

**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 校交1803**

**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

[中文命名实体识别实现 2](#_Toc3241)

[1 问题描述 2](#_Toc16363)

[1.1 基础任务 2](#_Toc18401)

[1.2 选做任务 2](#_Toc3005)

[2 基础模块 3](#_Toc10784)

[3 系统实现 4](#_Toc22912)

[3.1 data\_u\_ner.py 4](#_Toc4723)

[3.2 dataloader.py 4](#_Toc7764)

[3.3 infer.py 4](#_Toc29366)

[3.4 model.py 5](#_Toc30963)

[3.5 run.py 5](#_Toc22977)

[3.6 split.py 5](#_Toc9142)

[4 实验小结 5](#_Toc27620)

[附录 命名实体识别实现的源程序 6](#_Toc8100)

[data\_u\_ner.py 6](#_Toc9272)

[dataloader.py 8](#_Toc10482)

[infer.py 9](#_Toc19862)

[model.py 10](#_Toc18597)

[run.py 12](#_Toc22581)

[split.py 15](#_Toc13372)

# 中文命名实体识别实现

## 1 问题描述

命名实体识别的任务被定义为识别出文本中出现的专有名称和有意义的数量短语并加以归类。命名实体是文本中基本的信息元素，是正确理解文本的基础。狭义地讲，命名实体是指现实世界中的具体的或抽象的实体，如人、地点、机构等，通常用唯一的标志符（专有名称〉表示，如人名、地名、机构名等。广义地讲，命名实体还可以包含时间、数量表达式等。至于命名实体的确切含义，只能根据具体应用来确定。比如，在具体应用中，可能需要把住址、电子信箱地址、电话号码、会议名称等作为命名实体。

### 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

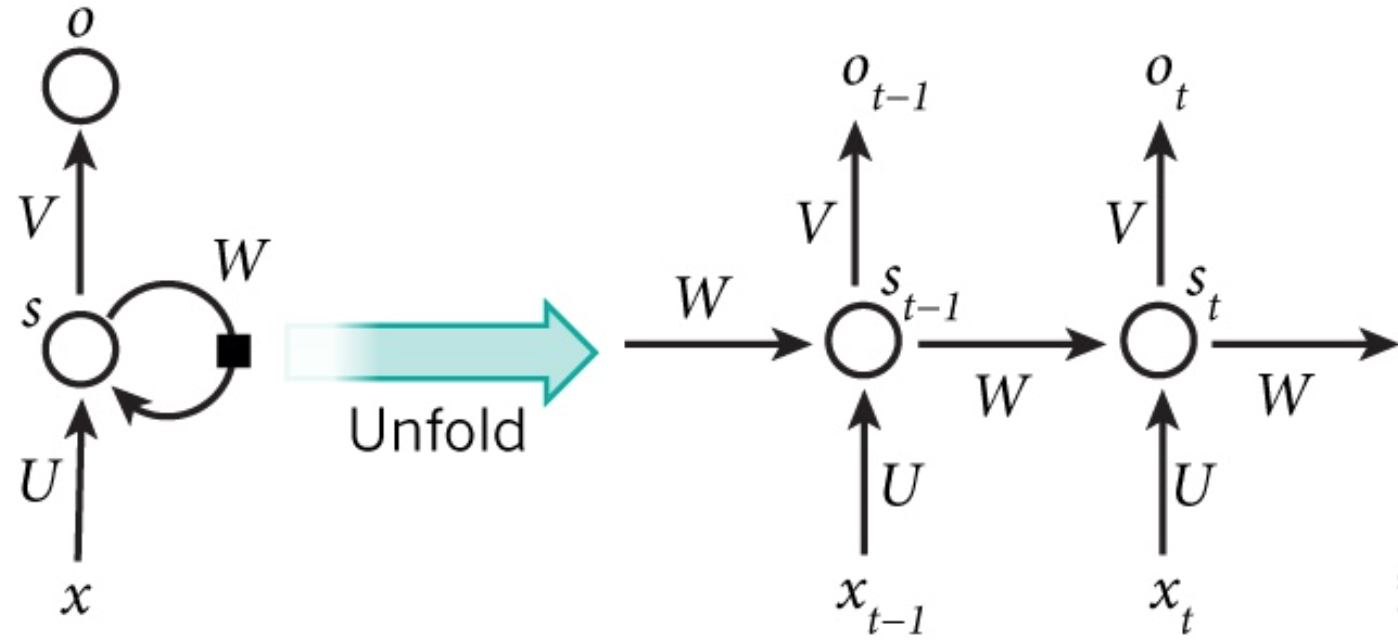
实验二提供的人民日报语料与分词所采用的语料并不一定是同分布的，你可以自行搜集更为合适的数据集进行训练。

1. 调整融合策略

## 2 基础模块

循环神经网络（RNN）被广泛运用于序列任务或时序任务，且具有良好的基线效果。在命名实体识别任务上，此类神经网络通常能够兼顾预测准确率与运行速度，得到了广泛运用。

RNN的基本结构可由下图表示：



是第层的输入，它可以是一个词的one-hot向量，也可以是概率分布表示；

是第层的隐藏状态，它负贵整个神经网络的记忆功能。由上一层的隐藏状态和本层输入共同决定，,通常是一个非线性的激活函数，比如tanh或ReLU。由于每一层的都会向后一直传递，所以理论上能够捕获到前面每一层发生的事情（但实际中太长的依赖很难训练）。

是第t层的输出，比如我有预测下一个词是什么时，就是一个长度为的向量，是所有词的总数，表示下一个词是的概率。最后用softmax对这些概率进行归一化

每一层的参数是共享的，这样极大地缩小了参数空间；每一层并不一定都得有输入和输出，比如对句子进行情感分析时只需要最后一层给一个输出即可，核心在于隐藏层的传递。

LSTM作为RNN的变种之一，通过引入门机制缓解了可能出现的梯度消失与梯度爆炸问题，其单元结构如图2.2所示。

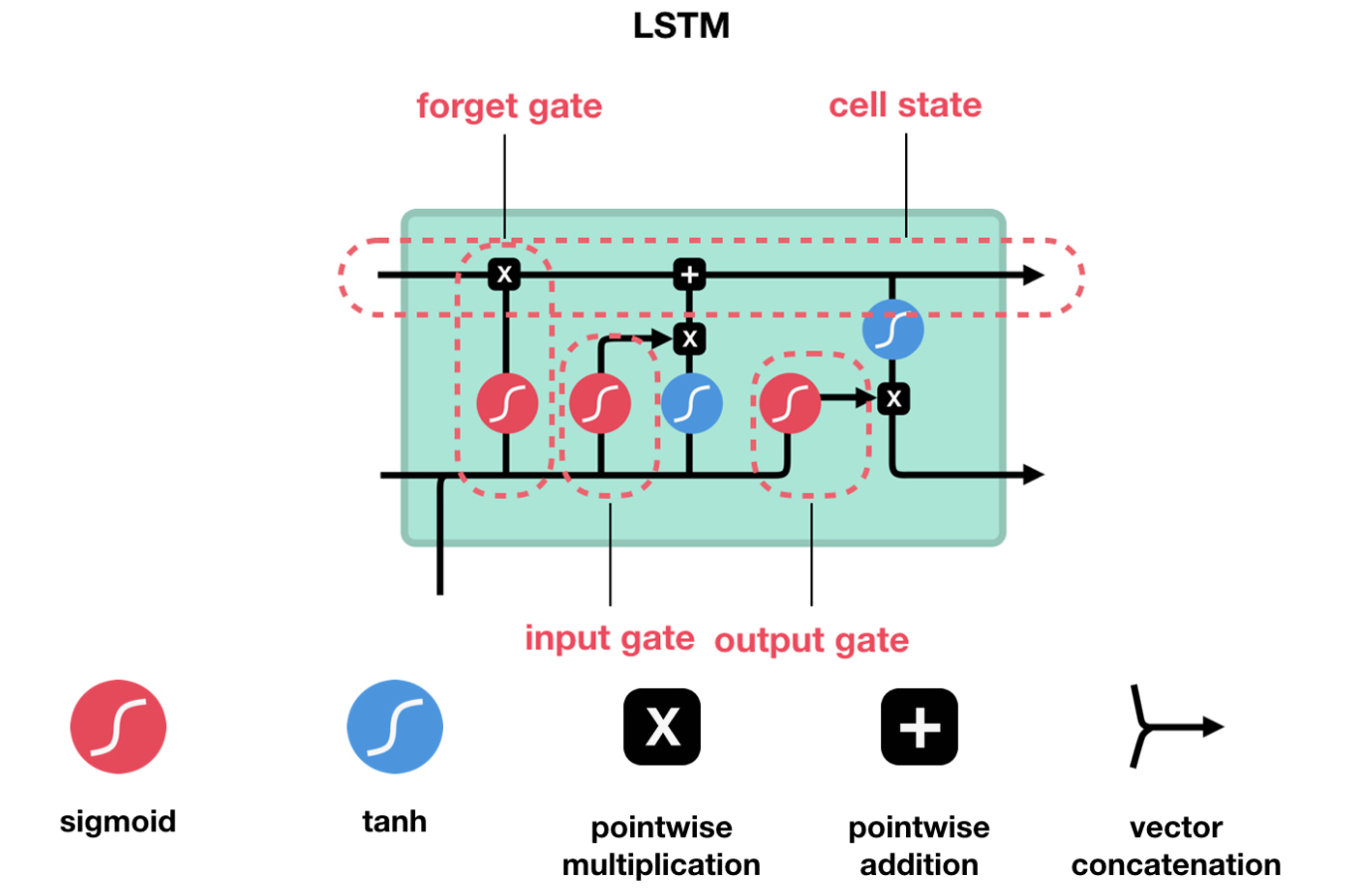


图 2-1 LSTM单元结构

## 3 系统实现

### 3.1 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。

### 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。

### 3.6 split.py

将数据集划分为训练集和测试集。

## 4 实验小结

本次实验是基于实验一的依次扩充，实现了对于中文实体的识别，大部分实验的代码老师都已经给出，所以实现起来没有技术上的压力。主要是看明白代码花费了一些时间，再就是对代码进行部分调整已达到优化的目的。

总体上实验不困难，加深了对于自然语言处理相关知识的掌握，对人工智能识别人类语言的过程有了浅层的认知。

# 附录 命名实体识别实现的源程序

## 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="gbk") as ifp:  
 file\_set(x\_train, y\_train, ifp, word\_num)  
  
 with open(VALID\_DATA, 'r', encoding="GBK") 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()  
  
  
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  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 model = torch.load('save/model\_epoch9.pkl', map\_location=torch.device('cpu'))  
 output = open('ner\_result.txt', 'w', encoding='gbk')  
  
 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)  
  
 with open('data/ner\_test.txt', 'r', encoding='gbk') as f:  
 line\_test = ''  
 for test in f:  
 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)

## 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\_data\_save.pkl', 'rb') as inp:  
 word2id = pickle.load(inp)  
 pickle.load(inp)  
 tag2id = pickle.load(inp)  
 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  
 )  
  
 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())

## 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')