

**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2005**

**学 号： U202015439**

**姓 名： 申智**

**指导教师： 姚德中**

**报告日期： 2023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 2](#_Toc137722770)

[1 问题描述 2](#_Toc137722771)

[1.1 实验目的 2](#_Toc137722772)

[1.2 实验任务 2](#_Toc137722773)

[1.2.1 基础任务 2](#_Toc137722774)

[1.3.2 选做任务 3](#_Toc137722775)

[2 模块实现 4](#_Toc137722776)

[2.1 1.data\_u\_ner.py 4](#_Toc137722777)

[2.2 dataloader.py 4](#_Toc137722778)

[2.3 model.py 4](#_Toc137722779)

[2.4 run.py 5](#_Toc137722780)

[2.5 infer.py 5](#_Toc137722781)

[2.6 format.py 5](#_Toc137722782)

[3 运行结果 6](#_Toc137722783)

[4 实验小结 7](#_Toc137722784)

[参考文献 8](#_Toc137722785)

[附录 命名实体识别实现的源程序 9](#_Toc137722786)

# 中文命名实体识别

## 1 问题描述

命名实体识别（Named Entity Recognition, NER），是指识别文本中具有特定意义的实体，主要包括人名、地名、机构名、专有名词等，以及时间、数量、货币、比例数值等文字。

NER是一种序列标注问题，因此数据的标注方式也遵照序列标注问题。命名实体识别不仅要找出实体的位置，还要对实体进行分类。位置和类别通过标签来表达，数据标注格式主要是BIO和BIOES两种。

BIO标注法中B表示实体开始，I表示实体内部，O表示实体外部；

BIOES标注法中B表示实体开始，I表示实体内部，E表示实体结束，S表示单个词形成实体，O表示实体外部。

### 1.1 实验目的

通过实验达到：⑴ 掌握序列标注、命名实体识别的理论基础及常规解决方案；⑵ 加深对循环神经网络、LSTM、条件随机场等模型基本原理的理解；⑶ 掌握基于TensorFlow等框架对循环神经网络、长短期记忆网络的实现方法。

### 1.2 实验任务

实验基础任务部分要求构造一个命名实体识别（NER）模型，除了基本的预测功能外，能够对测试集进行批量预测并将测试结果保存为文件。

#### 1.2.1 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。

#### 1.3.2 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：

1. 修改网络结构，例如引入BERT等预训练语言模型；
2. 调整、优化模型训练过程中的超参数。

（2）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。

（3）调整融合策略

## 2 模块实现

### 2.1 1.data\_u\_ner.py

处理包含已标记数据的NER\_train.txt文件。首先创建一个唯一标签的集合，并使用字典为它们分配ID。然后，定义了一个名为“getlist”的函数，该函数根据单词的长度将其转换为标签序列（如果单词只有一个字符，则标签将是'S'；如果单词长两个字符，则标签将是'B'和'E'；否则，标签将是'B'，后面跟着每个中间字符的'M'标签，最后以'E'标签结尾）。主要函数“handle\_data”读取TRAIN\_DATA文件并将其转换为单词ID序列和标签ID序列，这些序列被倾倒到指定的pickle文件SAVE\_PATH中。最后，该函数打印出x\_train和y\_train的第一个示例及其相应的单词和标签。

### 2.2 dataloader.py

定义了一个名为“Sentence”的类，该类继承了PyTorch的“Dataset”类。它需要两个参数x和y，这些参数分别是训练数据的输入序列和相应的标签序列。batch\_size参数指定了每个批次的大小，默认值为10。\_\_len\_\_函数返回数据集的样本数，而\_\_getitem\_\_函数返回指定索引的数据对（输入和标签）。collate\_fn函数用于在提取样本后将它们组合成一个批次的张量。在collate\_fn函数中，train\_data被按输入序列长度从大到小排序，然后分别计算出每个数据对的长度、输入序列、标签序列和掩码。通过使用pad\_sequence函数将所有序列填充到同一长度，并返回一个包含输入序列、标签序列、掩码和长度的元组。最后，读取保存在pickle文件中的数据并将其传递给DataLoader对象，以便在训练模型时使用迭代器进行批量处理。

### 2.3 model.py

定义了一个名为“CWS”的PyTorch模型类。它有四个参数vocab\_size、tag2id、embedding\_dim和hidden\_dim，分别表示词汇表大小、标签到ID的映射、词嵌入维度和LSTM隐层维度。使用torchcrf包中的CRF层来计算损失并进行推理。

在CWS类中，init\_hidden方法用于初始化LSTM的隐状态。\_get\_lstm\_features方法用于获取LSTM的输出特征，包括嵌入序列、mask、长度和隐状态。该方法首先将输入句子中的单词索引转换为嵌入向量，然后使用pack\_padded\_sequence函数将其填充到相同的长度。LSTM网络接收嵌入序列和初始隐藏状态，并返回一个由所有时间步的输出组成的张量。最后，通过FC层将LSTM的输出投影到softmax层，并返回得分。

forward方法用于计算模型损失。调用\_get\_lstm\_features方法获取LSTM特征，并将其传递给CRF层进行损失计算。

infer方法用于进行推理，即预测输入句子的标签序列。

### 2.4 run.py

命名实体识别模型的训练。首先使用argparse库获取参数，嵌入维度、学习率、最大迭代次数、批次大小和LSTM隐层维度等。然后，加载预处理好的数据集和标签字典，调用CWS类构建命名实体识别模型。接着，创建一个Adam优化器并定义DataLoader类来生成训练数据。最后，开始训练模型，并在训练时定期输出损失值和保存模型。

循环遍历训练数据，计算损失并执行反向传播更新参数。每100个步骤打印出平均损失值。每个epoch结束后将训练后的模型保存到磁盘上。

### 2.5 infer.py

使用训练模型进行命名实体识别。首先从保存的pickle文件中加载预训练模型。从另一个pickle文件中加载一些数据，包括单词到ID和标签到ID的字典，以及训练数据。从名为“NER\_test.txt”的文件中读取文本数据。对于每行文本，使用单词到ID字典将字符转换为ID，然后将其转换为PyTorch张量并将其移动到适当的设备。使用模型的`infer`方法生成预测，该方法将张量、注意力掩码和序列长度作为输入。最后输出结果。

### 2.6 format.py

去除“ner\_result.txt”中空白符开头的行，使格式符合最终要求。

## 3 运行结果

将输出结果用记事本打开，如图3-1。

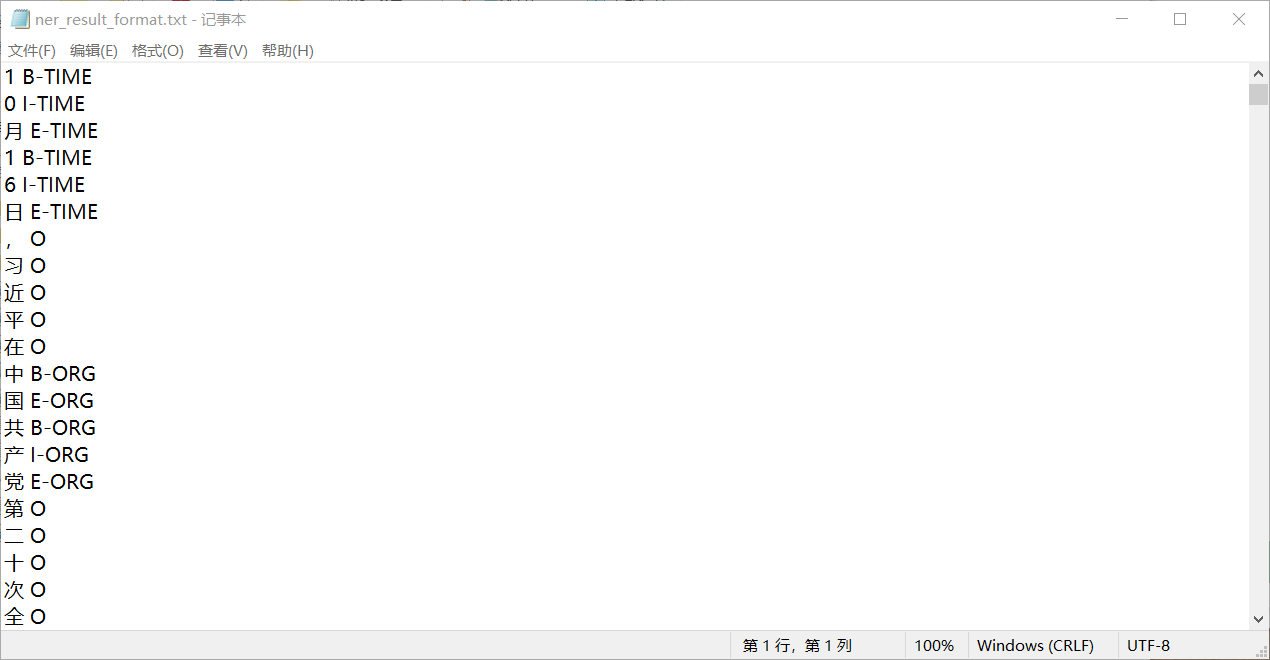


图3-1 输出结果

将结果提交到educoder平台，得到评估，如图3-2。

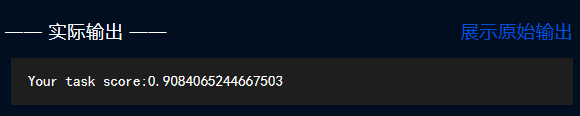


图3-2 评估成绩

## 4 实验小结

本次实验主要目的是通过基于Bi-LSTM+CRF模型的中文命名实体识别项目

在数据预处理方面，我采用了与实验一中文分词类似的方法，将原始中文文本数据进行了分词、词性标注、实体标注等处理，用于后续的训练和测试。在模型的输入层方面，将每个中文字符转化为对应的向量表示，并加入额外的特征信息（如词性、是否为数字或字母等），增强模型对实体边界的判断能力。在输出层方面，使用CRF模型来实现对实体序列的判定，并通过全局优化来达到更好的识别效果。

在模型的训练过程中，采用了Adam算法作为优化器，并结合交叉熵损失函数和正则化项，对模型进行了训练。同时，还采用了一些常见的技巧，如dropout、early stopping等来防止过拟合和提高模型的泛化能力。

在测试集上进行模型的评测后得到了较为良好的实验结果。同时，还对实验结果中的错误案例进行了分析，并提出了一些改进和优化的建议。

通过本次实验的实际操作，我更深入地了解了命名实体识别任务及其相关技术，这对我今后在自然语言处理领域的学习和应用具有重要意义。

# 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 命名实体识别实现的源程序

1. **1.data\_u\_ner.py**

import pickle

TRAIN\_DATA = "NER\_train.txt"

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

unique = set()

with open('ner\_train.txt', 'r',encoding='utf-8')as f:

for line in f:

try:

items = line.strip().split(' ')

if len(items) >= 2:

unique.update([items[1]])

except Exception as e:

print("Failed to process line:", line)

print("Error message:", str(e))

continue

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 = []

word\_num = 0

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

file\_set(x\_train, y\_train, 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)

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()

1. **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

1. **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)

1. **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=32)

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)

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

)

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())

1. **infer.py**

import torch

import pickle

from tqdm import tqdm

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

device = torch.device("cuda") if torch.cuda.is\_available() else torch.device("cpu")

model = torch.load('save/model\_epoch9.pkl',map\_location=device)

output = open('ner\_result.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)

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

lines = f.readlines()

for line in tqdm(lines):

text = line.strip()

x = [word2id.get(char,len(word2id)) for char in text]

x = torch.LongTensor(x).unsqueeze(0).to(device)

length = [len(text)]

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

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

for i in range(len(text)):

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

output.write("\n")

1. **format.py**

with open('ner\_result.txt', 'r', encoding='utf-8') as input\_file, open('ner\_result\_format.txt', 'w', encoding='utf-8') as output\_file:

# 逐行读取输入文件

for line in input\_file:

# 检查该行是否为空或开头是空白符

if not line.strip():

continue

# 如果该行不为空且开头不是空白符，则将其写入输出文件

output\_file.write(line)

print("已去除开头是空白符的行。\n")