

**课 程 实 验 报 告**

**课程名称： 自然语言处理实验**

**专业班级： 计算机科学与技术**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期：**

**计算机科学与技术学院**

目 录

[1 中文分词实现 2](#_Toc197962248)

[1.1 问题描述 2](#_Toc197962249)

[中文分词是自然语言处理中的一个重要任务，其目的是将连续的汉字序列切分成有意义的词语序列。由于中文没有像英文那样的自然分词符号（如空格），因此需要借助特定的算法来完成分词任务。本实验旨在通过实现和对比不同的分词算法，理解各种算法的原理、优缺点以及性能差异。 2](#_Toc197962250)

[1.2 基础模块 2](#_Toc197962251)

[1.3 系统实现 3](#_Toc197962252)

[1.4 实验小结 6](#_Toc197962253)

[2 中文命名实体识别实现 7](#_Toc197962254)

[2.1 问题描述 7](#_Toc197962255)

[2.2 基础模块 7](#_Toc197962256)

[2.3 系统实现 8](#_Toc197962257)

[2.4 实验小结 10](#_Toc197962258)

[参考文献 12](#_Toc197962259)

[附录A 中文分词实现的源程序 13](#_Toc197962260)

[附录B 命名实体识别实现的源程序 16](#_Toc197962261)

# 1 中文分词实现

## 1.1 问题描述

## 中文分词是自然语言处理中的一个重要任务，其目的是将连续的汉字序列切分成有意义的词语序列。由于中文没有像英文那样的自然分词符号（如空格），因此需要借助特定的算法来完成分词任务。本实验旨在通过实现和对比不同的分词算法，理解各种算法的原理、优缺点以及性能差异。

## 1.2 基础模块

中文分词指将汉字序列切分成单个词语的过程。在以英文为代表的拉丁语系中，单词之间是以空格作为自然分界符的。而中文只有字、句和段能通过明显的分界符来划界，唯独词没有一个形式上的分界符，这给中文的处理带来了独有的困难，因此中文分词技术得到了广泛研究。

理论上讲，构建一套完备的分词规则便可以将所有句子正确划分，但语言规则庞大复杂并且是动态发展的，编写这样一套规则是不现实的，因此目前主流的分词方法可以大致分为两种。

**1.2.1基于词典匹配的分词算法**

正向最大匹配法（FMM）：从左向右扫描文本，尝试匹配最长的词。

逆向最大匹配法（RMM）：从右向左扫描文本，尝试匹配最长的词。

双向最大匹配法（BIMM）：结合FMM和RMM的结果，选择更优的分词结果。

**1.2.2基于统计学习的分词算法**

将中文分词视作一个序列标注任务，给句子中的每个字打上合适的标签（如

BIES标注）。

使用的“BIES”标注方式：

B：代表该字是一个词的开头。

I：代表该字在一个词的内部。

E：代表该字是一个词的结尾。

S：代表该字自己就是一个词。

使用Bi-LSTM+CRF等深度学习模型进行分词，能够自动学习复杂的语言特征。

## 系统实现

**1.3.1基于词典的分词器**

使用正向最大匹配、逆向最大匹配与双向最大匹配三种策略。

使用提供的 Dictionary\_based 文件夹中词典，通过以下三种匹配方法实现：

正向最大匹配：从左至右取最大长度词查词典，匹配成功则输出。

逆向最大匹配：从右至左取最大长度词查词典。

双向最大匹配：按照规则（词数少优先，单字少优先）选择最终结果。

**核心代码：**

|  |  |
| --- | --- |
| 图1- 1 正向最大匹配实现 | 图1- 2 反向最大匹配实现 |

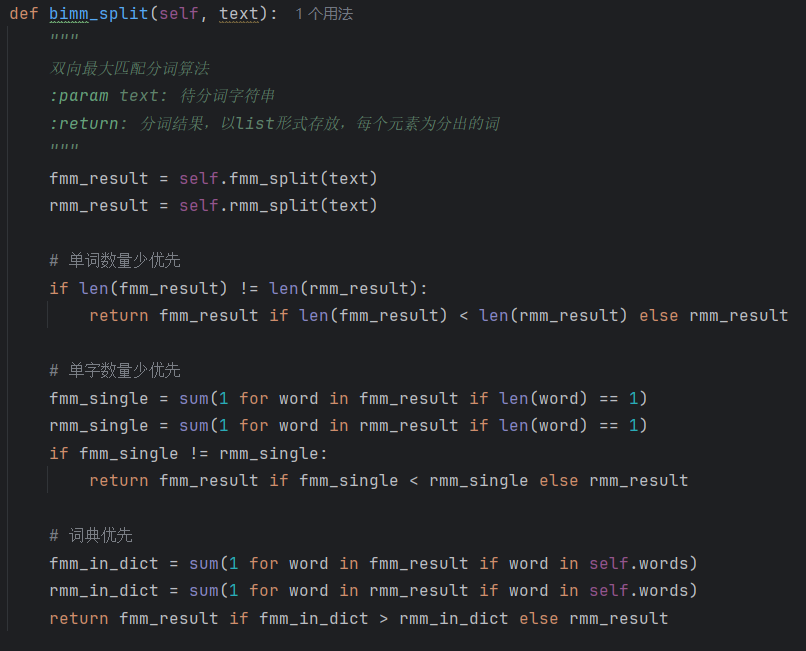


图1- 3 双向最大匹配实现

**测试结果**：

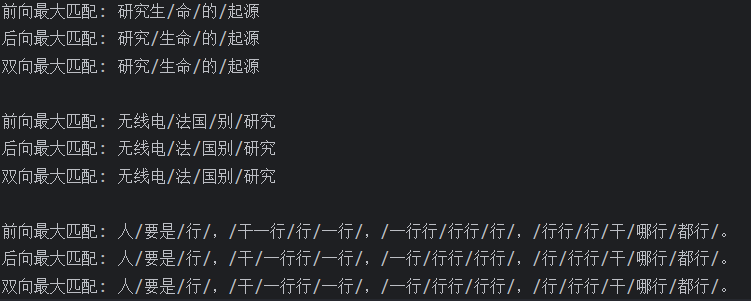


图1- 4 基于词典分词的测试结果

**1.3.2基于统计学习的分词器（Bi-LSTM+CRF）**

将分词问题建模为序列标注问题，采用BIES标签表示词边界。

使用资料包中的 Bi-LSTM+CRF 代码（PyTorch实现）：

采用BIES标签；

训练数据按 8:2 划分训练集与验证集；

网络结构为嵌入层 → 双向LSTM → CRF；

输出为每个字的标签序列，通过标签解码为词序列。



图1- 5 实验流程

**模型训练流程**：

加载数据：使用预处理好的 datasave.pkl 文件，内含 word2id, tag2id, 训练集、测试集等；

构造模型与优化器：模型由 CWS 类定义，使用 Adam 优化器；

训练过程：遍历训练集，执行前向传播、损失计算、反向传播和参数更新；

测试过程：每轮训练结束后在测试集上进行预测；使用 entity\_split 函数按 B/M/E/S 标注恢复词边界；计算分词的 Precision / Recall / F1-score。

保存模型：每个 epoch 结束后保存当前模型至 save/model\_epochX.pkl。

**日志记录**：

使用 logging 模块将每步损失与最终评估指标写入 save/log.txt 文件，同时在终端输出。

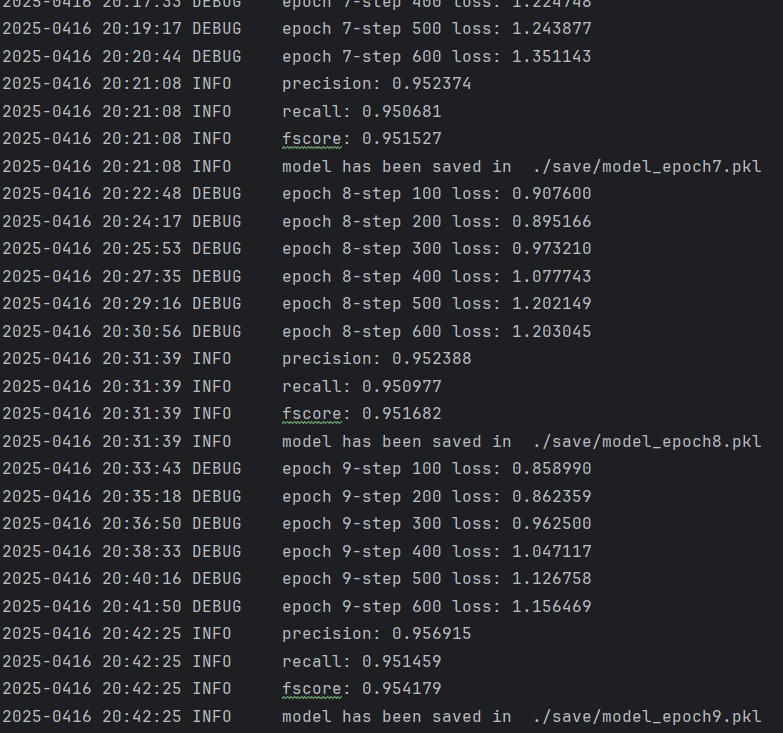


图1- 6 部分log日志

**推理结果**：



图1- 7 infer推理结果

## 1.4 实验小结

本次实验围绕中文分词任务，分别完成了基于词典的分词方法与基于深度学习的 BiLSTM-CRF 分词模型的实现与对比，对中文分词技术有了较为全面的认识。

在第一阶段，我们实现了基于正向最大匹配、逆向最大匹配和双向最大匹配的词典分词算法。该方法思路简单、实现高效，适合分词规则明确、词典覆盖全面的场景，但在处理未登录词（OOV）或歧义词语时存在明显局限。

在第二阶段，我们基于 PyTorch 实现了 BiLSTM-CRF 模型，并利用标注数据进行训练和评估。该模型通过 BiLSTM 获取上下文特征，再借助 CRF 捕捉标签之间的依赖关系，显著提升了分词准确率。尤其在处理未登录词、新词、语义歧义等复杂情况时，模型展现出较强的泛化能力。通过 precision、recall 与 F1-score 等指标的评估，验证了该方法在实际应用中的有效性。

综合来看，字典分词适合资源受限、响应速度要求高的应用场景，而 BiLSTM-CRF 模型分词则更适合精度要求高、语境复杂的任务。通过本次实验，我深入理解了两类方法的优缺点，并掌握了从数据处理、模型设计到训练评估的完整流程，为后续中文自然语言处理的学习与研究奠定了基础。

# 2 中文命名实体识别实现

## 2.1 问题描述

中文命名实体识别（Chinese Named Entity Recognition，CNER）是指从连续的中文文本中识别出具有特定意义的实体，例如人名、地名、组织机构名等。与英文相比，中文缺乏显式的词界限，命名实体边界难以直接定位，增加了识别难度。

本实验的目标是使用深度学习方法完成中文命名实体识别任务，具体采用 BiLSTM-CRF 模型，从标注语料中学习实体边界与类别。实验数据使用 BMES 标签体系（B=Begin，M=Middle，E=End，S=Single），将每个字打上标签，进而识别出命名实体。

## 2.2 基础模块

本实验项目目录结构清晰，主要分为以下几个模块：

1. 数据预处理

RMRB\_NER\_CORPUS.txt：原始人民日报标注语料。

0.split.py：将原始语料划分为训练集、验证集、测试集。

1.data\_u\_ner.py：构造字典、标签集合，生成标准输入输出格式，并序列化为 ner\_datasave.pkl。

2. 模型构建与训练

model.py：定义 BiLSTM-CRF 模型结构。

dataloader.py：定义 Sentence 类，负责将数据封装为 DataLoader 支持的形式。

run.py：主程序入口，负责模型训练与保存。

infer.py：预测与测试模块，支持加载模型并输出预测结果。

3. 其他

requirements.txt：所需依赖包列表。

README.md：项目说明文档。

## 2.3 系统实现

本实验目标是在人民日报标注语料上，使用 BiLSTM-CRF 模型进行中文命名实体识别，实现端到端的实体识别系统。



图2- 1 实验流程

**数据预处理**

原始数据采用人民日报标注语料 RMRB\_NER\_CORPUS.txt，其中每行包含一个字及其对应的实体标签。预处理流程如下：

1.数据划分（0.split.py）：按比例将数据划分为训练集、验证集、测试集，分别保存在 ner\_train.txt、ner\_valid.txt、ner\_test.txt。

2.数据编码（1.data\_u\_ner.py）：构建字典 word2id 和标签 tag2id；将每个样本转为数字序列；

最后使用 pickle 将词表、标签表和所有数据集打包保存为 ner\_datasave.pkl。

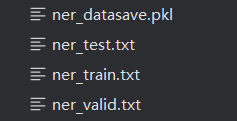


图2- 2 数据划分与打包

**模型结构**

输入层：对字进行 one-hot 编码并嵌入到 embedding\_dim 维空间中

编码层：使用双向 LSTM（BiLSTM）对序列进行上下文建模，输出隐藏状态

解码层：利用条件随机场（CRF）建模标签之间的依赖关系，提升序列标注的合理性

**训练流程与实体评估**

从 ner\_datasave.pkl 中加载训练集与测试集数据

构造 DataLoader 进行批量训练

使用 Adam 优化器更新模型参数

每训练一轮保存一次模型 model\_epochX.pkl

使用 BMES 标签体系，将标签序列转换为实体区间（entity\_split 函数）

对测试结果进行精度、召回率与 F1 值评估

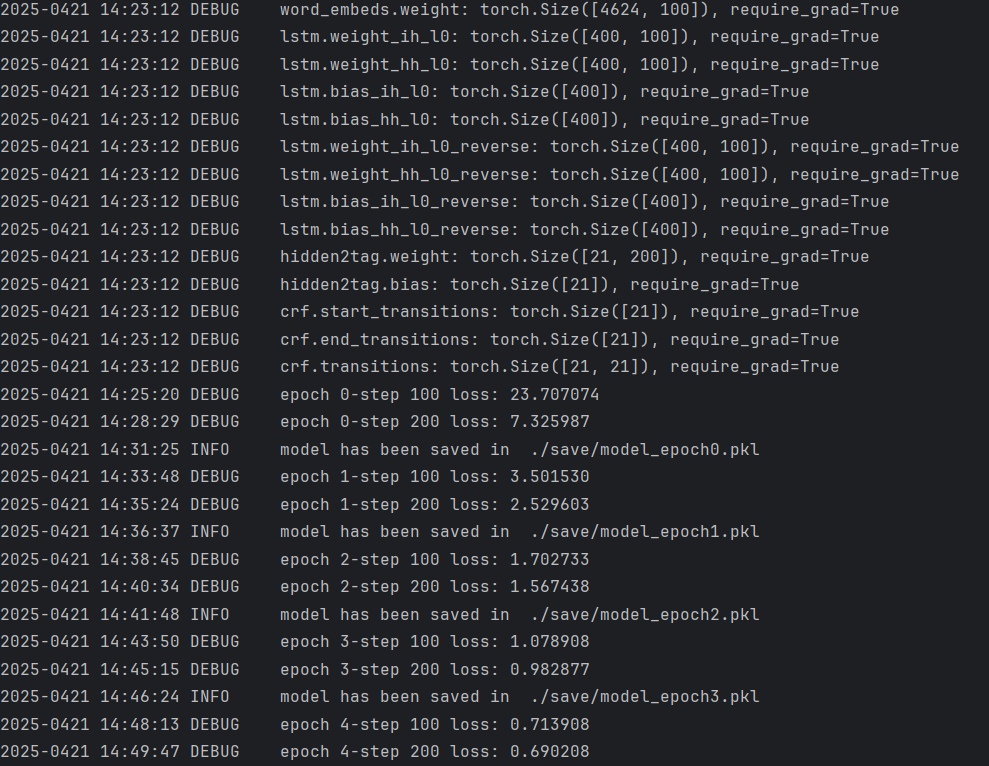


图2- 3 部分训练日志

**实体预测与输出**

训练完成后可通过模型进行预测，流程如下：

模型加载：从 save/model\_epoch.pkl 中读取训练完成的模型，默认加载到 CPU；

测试集读取：从 data/ner\_test.txt 中逐行读取测试样本，每个样本为一个汉字与标签；

构造输入张量：将测试序列转为字索引并创建掩码张量 mask；

调用模型推理函数：通过 model.infer(x, mask, length) 获取预测的标签索引序列；

结果写入文件：将每个字及其预测标签（如 B-M, E-M 等）写入输出文件 ner\_result.txt，每个样本间空行分隔。

该步骤实现了模型推理与预测结果输出，便于后续人工或脚本进行准确率、召回率等性能评估。

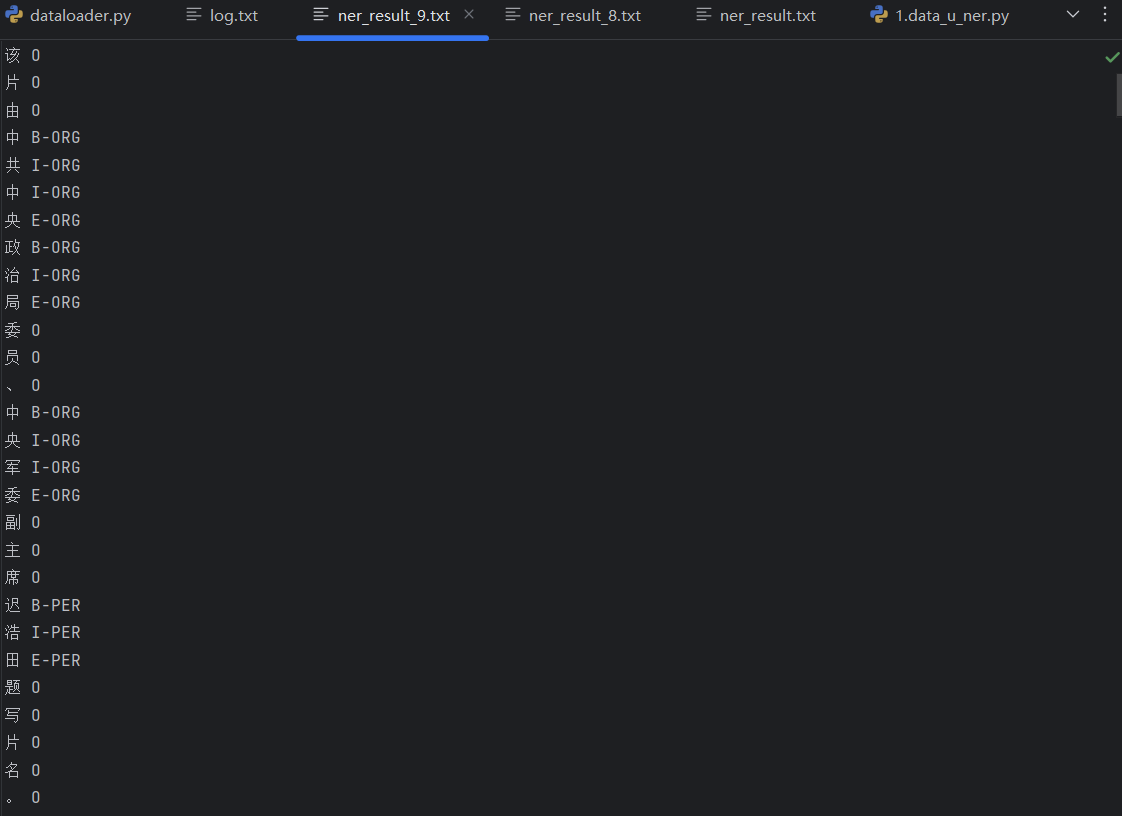


图2- 4 识别结果

## 2.4 实验小结

本实验基于 BiLSTM-CRF 实现了一个完整的中文命名实体识别系统。与词典分词相比，该模型具备更强的上下文理解能力与泛化能力，能够识别出未登录词并准确判断实体边界。

我掌握了数据预处理流程，能独立构造模型所需格式，深入理解 BiLSTM 与 CRF 模块的集成方式。还学会使用 PyTorch 框架进行模型训练、调参与评估

初步实现了实体识别与预测逻辑。

可改进：

未使用验证集进行 early stopping 或超参调整

未对不同类别实体（PER/LOC/ORG）进行区分，尚处于通用实体识别阶段

模型尚未融合预训练语言模型（如 BERT），后续可进一步提升性能

综上，本实验完成了从数据构建到模型训练及测试的完整流程，打下了坚实的中文 NER 系统实现基础。

# 参考文献

[1] 郑捷著. NLP汉语自然语言处理---原理与实践. 电子工业出版社

# 附录A 中文分词实现的源程序

import torch

import torch.nn as nn

from torchcrf import CRF

from torch.nn.utils.rnn import pack\_padded\_sequence, pad\_packed\_sequence

class Tokenizer(object):  
 def \_\_init\_\_(self, words, max\_len):  
 self.words = words  
 self.max\_len = max\_len  
  
 def fmm\_split(self, text):  
 *"""  
 正向最大匹配分词算法  
 :param text: 待分词字符串  
 :return: 分词结果，以list形式存放，每个元素为分出的词  
 """* result = []  
 index = 0  
 while index < len(text):  
 matched = False  
 for i in range(self.max\_len, 0, -1):  
 if index + i > len(text):  
 continue  
 word = text[index:index + i]  
 if word in self.words:  
 result.append(word)  
 index += i  
 matched = True  
 break  
 if not matched:  
 result.append(text[index])  
 index += 1  
 return result  
  
 def rmm\_split(self, text):  
 *"""  
 逆向最大匹配分词算法  
 :param text: 待分词字符串  
 :return: 分词结果，以list形式存放，每个元素为分出的词  
 """* result = []  
 index = len(text)  
 while index > 0:  
 matched = False  
 for i in range(self.max\_len, 0, -1):  
 if index - i < 0:  
 continue  
 word = text[index - i:index]  
 if word in self.words:  
 result.insert(0, word)  
 index -= i  
 matched = True  
 break  
 if not matched:  
 result.insert(0, text[index - 1])  
 index -= 1  
 return result  
  
 def bimm\_split(self, text):  
 *"""  
 双向最大匹配分词算法  
 :param text: 待分词字符串  
 :return: 分词结果，以list形式存放，每个元素为分出的词  
 """* fmm\_result = self.fmm\_split(text)  
 rmm\_result = self.rmm\_split(text)  
  
 # 单词数量少优先  
 if len(fmm\_result) != len(rmm\_result):  
 return fmm\_result if len(fmm\_result) < len(rmm\_result) else rmm\_result  
  
 # 单字数量少优先  
 fmm\_single = sum(1 for word in fmm\_result if len(word) == 1)  
 rmm\_single = sum(1 for word in rmm\_result if len(word) == 1)  
 if fmm\_single != rmm\_single:  
 return fmm\_result if fmm\_single < rmm\_single else rmm\_result  
  
 # 词典优先  
 fmm\_in\_dict = sum(1 for word in fmm\_result if word in self.words)  
 rmm\_in\_dict = sum(1 for word in rmm\_result if word in self.words)  
 return fmm\_result if fmm\_in\_dict > rmm\_in\_dict else rmm\_result  
  
  
def load\_dict(path):  
 tmp = set()  
 with open(path, 'r', encoding='utf-8') as f:  
 for line in f:  
 word = line.strip().split(' ')[0]  
 tmp.add(word)  
 return tmp  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 words = load\_dict('dict.txt')  
 max\_len = max(map(len, [word for word in words]))  
  
 # test  
 tokenizer = Tokenizer(words, max\_len)  
 texts = [  
 '研究生命的起源',  
 '无线电法国别研究',  
 '人要是行，干一行行一行，一行行行行行，行行行干哪行都行。'  
 ]  
 for text in texts:  
 # 前向最大匹配  
 print('前向最大匹配:', '/'.join(tokenizer.fmm\_split(text)))  
 # 后向最大匹配  
 print('后向最大匹配:', '/'.join(tokenizer.rmm\_split(text)))  
 # 双向最大匹配  
 print('双向最大匹配:', '/'.join(tokenizer.bimm\_split(text)))  
 print('')

# 附录B 命名实体识别实现的源程序

训练代码run.py

import pickle  
import logging  
import argparse  
import os  
import torch  
from torch.utils.data import DataLoader  
from torch.optim import Adam  
from model import CWS  
from dataloader import Sentence  
  
def get\_param():  
 parser = argparse.ArgumentParser()  
 parser.add\_argument('--embedding\_dim', type=int, default=100)  
 parser.add\_argument('--lr', type=float, default=0.005)  
 parser.add\_argument('--max\_epoch', type=int, default=10)  
 parser.add\_argument('--batch\_size', type=int, default=128)  
 parser.add\_argument('--hidden\_dim', type=int, default=200)  
 parser.add\_argument('--cuda', action='store\_true', default=False)  
 return parser.parse\_args()  
  
  
def set\_logger():  
 log\_file = os.path.join('save', 'log.txt')  
 logging.basicConfig(  
 format='%(asctime)s %(levelname)-8s %(message)s',  
 level=logging.DEBUG,  
 datefmt='%Y-%m%d %H:%M:%S',  
 filename=log\_file,  
 filemode='w',  
 )  
  
 console = logging.StreamHandler()  
 console.setLevel(logging.DEBUG)  
 formatter = logging.Formatter('%(asctime)s %(levelname)-8s %(message)s')  
 console.setFormatter(formatter)  
 logging.getLogger('').addHandler(console)  
  
  
def entity\_split(x, y, id2tag, entities, cur):  
 start, end = -1, -1  
 for j in range(len(x)):  
 if id2tag[y[j]] == 'B':  
 start = cur + j  
 elif id2tag[y[j]] == 'M' and start != -1:  
 continue  
 elif id2tag[y[j]] == 'E' and start != -1:  
 end = cur + j  
 entities.add((start, end))  
 start, end = -1, -1  
 elif id2tag[y[j]] == 'S':  
 entities.add((cur + j, cur + j))  
 start, end = -1, -1  
 else:  
 start, end = -1, -1  
  
  
def main(args):  
 use\_cuda = args.cuda and torch.cuda.is\_available()  
  
 with open('data/ner\_datasave.pkl', 'rb') as inp:  
 word2id = pickle.load(inp)  
 id2word = pickle.load(inp)  
 tag2id = pickle.load(inp)  
 id2tag = pickle.load(inp)  
 x\_train = pickle.load(inp)  
 y\_train = pickle.load(inp)  
 x\_test = pickle.load(inp)  
 y\_test = pickle.load(inp)  
  
 model = CWS(len(word2id), tag2id, args.embedding\_dim, args.hidden\_dim)  
 if use\_cuda:  
 model = model.cuda()  
 for name, param in model.named\_parameters():  
 logging.debug('%s: %s, require\_grad=%s' % (name, str(param.shape), str(param.requires\_grad)))  
  
 optimizer = Adam(model.parameters(), lr=args.lr)  
  
 train\_data = DataLoader(  
 dataset=Sentence(x\_train, y\_train),  
 shuffle=True,  
 batch\_size=args.batch\_size,  
 collate\_fn=Sentence.collate\_fn,  
 drop\_last=False,  
 num\_workers=6  
 )  
  
 test\_data = DataLoader(  
 dataset=Sentence(x\_test[:1000], y\_test[:1000]),  
 shuffle=False,  
 batch\_size=args.batch\_size,  
 collate\_fn=Sentence.collate\_fn,  
 drop\_last=False,  
 num\_workers=6  
 )  
  
 for epoch in range(args.max\_epoch):  
 step = 0  
 log = []  
 for sentence, label, mask, length in train\_data:  
 if use\_cuda:  
 sentence = sentence.cuda()  
 label = label.cuda()  
 mask = mask.cuda()  
  
 # forward  
 loss = model(sentence, label, mask, length)  
 log.append(loss.item())  
  
 # backward  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
 step += 1  
 if step % 100 == 0:  
 logging.debug('epoch %d-step %d loss: %f' % (epoch, step, sum(log)/len(log)))  
 log = []  
  
 path\_name = "./save/model\_epoch" + str(epoch) + ".pkl"  
 torch.save(model, path\_name)  
 logging.info("model has been saved in %s" % path\_name)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 set\_logger()  
 main(get\_param())