華中科技大学

课 程 报 告

课程名称： 自然语言处理

专业班级： 物联网工程 2001

学　　号： U202015458

姓　　名： 周臻鹏

指导教师： 魏巍

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。

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。

1.1 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**。

2. 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。

1.2 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

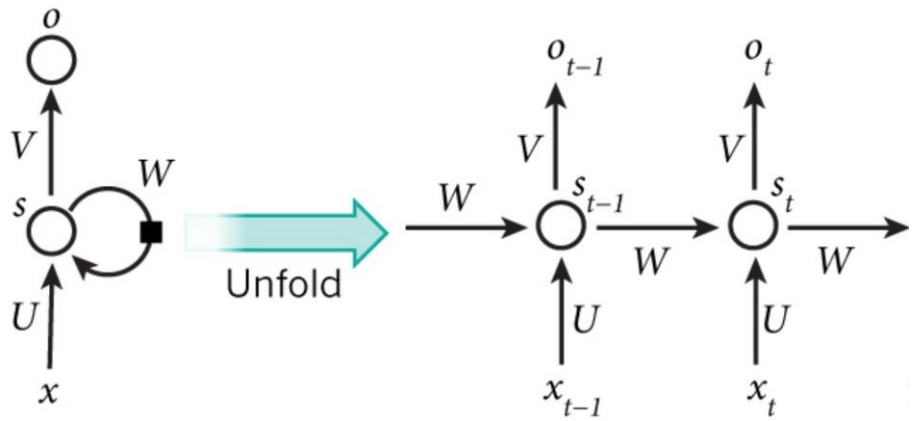
实验二提供的人民日报语料与分词所采用的语料并不一定是同分布的，你可以自行搜集更为合适的数据集进行训练。

（3）调整融合策略

2 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：



x_t是第t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；

s_t是第t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，$s_t = f(Ux_t + Ws_{t-1})$,f通常是一个非线性的激活函数，比如 tanh 或 ReLU。由于每一层的S_t都会向后一直传递，所以理论上s_t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）。

o_t是第 t 层的输出，比如我有预测下一个词是什么时，o_t就是一个长度为V的向量，V是所有词的总数，o_t[i]表示下一个词是w_i的概率。最后用 softmax 对这些概率进行归一化$o_t = \text{softmax}(Vs_t)$

每一层的参数U,W,V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。

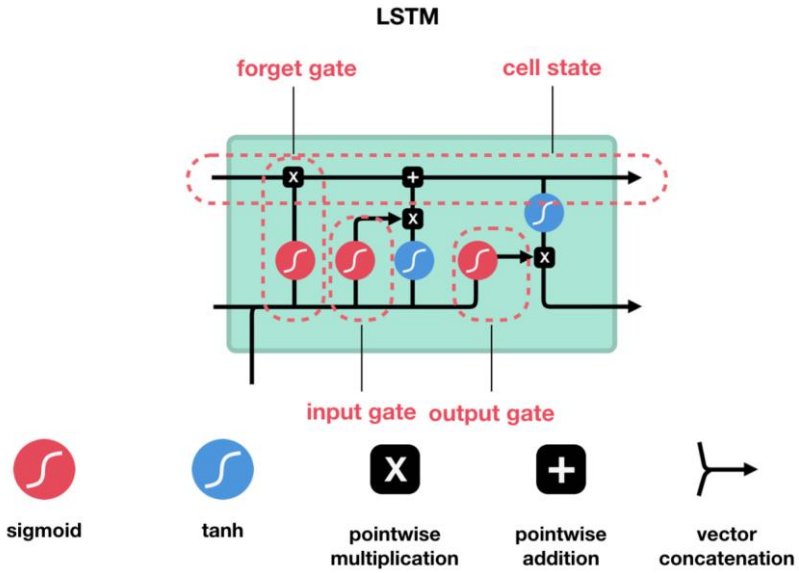


图 2-1 LSTM 单元结构

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。

```
def handle_data():
    '''
    处理数据，并保存至 savepath
    :return:
    '''
    outp = open(SAVE_PATH, 'wb')

    x_train = []
    y_train = []
    x_valid = []
    y_valid = []

    wordnum = 0
```

```
# with open(TRAIN_DATA, 'r', encoding="utf-8") as ifp:
with open(TRAIN_DATA, 'r') as ifp:
    line_x = []
    line_y = []
    for line in ifp:
        line = line.strip()
        if not line:
            x_train.append(line_x)
            y_train.append(line_y)
            line_x = []
            line_y = []
            continue
        line = line.split(' ')
        if line[0] in id2word:
            line_x.append(word2id[line[0]])
        else:
            id2word.append(line[0])
            word2id[line[0]] = wordnum
            line_x.append(wordnum)
            wordnum = wordnum + 1
        line_y.append(tag2id[line[1]])

with open(VALID_DATA, 'r') as ifp:
    line_x = []
    line_y = []
    for line in ifp:
        line = line.strip()
        if not line:
            x_valid.append(line_x)
            y_valid.append(line_y)
            line_x = []
            line_y = []
            continue
        line = line.split(' ')
        if line[0] in id2word:
            line_x.append(word2id[line[0]])
        else:
            id2word.append(line[0])
            word2id[line[0]] = wordnum
            line_x.append(wordnum)
            wordnum = wordnum + 1
        line_y.append(tag2id[line[1]])
```

```
        print(x_train[0])
        print([id2word[i] for i in x_train[0]])
        print(y_train[0])
        print([id2tag[i] for i in y_train[0]])
        # x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.1,
random_state=43)

        pickle.dump(word2id, outp)
        pickle.dump(id2word, outp)
        pickle.dump(tag2id, outp)
        pickle.dump(id2tag, outp)
        pickle.dump(x_train, outp)
        pickle.dump(y_train, outp)
        pickle.dump(x_valid, outp)
        pickle.dump(y_valid, outp)

        outp.close()
```

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。

```
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.tagset_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.tagset_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)
```

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。

```
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 = []

    # TODO: valid
    # FIll the code by yourself.

    path_name = "./save/model_epoch" + str(epoch) + ".pkl"
    torch.save(model, path_name)
    logging.info("model has been saved in  %s" % path_name)
```

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,更多在于自行的理解与感悟。在阅读代码的过程中,我对于中文实体识别的任务有了全新的认识,对于 BiLSTM 的结构原理和实际应用有了更深刻的认识,再就是对代码进行部分调整已达到优化的目的。

总体上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。

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

#spilt.py

```
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 = []

# print(len(corpus)) # 44011

import random
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')
```

#data_u_ner.py

```
import codecs
from sklearn.model_selection import train_test_split
import pickle

INPUT_DATA = "train.txt"
TRAIN_DATA = "ner_train.txt"
VALID_DATA = "ner_valid.txt"
SAVE_PATH = "./ner_datasave.pkl"

# create id2tag
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

# 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
    '''
    outpout_str = []
    if len(input_str) == 1:
        outpout_str.append(tag2id['S'])
    elif len(input_str) == 2:
        outpout_str = [tag2id['B'], tag2id['E']]
    else:
        M_num = len(input_str) - 2
        M_list = [tag2id['M']] * M_num
```

```
            outpout_str.append(tag2id['B'])
            outpout_str.extend(M_list)
            outpout_str.append(tag2id['E'])
        return outpout_str


def handle_data():
    '''
    处理数据，并保存至 savepath
    :return:
    '''
    outp = open(SAVE_PATH, 'wb')

    x_train = []
    y_train = []
    x_valid = []
    y_valid = []

    wordnum = 0
    # with open(TRAIN_DATA, 'r', encoding="utf-8") as ifp:
    with open(TRAIN_DATA, 'r') as ifp:
        line_x = []
        line_y = []
        for line in ifp:
            line = line.strip()
            if not line:
                x_train.append(line_x)
                y_train.append(line_y)
                line_x = []
                line_y = []
                continue
            line = line.split(' ')
            if line[0] in id2word:
                line_x.append(word2id[line[0]])
            else:
                id2word.append(line[0])
                word2id[line[0]] = wordnum
                line_x.append(wordnum)
                wordnum = wordnum + 1
            line_y.append(tag2id[line[1]])

    with open(VALID_DATA, 'r') as ifp:
        line_x = []
        line_y = []
```

```
        for line in ifp:
            line = line.strip()
            if not line:
                x_valid.append(line_x)
                y_valid.append(line_y)
                line_x = []
                line_y = []
                continue
            line = line.split(' ')
            if line[0] in id2word:
                line_x.append(word2id[line[0]])
            else:
                id2word.append(line[0])
                word2id[line[0]] = wordnum
                line_x.append(wordnum)
                wordnum = wordnum + 1
            line_y.append(tag2id[line[1]])


    print(x_train[0])
    print([id2word[i] for i in x_train[0]])
    print(y_train[0])
    print([id2tag[i] for i in y_train[0]])
    # x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.1, random_state=43)

    pickle.dump(word2id, outp)
    pickle.dump(id2word, outp)
    pickle.dump(tag2id, outp)
    pickle.dump(id2tag, outp)
    pickle.dump(x_train, outp)
    pickle.dump(y_train, outp)
    pickle.dump(x_valid, outp)
    pickle.dump(y_valid, outp)

    outp.close()


if __name__ == "__main__":
    handle_data()
```

#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.tagset_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.tagset_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_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 = []

        # TODO: valid
        # FIll the code by yourself.

        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())
```

#infer.py

```
import torch
import pickle

if __name__ == '__main__':
    model = torch.load('save/model.pkl', map_location=torch.device('cpu'))
    output = open('ner_result.txt', 'w', encoding='utf-8')

    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)

    with open('data/ner_test.txt', 'r', encoding='utf-8') 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(test[i], end='', file=output)
                    # if id2tag[predict[i]] in ['E', 'S']:
                    #     print(' ', end='', file=output)
                print(file=output)

                line_test = ''

            else:
                test = test.split(' ')
                line_test += test[0]
```

#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]
        mask = [torch.ones(l, dtype=torch.uint8) for l 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)
        mask = pad_sequence(mask, batch_first=True, padding_value=0)
```

```
        return data_x, data_y, mask, data_length


if __name__ == '__main__':
    # test
    with open('../data/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)

    train_dataloader = DataLoader(Sentence(x_train, y_train), batch_size=10, shuffle=True, collate_fn=Sentence.collate_fn)

    for input, label, mask, length in train_dataloader:
        print(input, label)
        break
```