

**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201801**

**学 号： U201815553**

**姓 名： 周大伟**

**指导教师： 魏巍**

**报告日期： 2021年 6月 15日**

**计算机科学与技术学院**

目 录

[中文分词实现 2](#_Toc29227)

[1 问题描述 2](#_Toc1892)

[1.1 基础任务 2](#_Toc11755)

[1.2 选做任务 2](#_Toc30447)

[2 模块设计-基于词典 3](#_Toc22662)

[2.1 前向最大匹配 3](#_Toc26252)

[2.2 后向最大匹配 3](#_Toc10768)

[2.3 双向最大匹配 4](#_Toc17650)

[3 模块设计-基于统计学习 4](#_Toc31086)

[3.1 data\_u.py 4](#_Toc27710)

[3.2 dataloader.py 5](#_Toc17396)

[3.3 infer.py 5](#_Toc18833)

[3.4 model.py 5](#_Toc31468)

[3.5 run.py 5](#_Toc9975)

[3 代码实现-基于词典 5](#_Toc3703)

[3.1 前向最大匹配 5](#_Toc20802)

[3.2 后向最大匹配 6](#_Toc10348)

[3.3 双向最大匹配 7](#_Toc25083)

[4 运行结果 9](#_Toc24343)

[5 实验小结 9](#_Toc30103)

[附录 中文分词实现的源程序 9](#_Toc9993)

[data\_u.py 9](#_Toc6287)

[dataloader.py 12](#_Toc17498)

[infer.py 13](#_Toc18963)

[model.py 14](#_Toc16139)

[run.py 15](#_Toc10601)

# 中文分词实现

## 1 问题描述

中文分词指的是将一个汉字序列切分成一个一个单独的词。中文分词是文本挖掘的基础，对于输入的一段中文，成功的进行中文分词，可以达到电脑自动识别语句含义的效果。它是信息提取、信息检索、机器翻译、文本分类、自动文摘、语音识别、文本语音转换、自然语言理解等中文信息处理领域的基础。

### 1.1 基础任务

1. 实现基于词典的分词算法

实验一资料包下的“Dictionary\_based”文件夹中提供了基础词典和分词算法的大致框架。分词算法的核心部分需要大家完成，实验中提供了若干测试样本用以帮助大家判断算法是否正确实现。

1. 实现基于统计学习的分词算法

实验中给出Bi-LSTM+CRF模型的基础实现，相关代码及说明文档位于实验一资料包下的“Bi-LSTM+CRF”文件夹下。请根据给定的实验资料中README.md文件配置相应实验环境，说明：（1）提供源码PyTorch语言编写（可根据个人掌情况用其他语言编写），默认运行版本是CPU版本；（2）如希望运行NPU版本，大家可跟任课老师联系，申请华为云资源运行（需提前统计名单：姓名+学号+个人手机号码+邮箱）；

### 1.2 选做任务

优化基础任务中实现的分词器，可考虑的优化方案有：

1. 修改网络结构，例如引入BERT等预训练语言模型；
2. 与命名实体识别算法相互配合，减少对命名实体的错误分割；
3. 构造合适的词典集（可扩充+人工整理）；
4. 实现新词发现（登录）功能，识别测试集中的新词（未登录词）；
5. 调整、优化模型训练过程中的超参数。

完成优化后对测试文件“Bi-LSTM+CRF/data/test.txt”进行分词，分词结果保存到.txt文件中utf-8编码，词与词之间以空格分隔，每个测试样本占一行。文件“Bi-LSTM+CRF/cws\_result.txt”中给出了输出示例。提交分词结果后，依据单词级别的F1-score进行评判，决定选做部分的实验分数。

单词级别的F1-score的计算方式如下：

Gold: 共同 创造 美好 的 新 世纪 —— 二○○一年 新年 贺词

Hypothesis: 共同 创造 美 好 的 新 世纪 —— 二○○一年 新年 贺词

Precision = 9 / 11 = 0.818

Recall = 9 / 10 = 0.9

F1-score = 2\*Precision\*Recall/(Precision+Recall)=0.857

## 2 模块设计-基于词典

### 2.1 前向最大匹配

从待分词句子的左边向右边搜索，寻找词的最大匹配。我们需要规定一个词的最大长度，每次扫描的时候寻找当前开始的这个长度的词来和字典中的词匹配，如果没有找到，就缩短长度继续寻找，直到找到字典中的词或者成为单字。

算法流程如下：

（1）从待分词子串中从前往后取出max\_len个字，然后扫描分词字典，测试该max\_len个字的子串是否在字典中；

（2）如果存在，则从待分词子串中删除掉该max\_len个字的子串，重新按照规则取子串，重复（1）；

（3）如果不存在于字典中，则减少该子串的最右一个字，之后重复（1）。

### 2.2 后向最大匹配

从待分词句子的右边向左边搜索，寻找词的最大匹配。同样，我们也需要规定一个词的最大长度，每次扫描的时候寻找当前开始的这个长度的词来和字典中的词匹配，如果没有找到，就缩短长度继续寻找，直到找到字典中的词或者成为单字。

算法流程如下：

（1）从待分词子串中从后往前取出max\_len个字，然后扫描分词字典，测试该max\_len个字的子串是否在字典中；

（2）如果存在，则从待分词子串中删除掉该max\_len个字的子串，重新按照规则取子串，重复（1）；

（3）如果不存在于字典中，则减少该子串的最左一个字，之后重复（1）。

### 2.3 双向最大匹配

将前向最大匹配算法和后向最大匹配算法进行比较，从而确定正确的分词方法。

算法流程如下：

（1）比较前向最大匹配和后向最大匹配结果；

（2）如果分词结果相同，返回其中任意一个；

（3）如果分词结果不同：

1、比较两者分词总数量，数量高者罚分；

2、比较两者分词中单字词数量，单字词多者罚分；

3、比较两者分词中非字典词数量，非字典词多者罚分；

4、选择罚分最少的作为最终结果。

## 3 模块设计-基于统计学习

### 3.1 data\_u.py

(1) getist：单个分词转换成tag序列。按行读入数据，并分析各个字对应的标签，然后返回分析结果。

(2) handle\_data：处理数据，并保存至save\_path。按行读取对应文件中的数据，并做相应的处理，然后把处理的结果保存到data\_save.pkl中。

### 3.2 dataloader.py

读取通过data\_u.py处理完后的文件data\_save.pkl，并将其向量化。

### 3.3 infer.py

通过已经训练好的模型，完成对测试文件的分析，并将分词结果保存到cws\_result.txt文件中。

### 3.4 model.py

(1) init\_hidden：通过torch.randn函数进行初始化操作。

(2) \_get\_lstm\_features：获取LSTM框架。

(3) forward：预测每个标签的loss值，以减少无效预测。

(4) infer：采用Bi-LSTM+CRF的基础结构的分析结果。

### 3.5 run.py

采用小批量梯度下降法，对模型进行训练，使得loss值降低。

小批量梯度下降，是对批量梯度下降以及随机梯度下降的一个折中办法。其思想是：每次迭代 使用 batch\_size个样本来对参数进行更新，每次使用一个batch可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。

## 3 代码实现-基于词典

### 3.1 前向最大匹配

def forward\_mm\_split(self, fmm\_text):  
 """  
 正向最大匹配分词算法  
 :param fmm\_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  
 """  
 # 字词列表，存放分词结果  
 word\_list = []  
 # 用于记录分词的起始位置  
 count = 0  
 # 字或词当前的长度  
 word\_len = self.max\_len  
 while word\_len > 0 and count < len(fmm\_text):  
 word = fmm\_text[count:count + word\_len]  
 word\_len = len(word)  
 if (word in self.words) or (word in self.delimiter):  
 word\_list.append(word)  
 count = count + word\_len  
 word\_len = self.max\_len  
 else:  
 word\_len = word\_len - 1  
 return word\_list

### 3.2 后向最大匹配

def reverse\_mm\_split(self, rmm\_text):  
 """  
 逆向最大匹配分词算法  
 :param rmm\_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  
 """  
 # 字词列表，存放分词结果  
 word\_list = []  
 # 用于记录分词的末尾位置  
 count = len(rmm\_text)  
 # 字或词当前的长度  
 word\_len = self.max\_len  
 while word\_len > 0 and count > 0:  
 if count <= word\_len:  
 word = rmm\_text[:count]  
 else:  
 word = rmm\_text[(count - word\_len):count]  
 word\_len = len(word)  
 if (word in self.words) or (word in self.delimiter):  
 word\_list.insert(0, word)  
 count = count - word\_len  
 word\_len = self.max\_len  
 else:  
 word\_len = word\_len - 1  
 return word\_list

### 3.3 双向最大匹配

def bidirectional\_mm\_split(self, bi\_text):  
 """  
 双向最大匹配分词算法  
 :param bi\_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  
 """  
 # 前向最大匹配得到的分词结果  
 forward = self.forward\_mm\_split(bi\_text)  
 # 后向最大匹配得到的分词结果  
 reverse = self.reverse\_mm\_split(bi\_text)  
 # 总词数  
 forward\_total\_words = len(forward)  
 reverse\_total\_words = len(reverse)  
 # 单字词个数  
 forward\_single\_words = 0  
 reverse\_single\_words = 0  
 # 非字典词数  
 forward\_illegal\_words = 0  
 reverse\_illegal\_words = 0  
 # 罚分，分值越低，表明结果越好  
 forward\_score = 0  
 reverse\_score = 0  
 if forward == reverse:  
 return forward  
 else:  
 # 统计前向匹配的各个词情况  
 for word in forward:  
 if len(word) == 1:  
 forward\_single\_words += 1  
 if word not in self.words:  
 forward\_illegal\_words += 1  
 # 统计后向匹配的各个词情况  
 for word in reverse:  
 if len(word) == 1:  
 reverse\_single\_words += 1  
 if word not in self.words:  
 reverse\_illegal\_words += 1  
 # 计算罚分  
 if forward\_total\_words < reverse\_total\_words:  
 reverse\_score += 1  
 else:  
 forward\_score += 1  
 if forward\_illegal\_words < reverse\_illegal\_words:  
 reverse\_score += 1  
 else:  
 forward\_score += 1  
 if forward\_single\_words < reverse\_single\_words:  
 reverse\_score += 1  
 else:  
 forward\_score += 1  
 # 比较罚分情况，罚分最小的选做最终结果  
 if forward\_score < reverse\_score:  
 return forward  
 else:  
 return reverse

## 4 运行结果

利用实验包已给出的代码框架，实现完对应的匹配函数后，运行测试样例。

1. 基于词典的中文分词，得到输出结果如下：

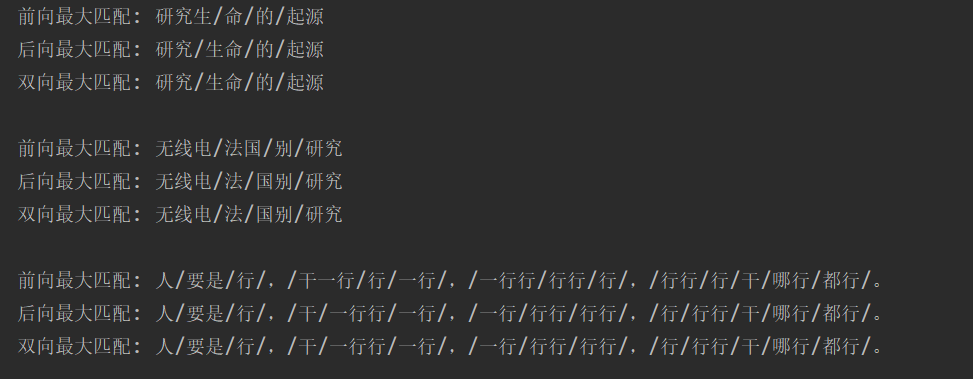


图 1-1 中文分词测试结果

1. 基于统计学习的中文分词，将输出的分词文件进行在线测评，结果如下：

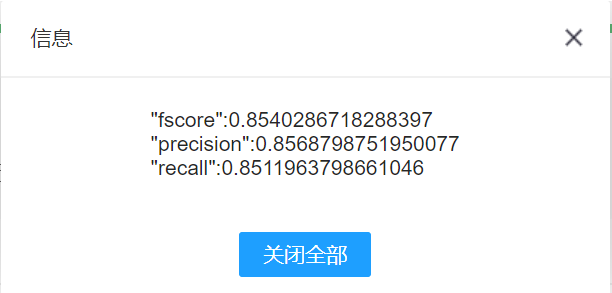


图 1-2 在线测评结果

## 5 实验小结

实验中总体的代码框架已给出，需要实现的部分为前向最大匹配、后向最大匹配和双向最大匹配三个核心函数。通过相关资料的参考以及对中文分词的个人理解，能够顺利实现各个匹配模式的实现。

基于统计学习的中文分词代码，由于已经给出了代码框架，所以只是在其基础上做了部分优化，并增加了一部分的训练数据。

# 附录 中文分词实现的源程序

## data\_u.py

import pickle  
  
from sklearn.model\_selection import train\_test\_split  
  
INPUT\_DATA = "train.txt"  
SAVE\_PATH = "./data\_save.pkl"  
id2tag = ['B', 'M', 'E', 'S'] # B：分词头部 M：分词词中 E：分词词尾 S：独立成词  
tag2id = {'B': 0, 'M': 1, 'E': 2, 'S': 3}  
word2id = {}  
id2word = []  
  
  
def getlist(input\_str):  
 """  
 单个分词转换为tag序列  
 :param input\_str: 单个分词  
 :return: tag序列  
 """  
 out\_str = []  
 if len(input\_str) == 1:  
 out\_str.append(tag2id['S'])  
 elif len(input\_str) == 2:  
 out\_str = [tag2id['B'], tag2id['E']]  
 else:  
 m\_num = len(input\_str) - 2  
 m\_list = [tag2id['M']] \* m\_num  
 out\_str.append(tag2id['B'])  
 out\_str.extend(m\_list)  
 out\_str.append(tag2id['E'])  
 return out\_str  
  
  
def handle\_data():  
 """  
 处理数据，并保存至save\_path  
 :return:  
 """  
 x\_data = []  
 y\_data = []  
 word\_num = 0  
 line\_num = 0  
 with open(INPUT\_DATA, 'r', encoding="utf-8") as ifp:  
 for line in ifp:  
 line\_num = line\_num + 1  
 line = line.strip()  
 if not line:  
 continue  
 line\_x = []  
 for i in range(len(line)):  
 if line[i] == " ":  
 continue  
 if line[i] in id2word:  
 line\_x.append(word2id[line[i]])  
 else:  
 id2word.append(line[i])  
 word2id[line[i]] = word\_num  
 line\_x.append(word\_num)  
 word\_num = word\_num + 1  
 x\_data.append(line\_x)  
  
 line\_arr = line.split()  
 line\_y = []  
 for item in line\_arr:  
 line\_y.extend(getlist(item))  
 y\_data.append(line\_y)  
  
 print(x\_data[0])  
 print([id2word[i] for i in x\_data[0]])  
 print(y\_data[0])  
 print([id2tag[i] for i in y\_data[0]])  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size=0.1, random\_state=43)  
 with open(SAVE\_PATH, 'wb') as output:  
 pickle.dump(word2id, output)  
 pickle.dump(id2word, output)  
 pickle.dump(tag2id, output)  
 pickle.dump(id2tag, output)  
 pickle.dump(x\_train, output)  
 pickle.dump(y\_train, output)  
 pickle.dump(x\_test, output)  
 pickle.dump(y\_test, output)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 handle\_data()

## dataloader.py

import torch  
import pickle  
from torch.utils.data import Dataset, DataLoader  
from torch.nn.utils.rnn import pad\_sequence  
  
  
class Sentence(Dataset):  
 def \_\_init\_\_(self, x, y, batch\_size=10):  
 self.x = x  
 self.y = y  
 self.batch\_size = batch\_size  
  
 def \_\_len\_\_(self):  
 return len(self.x)  
  
 def \_\_getitem\_\_(self, idx):  
 assert len(self.x[idx]) == len(self.y[idx])  
 return self.x[idx], self.y[idx]  
  
 @staticmethod  
 def collate\_fn(train\_data):  
 train\_data.sort(key=lambda data: len(data[0]), reverse=True)  
 data\_length = [len(data[0]) for data in train\_data]  
 data\_x = [torch.LongTensor(data[0]) for data in train\_data]  
 data\_y = [torch.LongTensor(data[1]) for data in train\_data]  
 masks = [torch.ones(i, dtype=torch.uint8) for i in data\_length]  
 data\_x = pad\_sequence(data\_x, batch\_first=True, padding\_value=0)  
 data\_y = pad\_sequence(data\_y, batch\_first=True, padding\_value=0)  
 masks = pad\_sequence(masks, batch\_first=True, padding\_value=0)  
 return data\_x, data\_y, masks, data\_length  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # test  
 with open('../data/data\_save.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)  
  
 train\_dataloader = DataLoader(Sentence(x\_train, y\_train), batch\_size=10, shuffle=True,  
 collate\_fn=Sentence.collate\_fn)  
  
 for input\_data, label, mask, length in train\_dataloader:  
 print(input\_data, label)  
 break

## infer.py

import torch  
import pickle  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 model = torch.load('save/model\_epoch9.pkl', map\_location=torch.device('cpu'))  
 output = open('cws\_result.txt', 'w', encoding='utf-8')  
  
 with open('data/data\_save.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)  
  
 with open('data/test\_final.txt', 'r', encoding='utf-8') as f:  
 for test in f:  
 flag = False  
 test = test.strip()  
  
 x = torch.LongTensor(1, len(test))  
 mask = torch.ones\_like(x, dtype=torch.uint8)  
 length = [len(test)]  
 for i in range(len(test)):  
 if test[i] in word2id:  
 x[0, i] = word2id[test[i]]  
 else:  
 x[0, i] = len(word2id)  
  
 predict = model.infer(x, mask, length)[0]  
 for i in range(len(test)):  
 print(test[i], end='', file=output)  
 if id2tag[predict[i]] in ['E', 'S']:  
 print(' ', end='', file=output)  
 print(file=output)

## model.py

import torch  
import torch.nn as nn  
from torchcrf import CRF  
from torch.nn.utils.rnn import pack\_padded\_sequence, pad\_packed\_sequence  
  
  
class CWS(nn.Module):  
  
 def \_\_init\_\_(self, vocab\_size, tag2id, embedding\_dim, hidden\_dim):  
 super(CWS, self).\_\_init\_\_()  
 self.embedding\_dim = embedding\_dim  
 self.hidden\_dim = hidden\_dim  
 self.vocab\_size = vocab\_size  
 self.tag2id = tag2id  
 self.tag\_set\_size = len(tag2id)  
  
 self.word\_embeds = nn.Embedding(vocab\_size + 1, embedding\_dim)  
  
 self.lstm = nn.LSTM(embedding\_dim, hidden\_dim // 2, num\_layers=1,  
 bidirectional=True, batch\_first=True)  
 self.hidden2tag = nn.Linear(hidden\_dim, self.tag\_set\_size)  
  
 self.crf = CRF(4, batch\_first=True)  
  
 def init\_hidden(self, batch\_size, device):  
 return (torch.randn(2, batch\_size, self.hidden\_dim // 2, device=device),  
 torch.randn(2, batch\_size, self.hidden\_dim // 2, device=device))  
  
 def \_get\_lstm\_features(self, sentence, length):  
 batch\_size, seq\_len = sentence.size(0), sentence.size(1)  
  
 # idx->embedding  
 embeds = self.word\_embeds(sentence.view(-1)).reshape(batch\_size, seq\_len, -1)  
 embeds = pack\_padded\_sequence(embeds, length, batch\_first=True)  
  
 # LSTM forward  
 self.hidden = self.init\_hidden(batch\_size, sentence.device)  
 lstm\_out, self.hidden = self.lstm(embeds, self.hidden)  
 lstm\_out, \_ = pad\_packed\_sequence(lstm\_out, batch\_first=True)  
 lstm\_feats = self.hidden2tag(lstm\_out)  
 return lstm\_feats  
  
 def forward(self, sentence, tags, mask, length):  
 emissions = self.\_get\_lstm\_features(sentence, length)  
 loss = -self.crf(emissions, tags, mask, reduction='mean')  
 return loss  
  
 def infer(self, sentence, mask, length):  
 emissions = self.\_get\_lstm\_features(sentence, length)  
 return self.crf.decode(emissions, mask)

## 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/data\_save.pkl', 'rb') as inp:  
 word2id = pickle.load(inp)  
 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 = []  
  
 # test  
 entity\_predict = set()  
 entity\_label = set()  
 with torch.no\_grad():  
 model.eval()  
 cur = 0  
 for sentence, label, mask, length in test\_data:  
 if use\_cuda:  
 sentence = sentence.cuda()  
 label = label.cuda()  
 mask = mask.cuda()  
 predict = model.infer(sentence, mask, length)  
  
 for i in range(len(length)):  
 entity\_split(sentence[i, :length[i]], predict[i], id2tag, entity\_predict, cur)  
 entity\_split(sentence[i, :length[i]], label[i, :length[i]], id2tag, entity\_label, cur)  
 cur += length[i]  
  
 right\_predict = [i for i in entity\_predict if i in entity\_label]  
 if len(right\_predict) != 0:  
 precision = float(len(right\_predict)) / len(entity\_predict)  
 recall = float(len(right\_predict)) / len(entity\_label)  
 logging.info("precision: %f" % precision)  
 logging.info("recall: %f" % recall)  
 logging.info("score: %f" % ((2 \* precision \* recall) / (precision + recall)))  
 else:  
 logging.info("precision: 0")  
 logging.info("recall: 0")  
 logging.info("score: 0")  
 model.train()  
  
 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())