2)

# 转换为DataFrame并显示前5条
df = pd.DataFrame(dataset[:5])
print(df[['text']])

# 数据预处理函数
def preprocess_function(examples):
    return tokenizer(examples["text"], truncation=True)

# 检查是否已经保存了处理后的数据

tokenized_data = data_unsupervised.map(
    preprocess_function,
    batched=True,   # batched=True将预处理函数一次应用于多个元素。
    remove_columns=["text", "label"]
)


print(tokenized_data)
```

```
# 创建MLM数据收集器
'''
进行MLM任务时需要使用的数据收集器，该数据收集器会以一定概率（由参数
mlm_probability控制）将序列中的Token替换成Mask标签。
不同于DataCollatorWithPadding、DataCollatorForTokenClassification
和DataCollatorForTokenClassification，该数据收集器只会将序列填充到最长序
列长度。
'''
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm_probability=0.15
)

pretrain_loader = DataLoader(
    tokenized_data["train"],  # 这里需要指定是训练集
    collate_fn=data_collator,
    batch_size=48,
    shuffle=True
)

pretrain_epoch = 1
pretrain_steps = pretrain_epoch * len(pretrain_loader)
pretrain_optimizer = AdamW(mlm_model.parameters(), lr=3e-5)
pretrain_scheduler = get_linear_schedule_with_warmup(
    optimizer=pretrain_optimizer,
    num_warmup_steps=0,
    num_training_steps=pretrain_steps
)

mlm_model.train()
total_pretrain_loss = 0
progress_bar = tqdm(pretrain_loader)
for batch in progress_bar:
    pretrain_optimizer.zero_grad()
    batch = {k: v.to(device) for k, v in batch.items()}
    outputs = mlm_model(**batch)
    loss = outputs.loss
    total_pretrain_loss += loss.item()
    loss.backward()
    pretrain_optimizer.step()
    pretrain_scheduler.step()
    progress_bar.set_postfix({'pretraining_loss':
f'{loss.item():.3f}', 'lr': f'{pretrain_scheduler.get_last_lr()
[0]:.2e}'})

print("预训练完成!模型文件已保存在mlm_results")
mlm_model.save_pretrained('mlm_results',
safe_serialization=False)
```




```
                                                       text
0  This is just a precious little diamond. The pl...
1  When I say this is my favourite film of all ti...
2  I saw this movie because I am a huge fan of th...
3  Being that the only foreign films I usually li...
4  After seeing Point of No Return (a great movie...
```

Final Test Performance:
Accuracy: 0.8886
F1 Score: 0.8886

拓展实验

使用 Focal Loss 处理类别不平衡问题

Focal Loss 可以通过降低容易分类样本的权重，使得模型更加关注难分类的样本，从而有效处理类别不平衡问题。以下是改进后的代码：

```
from transformers import BertTokenizer,
BertForSequenceClassification, AdamW,
get_linear_schedule_with_warmup
from torch.utils.data import DataLoader, Dataset
import torch
import torch.nn as nn
import pandas as pd
from tqdm.auto import tqdm
from sklearn.metrics import accuracy_score, f1_score,
precision_score, recall_score
```

```python
import os
import seaborn as sns
sns.set_style("whitegrid")


# 定义 Focal Loss
class FocalLoss(nn.Module):
    def __init__(self, alpha=0.25, gamma=2):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.criterion = nn.CrossEntropyLoss(reduction='none')

    def forward(self, inputs, targets):
        ce_loss = self.criterion(inputs, targets)
        pt = torch.exp(-ce_loss)
        focal_loss = self.alpha * (1 - pt) ** self.gamma *
ce_loss
        return focal_loss.mean()


tokenizer_folder = 'model'
tokenizer = BertTokenizer.from_pretrained(tokenizer_folder)
model =
BertForSequenceClassification.from_pretrained(tokenizer_folder,
num_labels=2)
device = torch.device("cuda:7")
model.to(device)


def print_data_info(train, val, test, n=3):
    for name, data in zip(['训练集', '验证集', '测试集'], [train,
val, test]):
        print(f"{name}大小: {len(data)}\n列名:
{data.columns.tolist()}\n示例:\n{data.sample(n)}\n")


class SentimentDataset(Dataset):
    def __init__(self, data_list, tokenizer, max_length=128):
        self.data = data_list
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sentense, label = self.data.iloc[idx]['sentence'],
self.data.iloc[idx]['label']
```

```
        inputs = self.tokenizer(sentense,
add_special_tokens=True, max_length=self.max_length,
padding='max_length',
                                truncation=True,
return_tensors='pt')
        input_ids = inputs['input_ids'].squeeze()
        attention_mask = inputs['attention_mask'].squeeze()
        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'labels': torch.tensor(label, dtype=torch.long)
        }


train_data = pd.read_parquet("data/sst2/data/train-00000-of-
00001.parquet")
val_data = pd.read_parquet("data/sst2/data/validation-00000-of-
00001.parquet")
test_data_r = pd.read_parquet("data/sst2/data/test-00000-of-
00001.parquet")
print('改进前的test_data\n')
print_data_info(train_data, val_data, test_data_r)

# 划分训练集为新的训练集和测试集
from sklearn.model_selection import train_test_split

train_data_new, test_data = train_test_split(train_data,
test_size=0.2, random_state=42)

print(train_data_new.columns)  # 确认列名是 'sentence' 还是 'text'

train_data_new = train_data_new.reset_index(drop=True)
test_data = test_data.reset_index(drop=True)
val_data = val_data.reset_index(drop=True)

# 创建数据集
train_dataset = SentimentDataset(train_data_new, tokenizer)
val_dataset = SentimentDataset(val_data, tokenizer)
test_dataset = SentimentDataset(test_data, tokenizer)

# 创建数据加载器
train_loader = DataLoader(train_dataset, batch_size=32,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)

print("改进后的test_data\n")
print_data_info(train_data_new, val_data, test_data)

# 学习率调度器，分层学习率
```

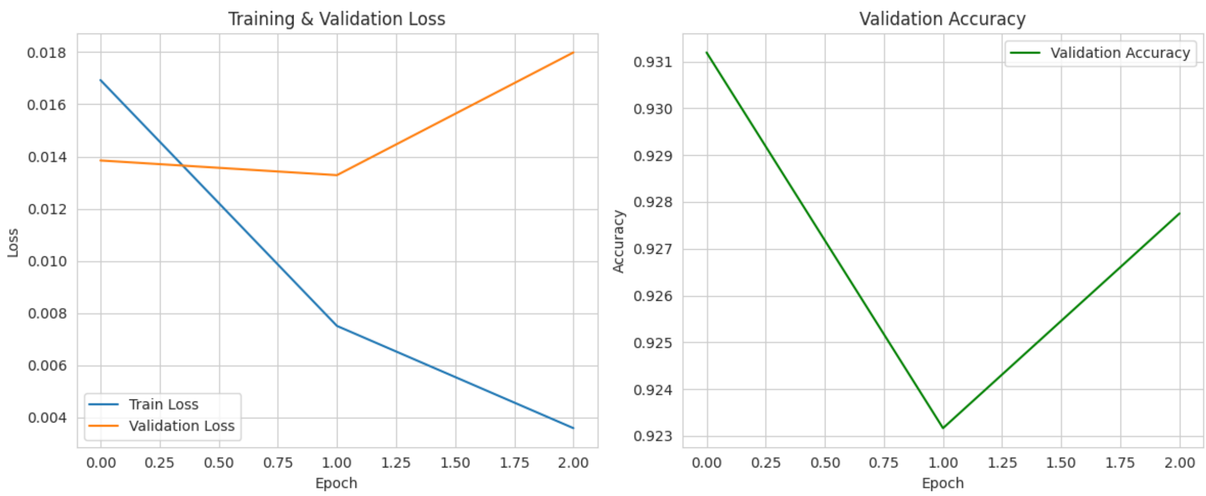
```python
no_decay = ['bias', 'LayerNorm.weight']
optimizer_parameters = [
    {'params': [p for n, p in model.named_parameters() if not
any(nd in n for nd in no_decay)], 'weight_decay': 0.01},
    {'params': [p for n, p in model.named_parameters() if any(nd
in n for nd in no_decay)], 'weight_decay': 0.0}
]
optimizer = AdamW(optimizer_parameters, lr=3e-5)

num_epochs = 3

total_steps = len(train_loader) * num_epochs
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0.1 * total_steps,
    num_training_steps=total_steps
)

best_val_accuracy = 0.0
train_losses = []
val_losses = []
val_accuracies = []
global_step = 0
OUTPUT_DIR = "./results"  # 模型和训练结果输出目录
PLOTS_DIR = "./plots"  # 图表保存目录

os.makedirs(OUTPUT_DIR, exist_ok=True)
os.makedirs(PLOTS_DIR, exist_ok=True)

focal_loss = FocalLoss()

for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    progress_bar = tqdm(train_loader, desc=f"Epoch {epoch +
1}/{num_epochs}")

    for batch in progress_bar:
        optimizer.zero_grad()
        batch = {k: v.to(device) for k, v in batch.items()}
        labels = batch['labels']
        outputs = model(**batch)
        logits = outputs.logits
        loss = focal_loss(logits, labels)
        loss.backward()
        total_loss += loss.item()
        optimizer.step()
        scheduler.step()
```

```
        progress_bar.set_postfix({'training_loss':
f'{loss.item():.3f}', 'lr': f'{scheduler.get_last_lr()
[0]:.2e}'})

    avg_loss = total_loss / len(train_loader)  # 计算平均损失
    print(f"Epoch {epoch + 1}/{num_epochs}, Loss:
{avg_loss:.4f}")

    train_losses.append(avg_loss)  # 记录训练损失

    # 验证阶段
    model.eval()
    total_val_loss = 0
    all_val_preds = []
    all_val_labels = []

    with torch.no_grad():
        for batch in val_loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            val_loss = focal_loss(outputs.logits,
batch['labels'])
            total_val_loss += val_loss.item()

            val_preds = torch.argmax(outputs.logits,
dim=1).cpu().numpy()
            val_labels = batch['labels'].cpu().numpy()
            all_val_preds.extend(val_preds)
            all_val_labels.extend(val_labels)

    avg_val_loss = total_val_loss / len(val_loader)
    val_accuracy = accuracy_score(all_val_labels, all_val_preds)
    val_accuracies.append(val_accuracy)
    val_precision = precision_score(all_val_labels,
all_val_preds, average='weighted')
    val_recall = recall_score(all_val_labels, all_val_preds,
average='weighted')
    val_f1 = f1_score(all_val_labels, all_val_preds,
average='weighted')

    val_losses.append(avg_val_loss)

    print(f"Epoch {epoch + 1}/{num_epochs}, Val Loss:
{avg_val_loss:.4f}, Val Accuracy: {val_accuracy:.4f}, "
          f"Val Precision: {val_precision:.4f}, Val Recall:
{val_recall:.4f}, Val F1: {val_f1:.4f}")

    if val_accuracy > best_val_accuracy:
        print(f" best accuracy {val_accuracy:.4f})")
        best_val_accuracy = val_accuracy
```

```
        model.save_pretrained(OUTPUT_DIR,
safe_serialization=False)

model =
BertForSequenceClassification.from_pretrained(OUTPUT_DIR)
model.to(device)
model.eval()  # 切换到评估模式


import matplotlib.pyplot as plt
import numpy as np


def evaluate(model, data_loader):
    all_preds = []
    all_labels = []
    incorrect_samples = []

    with torch.no_grad():
        for batch in tqdm(data_loader, desc="Testing"):
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            preds = torch.argmax(outputs.logits, dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(batch['labels'].cpu().numpy())

            incorrect_mask = (preds !=
batch['labels']).cpu().numpy()
            incorrect_indices = np.where(incorrect_mask)[0]

            # 解码并保存部分错误样本（最多5个）
            for i in incorrect_indices:
                if len(incorrect_samples) >= 5:
                    break
                incorrect_samples.append({
                    'sentence':
tokenizer.decode(batch['input_ids'][i],
skip_special_tokens=True),
                    'true_label': batch['labels'][i].item(),
                    'pred_label': preds[i].item()
                })

    # 计算指标
    accuracy = accuracy_score(all_labels, all_preds)
    precision = precision_score(all_labels, all_preds,
average='weighted')
    recall = recall_score(all_labels, all_preds,
average='weighted')
    f1 = f1_score(all_labels, all_preds, average='weighted')
```

```
    print("随机输出5个错误案例:")
    for case in incorrect_samples[:5]:
        print(f"Sentence: {case['sentence']}")
        print(f"True Label: {case['true_label']}, Predicted 
Label: {case['pred_label']}\n")

    return {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'preds': all_preds,
        'labels': all_labels,
        # 'incorrect_samples' : incorrect_samples
    }


def plot_metrics(train_losses, val_losses, val_accuracies):
    plt.figure(figsize=(12, 5))

    # Loss曲线
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Train Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training & Validation Loss')
    plt.legend()

    # Accuracy曲线
    plt.subplot(1, 2, 2)
    plt.plot(val_accuracies, label='Validation Accuracy', 
color='green')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Validation Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.savefig(f"{PLOTS_DIR}/training_metrics.png")  # 保存图表
    plt.show()


plot_metrics(train_losses, val_losses, val_accuracies)

# 运行测试
test_results = evaluate(model, test_loader)
print(f"""
测试集结果:
准确率: {test_results['accuracy']:.4f}
```

```
精确率: {test_results['precision']:.4f}
召回率: {test_results['recall']:.4f}
F1分数: {test_results['f1']:.4f}
""")
```



测试集结果:
准确率: 0.9398
精确率: 0.9423
召回率: 0.9398
F1分数: 0.9400

附：所有的输出

```
(moshi) (base) wangrui@digital-life:~/shz$ python lab2.py
/home/wangrui/miniconda3/envs/moshi/lib/python3.12/site-
packages/transformers/tokenization_utils_base.py:1601:
FutureWarning: `clean_up_tokenization_spaces` was not set. It
will be set to `True` by default. This behavior will be
depracted in transformers v4.45, and will be then set to `False`
by default. For more details check this issue:
https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
Some weights of BertForSequenceClassification were not
initialized from the model checkpoint at model and are newly
initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be
able to use it for predictions and inference.
改进前的test_data

训练集大小: 67349
列名: ['idx', 'sentence', 'label']
示例:
         idx                                            sentence
label
59113  59113          that the new film is a lame kiddie flick
    0
56034  56034  reflect that its visual imagination is breatht...
     1
35121  35121                 most multilayered and sympathetic
    1

验证集大小: 872
列名: ['idx', 'sentence', 'label']
示例:
     idx                                            sentence
label
601  601                              fancy a real downer ?
0
844  844  given how heavy-handed and portent-heavy it is...
0
349  349   ... turns so unforgivably trite in its last 10...
0

测试集大小: 1821
列名: ['idx', 'sentence', 'label']
示例:
       idx                                            sentence
label
1525  1525  all the well-meaningness in the world ca n't e...
  -1
1392  1392                              go see it and enjoy .
  -1
```

```
867    867  though the controversial korean filmmaker 's l...
  -1

Index(['idx', 'sentence', 'label'], dtype='object')
改进后的test_data

训练集大小: 53879
列名: ['idx', 'sentence', 'label']
示例:
         idx                                           sentence
label
23050  14928  his penchant for tearing up on cue -- things t...
     1
41506  65216                                    expanded vision
    1
5581   27335                           its own languorous charm
    1

验证集大小: 872
列名: ['idx', 'sentence', 'label']
示例:
     idx                                           sentence
label
745  745  made with no discernible craft and monstrously...
0
172  172  it seems like i have been waiting my whole lif...
1
237  237  a by-the-numbers effort that wo n't do much to...
0

测试集大小: 13470
列名: ['idx', 'sentence', 'label']
示例:
         idx                                           sentence
label
12694  29282                         stunningly unoriginal
    0
6969   13624            that embraces its old-fashioned themes
    1
2108   61308  to speak about other than the fact that it is ...
     0

/home/wangrui/miniconda3/envs/moshi/lib/python3.12/site-
packages/transformers/optimization.py:591: FutureWarning: This
implementation of AdamW is deprecated and will be removed in a
future version. Use the PyTorch implementation torch.optim.AdamW
instead, or set `no_deprecation_warning=True` to disable this
warning
  warnings.warn(
```

```
Epoch 1/3: 100%|██████████████████████████████████████|
1684/1684 [02:08<00:00, 13.13it/s, training_loss=0.215,
lr=2.22e-05]
Epoch 1/3, Loss: 0.2543
Epoch 1/3, Val Loss: 0.1994, Val Accuracy: 0.9186, Val
Precision: 0.9187, Val Recall: 0.9186, Val F1: 0.9186
 best accuracy 0.9186)
Epoch 2/3: 100%|██████████████████████████████████████|
1684/1684 [02:08<00:00, 13.15it/s, training_loss=0.013,
lr=1.11e-05]
Epoch 2/3, Loss: 0.1083
Epoch 2/3, Val Loss: 0.2219, Val Accuracy: 0.9255, Val
Precision: 0.9255, Val Recall: 0.9255, Val F1: 0.9255
 best accuracy 0.9255)
Epoch 3/3: 100%|██████████████████████████████████████|
1684/1684 [02:08<00:00, 13.13it/s, training_loss=0.022,
lr=0.00e+00]
Epoch 3/3, Loss: 0.0586
Epoch 3/3, Val Loss: 0.2416, Val Accuracy: 0.9232, Val
Precision: 0.9233, Val Recall: 0.9232, Val F1: 0.9231
Testing: 100%|
████████████████████████████████████████████████████████████████
████████████| 421/421 [00:11<00:00, 37.98it/s]
随机输出5个错误案例:
Sentence: is just the point
True Label: 1, Predicted Label: 0

Sentence: sexy, violent, self - indulgent and maddening
True Label: 0, Predicted Label: 1

Sentence: you'd swear you
True Label: 1, Predicted Label: 0

Sentence: walks a tricky tightrope between being wickedly funny
and just plain wicked
True Label: 0, Predicted Label: 1

Sentence: all but spits out denzel washington's fine performance
in the title role.
True Label: 1, Predicted Label: 0


测试集结果:
准确率: 0.9532
精确率: 0.9534
召回率: 0.9532
F1分数: 0.9532
```

```
/home/wangrui/miniconda3/envs/moshi/lib/python3.12/site-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be depracted in transformers v4.45, and will be then set to `False` by default. For more details check this issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
Some weights of the model checkpoint at model were not used when initializing BertForMaskedLM: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight', 'cls.seq_relationship.bias', 'cls.seq_relationship.weight']
- This IS expected if you are initializing BertForMaskedLM 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 BertForMaskedLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
                                                text
0  This is just a precious little diamond. The pl...
1  When I say this is my favourite film of all ti...
2  I saw this movie because I am a huge fan of th...
3  Being that the only foreign films I usually li...
4  After seeing Point of No Return (a great movie...
Map: 100%|████████████████████████████████████████| 40000/40000 [01:21<00:00, 493.67 examples/s]
Map: 100%|████████████████████████████████████████| 10000/10000 [00:20<00:00, 495.76 examples/s]
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 40000
    })
    test: Dataset({
        features: ['input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 10000
    })
```

```
})
100%|██████████████████████████████████████████████████| 834/834 [08:04<00:00,  1.72it/s, pretraining_loss=2.130, lr=0.00e+00]
预训练完成!模型文件已保存在mlm_results


                                                text
0  I rented I AM CURIOUS-YELLOW from my video sto...
1  "I Am Curious: Yellow" is a risible and preten...
2  If only to avoid making this type of film in t...
3  This film was probably inspired by Godard's Ma...
4  Oh, brother...after hearing about this ridicul...
/home/wangrui/miniconda3/envs/moshi/lib/python3.12/site-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be depracted in transformers v4.45, and will be then set to `False` by default. For more details check this issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at ./mlm_results and are newly initialized: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight', 'classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

Epoch 1
100%|██████████████████████████████████████████████████| 1407/1407 [01:50<00:00, 12.73it/s]
100%|██████████████████████████████████████████████████| 157/157 [00:06<00:00, 23.26it/s]
Train Loss: 0.3248
Val Accuracy: 0.8904, F1: 0.8904

Epoch 2
```

```
100%|
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
█████████████████████████████████████████| 1407/1407
[01:49<00:00, 12.84it/s]
100%|
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
███████████████████████████████████████████| 157/157
[00:06<00:00, 23.28it/s]
Train Loss: 0.1861
Val Accuracy: 0.8932, F1: 0.8932

Epoch 3
100%|
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
█████████████████████████████████████████| 1407/1407
[01:49<00:00, 12.85it/s]
100%|
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
███████████████████████████████████████████| 157/157
[00:06<00:00, 23.56it/s]
Train Loss: 0.0890
Val Accuracy: 0.8888, F1: 0.8888
100%|
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
█████████████████████████████████████████| 1563/1563
[01:06<00:00, 23.44it/s]

Final Test Performance:
Accuracy: 0.8886
F1 Score: 0.8886



/home/wangrui/miniconda3/envs/moshi/lib/python3.12/site-
packages/transformers/tokenization_utils_base.py:1601:
FutureWarning: `clean_up_tokenization_spaces` was not set. It
will be set to `True` by default. This behavior will be
depracted in transformers v4.45, and will be then set to `False`
by default. For more details check this issue:
https://github.com/huggingface/transformers/issues/31884
```

```
  warnings.warn(
Some weights of BertForSequenceClassification were not
initialized from the model checkpoint at model and are newly
initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be
able to use it for predictions and inference.
改进前的test_data

训练集大小: 67349
列名: ['idx', 'sentence', 'label']
示例:
         idx                                              sentence
label
2095    2095                          an appealing couple --
    1
12260  12260  is as uncompromising as it is nonjudgmental , ...
     1
24526  24526  would n't matter so much that this arrogant ri...
     0

验证集大小: 872
列名: ['idx', 'sentence', 'label']
示例:
     idx                                              sentence
label
341  341  it deserves to be seen by anyone with even a p...
1
118  118  every nanosecond of the the new guy reminds yo...
0
832  832  manages to show life in all of its banality wh...
0

测试集大小: 1821
列名: ['idx', 'sentence', 'label']
示例:
       idx                                              sentence
label
1683  1683  the editing is chaotic , the photography grain...
  -1
1704  1704  a fast-moving and remarkable film that appears...
  -1
1708  1708  one of the best examples of how to treat a sub...
  -1

Index(['idx', 'sentence', 'label'], dtype='object')
改进后的test_data

训练集大小: 53879
列名: ['idx', 'sentence', 'label']
示例:
```

```
         idx                                              sentence  
label
45967  41515                                  screen presence   
    1
30634  15405  a film in a class with spike lee 's masterful  ...
     1
29699   3825  fails in making this character understandable  ...
     0

验证集大小: 872
列名: ['idx', 'sentence', 'label']
示例:
     idx                                           sentence  
label
618  618  without non-stop techno or the existential ove...
0
388  388  when leguizamo finally plugged an irritating c...
0
233  233  i 'd have to say the star and director are the...
0

测试集大小: 13470
列名: ['idx', 'sentence', 'label']
示例:
         idx                                              sentence  
label
11605  54044                              flashy editing style   
    1
5952   16152                         ca n't rescue this effort   
    0
6162    2210  is akin to a reader 's digest condensed versio...
     0

/home/wangrui/miniconda3/envs/moshi/lib/python3.12/site-
packages/transformers/optimization.py:591: FutureWarning: This
implementation of AdamW is deprecated and will be removed in a
future version. Use the PyTorch implementation torch.optim.AdamW
instead, or set `no_deprecation_warning=True` to disable this
warning
  warnings.warn(
Epoch 1/3: 100%|
████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████
████████████████████████████████████| 1684/1684 [03:59<00:00,
7.02it/s, training_loss=0.003, lr=2.22e-05]
Epoch 1/3, Loss: 0.0169
Epoch 1/3, Val Loss: 0.0138, Val Accuracy: 0.9312, Val
Precision: 0.9328, Val Recall: 0.9312, Val F1: 0.9312
 best accuracy 0.9312)
```

```
Epoch 2/3: 100%|
| 1684/1684 [04:00<00:00, 7.01it/s, training_loss=0.003, lr=1.11e-05]
Epoch 2/3, Loss: 0.0075
Epoch 2/3, Val Loss: 0.0133, Val Accuracy: 0.9232, Val Precision: 0.9232, Val Recall: 0.9232, Val F1: 0.9232
Epoch 3/3: 100%|
| 1684/1684 [03:59<00:00, 7.03it/s, training_loss=0.002, lr=0.00e+00]
Epoch 3/3, Loss: 0.0036
Epoch 3/3, Val Loss: 0.0180, Val Accuracy: 0.9278, Val Precision: 0.9280, Val Recall: 0.9278, Val F1: 0.9277
Testing: 100%|
| 421/421 [00:18<00:00, 22.34it/s]
随机输出5个错误案例:
Sentence: indie.
True Label: 1, Predicted Label: 0

Sentence: is eerily convincing as this bland blank of a man with unimaginable demons within
True Label: 1, Predicted Label: 0

Sentence: sexy, violent, self - indulgent and maddening
True Label: 0, Predicted Label: 1

Sentence: ' s weird, wonderful, and not necessarily for kids
True Label: 1, Predicted Label: 0

Sentence: you'd swear you
True Label: 1, Predicted Label: 0


测试集结果:
准确率: 0.9398
精确率: 0.9423
召回率: 0.9398
F1分数: 0.9400
```