上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

2025《人工智能导论》大作业

NIS4307 Rumor Detector

任务名称:	Rumor Detector
完成组号:	第 1 组
小组成员:	马悦钊　李卓恒
	刘梓芃　聂鸣涛
完成时间:	2025 年 6 月 5 日

目 录

第 1 章　任务目标

本次课程设计的任务是基于谣言检测数据集，构建一个检测模型。该模型可以对数据集中的推文进行谣言检测与识别。任务目标如下：

- 数据集：使用给定的谣言检测数据集，数据集包含推文文本和标签（谣言或非谣言）。
- 模型训练：使用逻辑回归或 GRU 等深度学习模型进行谣言检测，实现二分类任务，用 0 代表非谣言、1 代表谣言。
- 模型评估：使用测试集对模型进行评估，计算分类准确率、精确率、召回率和 F1 分数等指标。
- 模型接口：实现一个接口类，提供谣言检测功能，对外提供一个接口函数，接收一条字符串作为输入，输出一个整数值作为对应的预测类别。
- 泛化能力：模型应具有较好的泛化能力，能够适应不同类型的谣言检测任务。
- 结果可视化：对模型训练结果进行可视化展示。

第 2 章　具体内容

2.1　实施方案

在经过小组成员的讨论后，我们决定采用 AdvancedBiLSTM3（改进型双向长短时记忆网络）模型开展谣言检测任务。该模型在传统 BiLSTM 基础上，结合了注意力机制、Dropout 正则化和 LayerNorm 等技术，能够更好地捕捉文本的上下文依赖关系，并有效缓解过拟合问题。具体方案如下：

首先，我们对数据进行预处理：

- **构建数据集**：除了老师给定的训练集和测试集外，我们还从 PHEME Dataset 额外获取到更多的推特谣言数据集，经过分类处理后按照 8：1：1 的比例划分为训练集、验证集和测试集。训练集包含约 5500 条推文，验证集和测试集各包含约 600 条推文。
- **文本清洗**：使用正则表达式去除 URL 和 @ 提及，统一转换为小写，并通过 NLTK 进行分词和词干提取，降低词形变化对模型的干扰。
- **词表构建**：基于训练集文本构建词表，过滤低频词（最小频率 2），并添加`<PAD>`（填充符）和`<UNK>`（未知词）标记。
- **序列编码**：将文本转换为固定长度的数字序列（`MAX_LEN=64`），不足长度的文本使用`<PAD>`进行填充，超长部分截断（采用前向截断），确保输入数据的一致性。

在模型架构上，我们采用的 AdvancedBiLSTM3 模型包含以下关键模块：

- **嵌入层**：使用可训练的词向量，维度为`EMBEDDING_DIM=128`，并在嵌入层后添加`Dropout`以缓解过拟合。
- **双向 LSTM 层**：使用双向 LSTM，设置`bidirectional=True`，并设置隐藏层维度为`HIDDEN_DIM=256`以捕捉上下文语义。此外还采用 2 层堆叠结构，并在层间使用`Dropout`。
- **注意力机制**：引入多层注意力机制，通过`Tanh`激活函数和 128 维的中间层增强特征表达，使用掩码处理填充值，避免无效位置参与注意力计算。
- **分类器**：包含多层全连接网络，使用`ReLU`激活函数和`LayerNorm`归一化，最终通过`sigmoid`函数输出二分类结果，即以 0 表示非谣言，1 表示谣言。

与传统的 BiGRU 或逻辑回归模型相比，AdvancedBiLSTM3 模型具备更强的特征表

达能力和鲁棒性，能够自动学习文本中的复杂语义关系，适应不同领域的谣言检测任务。通过注意力机制，模型能够聚焦于文本中的关键信息片段，提升对谣言特征的捕捉能力。在对比不同策略的训练成果后，我们发现该模型在准确率和泛化能力方面均优于传统方法，具体的实验结果将在后续章节中详细展示。

2.2　核心代码分析

2.2.1　模型定义 model.py

```python
import torch
import torch.nn as nn
class AdvancedBiLSTM3(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, num_layers=2, dropout=0.5):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        # 嵌入层 Dropout 增强鲁棒性
        self.embedding_dropout = nn.Dropout(dropout + 0.1)
        self.lstm = nn.LSTM(
            embedding_dim,
            hidden_dim // 2,
            num_layers=min(num_layers, 2),
            bidirectional=True,
            batch_first=True,
            dropout=dropout if num_layers > 1 else 0
        )
        # 带中间层的注意力机制
        self.attention = nn.Sequential(
            nn.Linear(hidden_dim, 128),
            nn.Tanh(),
            nn.Dropout(dropout),
            nn.Linear(128, 1)
        )
        # 带正则化的分类器
        self.classifier = nn.Sequential(
            nn.Dropout(dropout + 0.1),
            nn.Linear(hidden_dim, 64),
            nn.ReLU(),
            nn.LayerNorm(64),
            nn.Dropout(dropout),
            nn.Linear(64, 1)
        )
```

在模型定义中，我们通过hidden_dim//2设置双向 LSTM 的隐藏层维度，确保模型能够捕捉到文本的双向上下文信息。此外我们还引入带中间层的注意力机制，通过Tanh激活函数和 128 维的中间层增强特征表达，使用掩码处理填充值，避免无效位置参与注意力计算。分类器则包含多层全连接网络，使用ReLU激活函数和LayerNorm归一化，最终通过sigmoid函数输出二分类结果。

2.2.2　训练模型 train_lstm.py

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import pandas as pd
from collections import Counter
import re
import joblib
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, recall_score,
    f1_score, confusion_matrix, classification_report
import numpy as np
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from model import AdvancedBiLSTM3 as AdvancedBiLSTM
import nltk
try:
    nltk.data.find('tokenizers/punkt_tab')
except LookupError:
    nltk.download('punkt_tab')

# 超参数设置
BATCH_SIZE = 32          # 批大小
EMBEDDING_DIM = 128      # 嵌入维度 (可修改)
HIDDEN_DIM = 256         # 隐藏层维度 (可修改)
EPOCHS = 30              # 训练轮数 (可修改)
MAX_LEN = 64             # 文本最大长度
LEARNING_RATE = 0.9e-2   # 学习率 (可修改)
FACTOR = 0.9             # 学习率衰减因子
```

```
WEIGHT_DECAY = 1e-4      # L2 正则化
DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# 路径设置
model_parameter = f'{EMBEDDING_DIM}_{HIDDEN_DIM}_{EPOCHS}_{LEARNING_RATE}'
model_path = f'../Output/Model/{model_parameter}.pt'
vocab_path = f'../Output/Model/vocab_{model_parameter}.pkl'
train_path = '../Dataset/split/train.csv'
ex_train_path = '../Dataset/split/ex_train.csv'
val_path = '../dataset/split/val.csv'
ex_val_path = '../dataset/split/ex_val.csv'
test_path = '../dataset/test/test.csv'
graph_path = f'../Output/Graph/{model_parameter}.png'

def tokenize(text):
    """对文本进行分词和预处理"""
    # 处理 URL 和 @ 提及
    text = re.sub(r'http\S+', '<URL>', text)
    text = re.sub(r'@\w+', '@USER', text)
    # 使用 NLTK 分词 + 词干提取
    tokens = word_tokenize(text)
    stemmer = PorterStemmer()
    return [stemmer.stem(w.lower()) for w in tokens if w.isalpha()]

def build_vocab(texts, min_freq=2):
    """构建词表，与示例代码相同"""
    pass

def encode(text, vocab):
    """将文本编码为数字序列，与示例代码相同"""
    pass

class RumorDataset(Dataset):
    """自定义数据集类，与示例代码相同"""
    pass
```

```
64
65  def evaluate(model, loader):
66      """评估函数，调用外部库计算返回准确率、精确率、召回率和F1-score"""
67      pass
68
69  def plot_learning_curve(train_metrics, val_metrics, epochs,
        save_path):
70      """绘制包含损失率和核心指标的双图学习曲线"""
71      pass
72
73  def main():
74      # 读取数据集
75      print("正在加载数据...")
76      train_df = pd.read_csv(train_path)
77      ex_train_df = pd.read_csv(ex_train_path)  # 读取新增训练集
78      print(f"ex训练集大小: {len(ex_train_df)}")
79      print(f"原始训练集大小: {len(train_df)}")
80      train_df = pd.concat([train_df,ex_train_df], ignore_index=True
        )
81      print(f"合并后的训练集大小: {len(train_df)}")
82      val_df = pd.read_csv(val_path)
83      ex_val_df = pd.read_csv(ex_val_path)
84      print(f"验证集大小: {len(val_df)}")
85      print(f"新增验证集大小: {len(ex_val_df)}")
86      val_df = pd.concat([val_df,ex_val_df], ignore_index=True)
87      print(f"合并后的验证集大小: {len(val_df)}")
88
89      # 构建词表
90      print("正在构建词表...")
91      vocab = build_vocab(train_df['text'])
92      joblib.dump(vocab, vocab_path)  # 保存词表
93
94      # 构建数据集
95      train_set = RumorDataset(train_df, vocab)
96      val_set = RumorDataset(val_df, vocab)
97
98      train_loader = DataLoader(train_set, batch_size=BATCH_SIZE,
        shuffle=True)
```

```
    val_loader = DataLoader(val_set, batch_size=BATCH_SIZE)

    # 初始化模型、优化器和损失函数
    print("正在初始化模型...")
    model = AdvancedBiLSTM(len(vocab), EMBEDDING_DIM, HIDDEN_DIM).to(DEVICE)
    optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='max', factor=FACTOR, patience=3, verbose=True
    )
    criterion = nn.BCEWithLogitsLoss()

    # 记录训练过程指标
    train_history = {
        'loss': [], 'accuracy': [], 'precision': [], 'recall': [], 'f1': []
    }
    val_history = {
        'loss': [], 'accuracy': [], 'precision': [], 'recall': [], 'f1': []
    }
    print("开始训练模型...")
    # 训练模型
    best_val_f1 = 0.0
    for epoch in range(EPOCHS):
        model.train()
        epoch_loss = 0
        # 训练一个 epoch
        for x, y in train_loader:
            x, y = x.to(DEVICE), y.to(DEVICE)
            logits = model(x)
            loss = criterion(logits, y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
        avg_loss = epoch_loss / len(train_loader)

```

```
134         # 计算训练集指标
135         train_acc, train_prec, train_rec, train_f1, _ = evaluate(
    model, train_loader)
136         train_history['accuracy'].append(train_acc)
137         train_history['precision'].append(train_prec)
138         train_history['recall'].append(train_rec)
139         train_history['f1'].append(train_f1)
140         train_history['loss'].append(avg_loss)
141 
142         val_acc, val_prec, val_rec, val_f1, val_loss = evaluate(
    model, val_loader)
143         val_history['accuracy'].append(val_acc)
144         val_history['precision'].append(val_prec)
145         val_history['recall'].append(val_rec)
146         val_history['f1'].append(val_f1)
147         val_history['loss'].append(val_loss)
148 
149         scheduler.step(val_f1)  # 根据验证集 F1 分数调整学习率
150 
151         print(f'Epoch {epoch+1}/{EPOCHS}')
152         # 保存最佳模型
153         if val_f1 > best_val_f1:
154             best_val_f1 = val_f1
155             torch.save(model.state_dict(), model_path)
156             print(f'Saved best (Train F1: {val_f1:.4f})')
157         print(f'Train: Loss={avg_loss:.4f}, Acc={train_acc:.4f},
    Prec={train_prec:.4f}, Rec={train_rec:.4f}, F1={train_f1:.4f}')
158         print(f'Val: Loss={val_loss:.4f}, Acc={val_acc:.4f}, Prec
    ={val_prec:.4f}, Rec={val_rec:.4f}, F1={val_f1:.4f}')
159         print('-' * 60)
160     print("\n 训练完成!")
161     plot_learning_curve(train_history, val_history, EPOCHS,
    graph_path)
162 
163 if __name__ == '__main__':
164     main()
```

在训练模型的代码中，我们首先定义了超参数和路径设置，接着使用tokenize函数对文本进行分词和预处理，build_vocab函数构建词表，并将其保存到指定路径。然

后，我们定义了自定义数据集类RumorDataset，用于加载和处理数据。接下来，我们初始化模型、优化器和损失函数，并记录训练过程中的指标。通过循环迭代训练数据，计算损失并更新模型参数，同时在每一轮训练结束后评估模型在验证集上的性能。

2.2.3　接口类定义 classify.py

```
import torch
import joblib
from model import AdvancedBiLSTM3 as AdvancedBiLSTM
from train_lstm import *

class RumourDetectClass:
    def __init__(self, model_path, vocab_path, embedding_dim=EMBEDDING_DIM, hidden_dim=HIDDEN_DIM, device=DEVICE):
        # 加载词表和模型参数
        self.vocab = joblib.load(vocab_path)
        self.model = AdvancedBiLSTM(len(self.vocab), embedding_dim, hidden_dim).to(device)
        self.model.load_state_dict(torch.load(model_path, map_location=device))
        self.model.eval()

    @classmethod
    def construct_detector(cls):
        embedding_dim = EMBEDDING_DIM
        hidden_dim = HIDDEN_DIM
        epochs = EPOCHS
        learning_rate = LEARNING_RATE
        device = DEVICE
        model_path = f'../Output/Model/best_{embedding_dim}_{hidden_dim}_{epochs}_{learning_rate}.pt'
        vocab_path = f'../Output/Model/vocab_{embedding_dim}_{hidden_dim}_{epochs}_{learning_rate}.pkl'
        return cls(model_path, vocab_path, embedding_dim, hidden_dim, device)


```

```
26      def classify(self, text: str) -> int:
27          # 预测流程
28          ids = encode(text, self.vocab)
29          x = torch.tensor([ids], dtype=torch.long).to(DEVICE)
30          with torch.no_grad():
31              logits = self.model(x)
32              pred = (torch.sigmoid(logits) > 0.5).float().item()
33          return int(pred)
```

在接口类中，我们实现了RumourDetectClass接口类，该类提供了两个初始化方法：__init__和construct_detector，分别可以进行指定或默认模型和词表的初始化谣言检测器。此外还实现了一个接口函数classify，接收一条字符串作为输入，输出一个整数值作为对应的预测类别。通过construct_detector方法，我们可以快速构建谣言检测器，加载默认模型和词表。

本项目代码发布在 GitSJTU 上，包含了数据预处理、模型训练、模型评估和接口类的完整实现。

2.3　测试结果分析

在确定模型架构后，我们经过不断调整训练时使用的参数，最终选定了以下配置：

- 嵌入维度：`EMBEDDING_DIM = 128`
- 隐藏层维度：`HIDDEN_DIM = 256`
- 训练轮数：`EPOCHS = 30`
- 学习率：`LEARNING_RATE = 0.009`
- 批大小：`BATCH_SIZE = 32`
- 最大文本长度：`MAX_LEN = 64`
- 学习率衰减因子：`FACTOR = 0.9`
- L2 正则化：`WEIGHT_DECAY = 1e-4`

在经过 30 轮训练后，我们得到了如下图2.1所示的模型训练指标，30 轮训练总耗时约 5 分钟，最佳模型在第 21 轮产生，其验证集 F1 分数达到了 0.8592。可以看到，随着训练轮数的增加，训练集和验证集的损失率逐渐波动降低，验证集的准确率、精确率、召回率和 F1 分数也在不断波动提升，表明模型在谣言检测任务上取得了较好的效果。

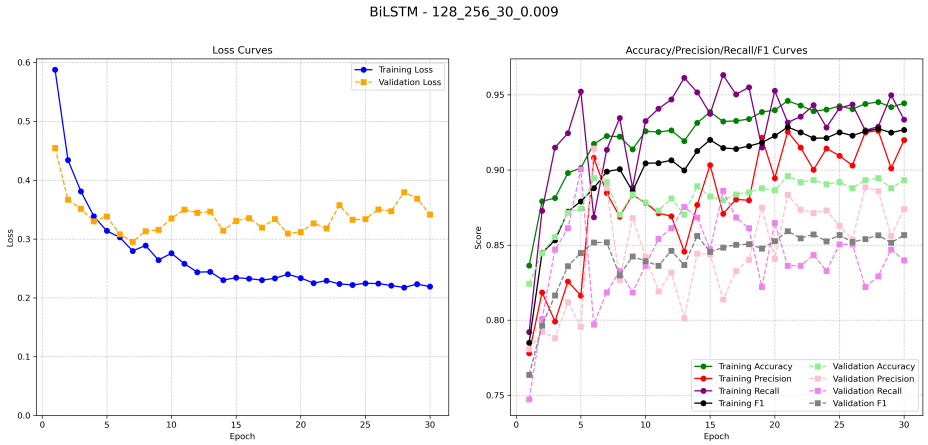

图 2.1　模型训练过程中的损失率和核心指标的双图学习曲线

使用 test.py 脚本对测试集进行评估，我们得到了如下表2.1所示的测试结果。结果表明，此模型在我们的测试集上表现良好，其中预测准确率达到了 86.64%。

预测 \ 真实	非谣言（0）	谣　言（1）
非谣言（0）	303	44
谣　言（1）	28	164

表 2.1　测试结果

第 3 章　工作总结

3.1　收获与心得

通过本次课程设计，我们深入学习了深度学习模型在自然语言处理中的应用，特别是双向门控循环单元（BiGRU）和改进型双向长短时记忆网络（AdvancedBiLSTM）模型在谣言检测任务中的应用，也深刻体会到了理解 BiLSTM 结合注意力机制在序列分类中的优势。通过对比不同模型的性能，我们认识到模型的选择对任务结果的影响。此外，我们实现了从数据预处理、模型训练到模型评估的全流程开发，初步掌握了如何使用 PyTorch 等深度学习框架进行自然语言处理任务。

3.2　遇到问题及解决思路

在项目实施过程中，我们也遇到了一些问题，例如数据集不平衡导致模型偏向于某一类标签、训练轮数过多导致出现过拟合现象、以及模型参数设置不当导致训练效果不佳等。针对这些问题，我们采取以下解决思路：

- **数据集不平衡**：我们通过自行搜索 PHEME Dataset 等公开数据集，增加了训练集的样本量，并对数据进行分层随机抽样，确保各类标签的样本数量相对均衡，极大丰富了数据集内容，从而获得更全面的训练样本。
- **过拟合问题**：针对较多训练轮数中验证集损失曲线先降后升的过拟合问题，我们引入了 Dropout 正则化技术，并通过交叉验证选择最佳模型参数，避免模型在训练集上过拟合。
- **模型参数设置**：在实验过程中，我们不断调整模型参数，如学习率、嵌入维度、隐藏层维度等，尝试不同的网络结构和超参数组合。通过对比不同配置下的模型性能，我们最终找到了在有限时间内训练出较为稳定模型的佳配置。

通过这些问题的解决，我们不仅提升了模型的性能，也加深了对深度学习模型在自然语言处理任务中应用的理解。

第 4 章　课程建议

本次课程设计通过实践操作，让我们对深度学习与自然语言处理的结合应用有了具象认知，但在课程学习及实践过程中，也发现一些可以优化改进的方向，此处提出几点建议，希望能为后续课程设计提供参考：

当前课程较多聚焦于基础概念的介绍讲解，但对具体算法的原理推导与代码实现讲解较少，部分概念比较晦涩难懂但缺乏深入剖析，导致学生在理解上存在困难。希望老师能结合代码实例进行拆解演示，如结合 PyTorch 等库的具体实现，深入讲解 GRU、LSTM 等模型的工作原理与数学推导，帮助学生更好地理解模型背后的逻辑。

此外，本课程前期未铺垫相关实践案例，而课程设计在学期末才公布，与其他课程结课任务、考试复习等时间冲突，导致学生难以分配足够精力深入探索，也是我认为可以改进的地方。在本次课程设计中，我们小组成员普遍感受到时间紧迫，从理解任务、数据预处理到模型调优全程压缩在短时间内，尤其是在数据预处理、模型训练与调优等环节，难以进行充分的实验与探索。建议将课程设计主题提前半学期公布，分阶段设置任务节点（如第 8 周完成数据预处理、第 12 周提交模型初版等），并配套阶段性指导，帮助学生合理规划时间，确保实践质量。

而在完成项目的过程中，我们也意识到仅凭课堂上学到的知识，难以独立完成整个项目，特别是对 PyTorch 、Sklearn 等库的使用不够熟悉，导致在实现过程中遇到很多问题。建议老师能在前期的课程中结合具体案例，帮助学生更好地掌握这些工具的使用方法。