

**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 CS1903**

**学 号： U201914995**

**姓 名： 罗虞阳**

**指导教师： 姚德中**

**报告日期： 2022年 6月 19日**

**计算机科学与技术学院**

目 录

[中文命名实体识别实现 2](#_Toc106739507)

[1 问题描述 2](#_Toc106739508)

[1.1 基础任务 2](#_Toc106739509)

[1.2 选做任务 3](#_Toc106739510)

[2 基础模块 3](#_Toc106739511)

[2.1 系统模块划分 3](#_Toc106739512)

[2.2 系统模块设计 4](#_Toc106739513)

[2.3 系统模块优化 4](#_Toc106739514)

[3 系统实现 5](#_Toc106739515)

[3.1系统模块实现 5](#_Toc106739516)

[3.2 实验结果及分析 6](#_Toc106739517)

[4 实验小结 6](#_Toc106739518)

[参考文献 8](#_Toc106739519)

[附录A 命名实体识别实现的源程序 9](#_Toc106739520)

[0\_split.py 9](#_Toc106739521)

[1\_data\_u\_ner.py 10](#_Toc106739522)

[dataloader.py 14](#_Toc106739523)

[infer.py 15](#_Toc106739524)

[model.py 17](#_Toc106739525)

[my\_infer.py 19](#_Toc106739526)

[run.py 20](#_Toc106739527)

# 中文命名实体识别实现

## 1 问题描述

命名实体识别的研究是自然语言处理中的一项基本工作，不仅是分词和标注过程中的一个重要的环节，而且在句法分析、机器翻译、信息检索、提取以及自动问答系统等领域中也有直接的应用。由于中文文本中词与词之间没有分隔符，中文文本的分词和中文命名实体的识别是相互缠绕、密不可分的。

一般来说，命名实体识别的任务就是对于一片待处理文本，识别出其中出现的人名、地名、机构名、日期、时间、百分数、货币这七类命名实体。其中人名、地名、机构名的识别是最难，也是最重要的三类。

本次实验的目的是通过实验了解中文分词的大致思路。在实验中实现中文分词处理，可考虑使用课堂讲解过的算法（比如基于统计、基于词典的分词方法等），或者课外学习算法（比如Bi-LSTM+CRF模型等）进行分词，最后对比不同算法分词效果和性能，加深对中文分词算法的理解。

### 基础任务

1. 实现基于Bi-LSTM+CRF的命名实体识别算法

实验二资料包下的“RMRB\_NER\_CORPUS.txt”文件中提供了基于人民日报的NER标注数据，需要对数据集进行合理比例的划分，使其可以用于训练命名实体识别模型。

分词实验与命名实体识别实验所采用的模型有一定交集，因此除了自主实现模型以外，还可以参考实验1必做项中给出的Bi-LSTM+CRF标准实现并对其进行部分修改。若选择对实验1必做项中的Bi-LSTM+CRF模型进行修改，主要需要修改的部分包括数据预处理、模型的输入输出层。

1. 尝试用命名实体识别算法提升分词模型的性能

命名实体识别结果将对特定名词的识别产生提升效果，请你尝试利用NER模型结果优化实验一中的分词结果。请自行设计融合策略，并在实验报告中进行说明。

### 选做任务

为了进一步优化实验一的分词结果，可以从以下角度进行改进：

1. 优化命名实体识别模型，可考虑的优化方案有：
2. 修改网络结构，例如引入BERT等预训练语言模型；
3. 调整、优化模型训练过程中的超参数。
4. 数据增强

实验二提供的人民日报语料与分词所采用的语料并不一定是同分布的，你可以自行搜集更为合适的数据集进行训练。

1. 调整融合策略

## 2 基础模块

### 2.1 系统模块划分

系统主要分为数据预处理模块，模型构建模块和结果推断模块这三个模块，系统各模块的组成和层次结构如图2.1所示。每一个模块的具体设计在下一节给出。

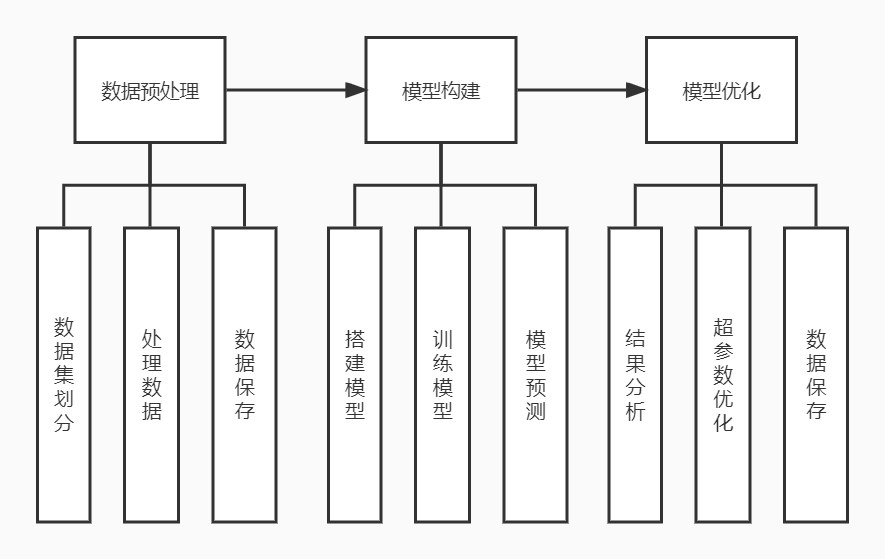


图2.1 模块组成和层次结构

### 2.2 系统模块设计

系统主要分为数据预处理，模型构建和模型优化三个部分，下面介绍每个部分的详细设计。

数据预处理首先是将原始数据集划分为训练集（train），验证集（valid）和测试集（test），划分比例为8:1:1，其中训练集是用来训练参数和梯度下降，验证集是在每个epoch完成后，测试当前模型的准确率，测试集用来在模型最终训练完成后测试准确率，然后将划分好的三个数据集保存起来。原始人民日报语料库数据中是字和对应的BIOES标注，并将其进行编码，然后将字和标注分别保存到两个集合中，方便后续训练和验证，到此数据预处理部分完成。

BiLSTM-CRF模型主体有双向长短时记忆网络和条件随机场组成，模型输入的是字符特征，输出时每个字符对应的预测标签。输入的句子通过embedding layer后生成词向量。BiLSTM接收每个字符的embedding，并预测每个字符的对应于几个标注标签的概率。CEF层将BiLSTM的emmision作为输入，输出符合标注转移约束条件的最大可能的预测标注序列。

模型优化主要是对模型的超参数进行人工调参，从中选取得分最高的参数作为最后确定的参数进行计算，得出文本的BIOES标注。

### 2.3 系统模块优化

在优化上，我主要进行的是对模型超参数的调整和融合策略的改进。

模型优化主要是对模型的超参数进行人工调参，从中选取得分最高的参数作为最后确定的参数进行计算，得出文本的BIOES标注。主要是对embedding维度，学习率，训练迭代次数和隐藏层维度进行人工调参，多次尝试求得最优参数。

新添加融合模型，利用模型预测得到的结果优化实验一中的中文分词结果，严格按照粗粒度有限的策略进行结果合并，即除非句子中单词的分词数量与所减少，否则不改变分词的结果，最后按照得到的标签序列，对测试文件进行分割。

## 3 系统实现

### 3.1系统模块实现

data\_u\_ner.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，方便后续直接进行读取。

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。

infer.py

首先读取输入测试文件中的句子，使用与数据预处理相同的方式处理输入句子后输入到加载好的训练模型中，得到输出后将标签保存到cws\_result.txt中，等待后续处理。

model.py

model中需要初始化embedding层维度，隐藏层维度，输入词数目等参数，然后使用torch.nn中的模型构建出embedding层，LSTM，CRF以及隐藏层。通过lstm的前向传播得到该层的emission，经过CRF层计算得到loss，然后通过loss进行权重的更新，达到训练模型的效果。

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。具体实现是将测试集中的每行数据和对应标签输入到模型中计算loss，然后将loss进行反向传播更新权重。每一次迭代都将模型的参数信息进行保存。

在测试集上，依次比较模型预测的标签和真是标签的差别，统计出precision，recall和f1score这三个统计量。

split.py

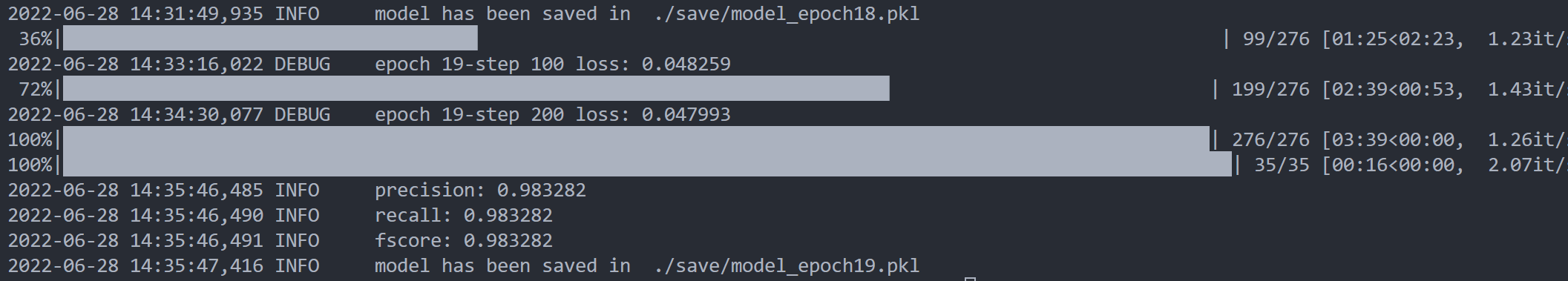
将数据集划分为训练集和测试集。

Mergy.py

读取中文分词标签文件，中文命名实体标注文件，测试数据集文件，介个中文分词标签和中文命名实体标注这两个文件，使用粗粒度划分的策略，如果中文命名实体划分让句子中分词数目减少，则合并，否则分词结果不变。最后将更新后的分词结果保存到cws\_resutl.txt中。

### 3.2 实验结果及分析

模型的超参数设置如下：embedding\_dim设置为768，学习率设为0.0001，最大max\_epoch为50, batch\_size设置为128, hidden\_dim设置为256，实验最后的loss为0.047993，准确率为0.983282。



中文分词原始准确率在在线测评上结果为84.4，引入中文命名实体识别的结果后准确率提升到85.5,提升幅度较小。

原始模型和进行调参后的模型得到的结果对比，提升幅度不大，可能原因和改进处有如下几点。首先是数据集较少，人民日报语料库中的单词无法完全覆盖测试集中的单词，而且人名日报语料库中的数据可能和测试集中的数据不是独立同分布，这会影响模型的判断，因为模型还是根据概率来进行分词和实体识别的判断；其次，调参后的模型只对参数进行了修改和优化，但是没有完全覆盖参数值域，导致尽管可能参数得到优化，但是不是该模型针对该数据集的最优参数，解决方法是可以使用格子搜索，随机搜索或者贝叶斯优化来自动调参；再其次，模型还可以引入bert等预训练好的模型，再在之上进行微调，可能会取得更好的效果。

## 4 实验小结

本次实验主要完成了中文命名实体识别的任务。基础代码已经由老师给出，主要任务是进行模型优化的探索和将中文命名实体识别的结果同中文分词的结果相融合，优化提升中文分词模型结果的准确率。

在这次的实验中，尽管我尝试调节了参数并枚举了较多的参数值，但是结果提升并不是很明显，在上一小节的实验结果及分析这一部分中我也详细的分析了原因和待改进的地方，之后如果有时间，我会学习如何使用bert来对现有模型进行优化。同时我也尝试利用实验中得到的中文命名实体识别结果提升中文分词结果的准确性，但是提升效果十分不明显，可能是融合策略或者模型预测结果有待提高，未来如果有时间，我会继续尝试优化该模型，学习其他优化方法进行优化，提升模型预测准确性。

通过此次试验，我了解了针对某个NLP任务的数据分析处理，模型搭建，调参，模型优化的过程，学习到了BiLSTM和CRF的相关知识，是关于NLP相关实验的一次宝贵体验，我在这个过程中收获了很多相关专业知识，也对这个领域有了更深的了解，有助于我未来对研究方向的选择。

# 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 命名实体识别实现的源程序

## 0\_split.py

import random

corpus\_file = 'RMRB\_NER\_CORPUS.txt'

corpus = []

with open(corpus\_file, 'r', encoding='utf-8')as f:

record = []

for line in f:

if line != '\n':

record.append(line.strip('\n').split(' '))

else:

corpus.append(record)

record = []

random.seed(43)

random.shuffle(corpus)

fullLen = len(corpus)

splitLen = len(corpus) // 10

train = corpus[:splitLen \* 8]

valid = corpus[splitLen \* 8:splitLen \* 9]

test = corpus[splitLen \* 9:]

train\_file = 'ner\_train.txt'

valid\_file = 'ner\_valid.txt'

test\_file = 'ner\_test.txt'

for split\_file, split\_corpus in zip([train\_file, valid\_file, test\_file],

[train, valid, test]):

with open(split\_file, 'w')as f:

for sentence in split\_corpus:

for word, label in sentence:

f.write(word)

f.write(' ')

f.write(label)

f.write('\n')

f.write('\n')

## 1\_data\_u\_ner.py

import pickle

INPUT\_DATA = "train.txt"

TRAIN\_DATA = "ner\_train.txt"

VALID\_DATA = "ner\_valid.txt"

SAVE\_PATH = "ner\_data\_save.pkl"

unique = set()

with open('ner\_train.txt', 'r')as f:

for line in f:

try:

unique.update([line.strip('\n').split(' ')[1]])

except:

pass

id2tag = list(unique)

print(id2tag)

tag2id = {}

for i, label in enumerate(id2tag):

tag2id[label] = i

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

"""

output\_str = []

if len(input\_str) == 1:

output\_str.append(tag2id['S'])

elif len(input\_str) == 2:

output\_str = [tag2id['B'], tag2id['E']]

else:

m\_num = len(input\_str) - 2

m\_list = [tag2id['M']] \* m\_num

output\_str.append(tag2id['B'])

output\_str.extend(m\_list)

output\_str.append(tag2id['E'])

return output\_str

def handle\_data():

"""

处理数据，并保存至save\_path

:return:

"""

output = open(SAVE\_PATH, 'wb')

x\_train = []

y\_train = []

x\_valid = []

y\_valid = []

word\_num = 0

with open(TRAIN\_DATA, 'r', encoding="utf-8") as ifp:

file\_set(x\_train, y\_train, ifp, word\_num)

with open(VALID\_DATA, 'r', encoding="utf-8") as ifp:

file\_set(x\_valid, y\_valid, ifp, word\_num)

print(x\_train[0])

print([id2word[temp] for temp in x\_train[0]])

print(y\_train[0])

print([id2tag[temp] for temp in y\_train[0]])

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\_valid, output)

pickle.dump(y\_valid, output)

output.close()

print("finish handle data")

def file\_set(x, y, ifp, word\_num):

line\_x = []

line\_y = []

for file\_line in ifp:

file\_line = file\_line.strip()

if not file\_line:

x.append(line\_x)

y.append(line\_y)

line\_x = []

line\_y = []

continue

file\_line = file\_line.split(' ')

if file\_line[0] in id2word:

line\_x.append(word2id[file\_line[0]])

else:

id2word.append(file\_line[0])

word2id[file\_line[0]] = word\_num

line\_x.append(word\_num)

word\_num = word\_num + 1

line\_y.append(tag2id[file\_line[1]])

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]

data\_mask = [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)

data\_mask = pad\_sequence(data\_mask, batch\_first=True, padding\_value=0)

return data\_x, data\_y, data\_mask, data\_length

if \_\_name\_\_ == '\_\_main\_\_':

# test

with open('../data/ner\_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 in\_put, label, mask, length in train\_dataloader:

print(in\_put, label)

break

## infer.py

import torch

import pickle

from tqdm import tqdm

if \_\_name\_\_ == '\_\_main\_\_':

model = torch.load('save/model\_epoch19.pkl', map\_location=torch.device('cpu'))

output = open('./data/ner\_res.txt', 'w', encoding='utf-8')

with open('data/ner\_data\_save.pkl', 'rb') as fnp:

word2id = pickle.load(fnp)

id2word = pickle.load(fnp)

tag2id = pickle.load(fnp)

id2tag = pickle.load(fnp)

x\_train = pickle.load(fnp)

y\_train = pickle.load(fnp)

x\_test = pickle.load(fnp)

y\_test = pickle.load(fnp)

num\_file=sum([1 for i in open('./data/test\_data.txt', 'r', encoding='utf-8')])

with open('./data/test\_data.txt', 'r', encoding='utf-8') as f:

line\_test = ''

for test in tqdm(f, total = num\_file):

flag = False

test = test.strip()

if not test:

test = test.split(' ')

x = torch.LongTensor(1, len(line\_test))

mask = torch.ones\_like(x, dtype=torch.uint8)

length = [len(line\_test)]

for i in range(len(line\_test)):

if line\_test[i] in word2id:

x[0, i] = word2id[line\_test[i]]

else:

x[0, i] = len(word2id)

predict = model.infer(x, mask, length)[0]

for i in range(len(line\_test)):

print(line\_test[i], id2tag[predict[i]], file=output)

print(file=output)

line\_test = ''

else:

test = test.split(' ')

line\_test += test[0]

## 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(21, 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)

## my\_infer.py

import torch

import pickle

from tqdm import tqdm

if \_\_name\_\_ == '\_\_main\_\_':

model = torch.load('save/model\_epoch29.pkl', map\_location=torch.device('cpu'))

output = open('./data/ner\_res.txt', 'w', encoding='utf-8')

with open('data/ner\_data\_save.pkl', 'rb') as fnp:

word2id = pickle.load(fnp)

id2word = pickle.load(fnp)

tag2id = pickle.load(fnp)

id2tag = pickle.load(fnp)

x\_train = pickle.load(fnp)

y\_train = pickle.load(fnp)

x\_test = pickle.load(fnp)

y\_test = pickle.load(fnp)

num\_file=sum([1 for i in open('./data/test\_data.txt', 'r', encoding='utf-8')])

with open('./data/test\_data.txt', 'r', encoding='utf-8') as f:

# line\_test = ''

for line\_test in tqdm(f, total = num\_file):

flag = False

line\_test = line\_test.strip()

#line\_test = line\_test.split(' ')

x = torch.LongTensor(1, len(line\_test))

mask = torch.ones\_like(x, dtype=torch.uint8)

length = [len(line\_test)]

for i in range(len(line\_test)):

if line\_test[i] in word2id:

x[0, i] = word2id[line\_test[i]]

else:

x[0, i] = len(word2id)

predict = model.infer(x, mask, length)[0]

for i in range(len(line\_test)):

print(id2tag[predict[i]], end=' ',file=output)

print(file=output)

## 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

from tqdm import tqdm

def get\_param():

parser = argparse.ArgumentParser()

parser.add\_argument('--embedding\_dim', type=int, default=768)

parser.add\_argument('--lr', type=float, default=0.001)

parser.add\_argument('--max\_epoch', type=int, default=20)

parser.add\_argument('--batch\_size', type=int, default=128)

parser.add\_argument('--hidden\_dim', type=int, default=256)

parser.add\_argument('--cuda', action='store\_true', default=True)

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\_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, y\_test),

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 = []

# train

for sentence, label, mask, length in tqdm(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

with torch.no\_grad():

model.eval()

cnt=0

amount=0

for sentence, label, mask, length in tqdm(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)):

predict\_label=predict[i]

ground\_true\_label=label[i]

for j in range(len(predict\_label)):

amount+=1

# print(predict\_label[j], ground\_true\_label[j].item())

if predict\_label[j]==ground\_true\_label[j].item():

cnt+=1

# 开始统计正确率信息

if cnt!=0:

precision=float(cnt)/amount

recall=float(cnt)/amount

logging.info("precision: %f" % precision)

logging.info("recall: %f" % recall)

logging.info("fscore: %f" % ((2 \* precision \* recall) / (precision + recall)))

else:

logging.info("precision: 0")

logging.info("recall: 0")

logging.info("fscore: 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())