華中科技大学

课 程 实 验 报 告

课程名称： 自然语言处理实验

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 10 日

计算机科学与技术学院

目 录

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。

实验任务包括基础任务与选做任务。基础任务中需实现基于词典和基于统计的中文分词算法，完成后可获得实验课程基础分。选做任务中需要对基础任务中的分词器进行优化，选做部分的分数通过分词器在测试集上的表现决定。最终提交的实验报告中应包括基础任务完成情况与选做任务中采取的优化措施。

1.1 基础任务

I 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。

II 实现基于统计的分词算法

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：

修改网络结构，例如引入 BERT 等预训练语言模型；

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；

构造合适的词典集（可扩充+人工整理）；

实现新词发现（登录）功能，识别测试集中的新词（未登录词）；

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 基础模块

2.1 基于词典的分词算法模块

2.1.1 正向最大匹配法

1. 从左向右取句子的前 m 个字作为匹配字段(m 为词典中最长词的长度)
2. 查找词典进行匹配
3. 若匹配成功，则将该字段作为一个词切分出去
4. 若匹配不成功，则将该字段最后一个字去掉，剩下的字作为新匹配字段，再次进行匹配
5. 重复上述过程，直到切分所有词为止

逆向最大匹配法则从待分词句子的末端开始，也就是从右向左开始匹配扫描，每次取末端 m 个字作为匹配字段，匹配失败，则去掉匹配字段前面的一个字，继续匹配。双向最大匹配法将正向最大匹配法得到的分词结果和逆向最大匹配法得到的结果进行比较，选取更为合适的作为结果。这里给出一种选择方式：

1. 如果正反向分词结果词数不同，取分词数量少的那个
2. 如果分词结果词数相同：
    a) 分词结果相同，没有歧义，返回任意一个
    b) 分词结果不同，返回其中单字数量较少的那个

2.1.2 逆向最大匹配法

算法流程为：

1、输入最大词长 maxWordLength，字典 wordDict，待分句子。

2、从待分句子的末尾开始向前截取长度为 maxWordLength 的子句，进行分词。

3、对一个子句的分词过程为，首先判断子句是否在字典中，若在，则保存这个子句，并从原句中删除这个子句，转到 2。

若不在，则判断子句长度是否为 1，若为 1，则将单字保存，从原句中删除单字，转到 2。若不为 1，则将子句中最右边的一个字删除，形成新的子句，转到 3。

2.1.1 双向最大匹配法

双向最大匹配法是建立在正向最大匹配法和逆向最大匹配法之上的，是对两者结果的比较，选出更优的那一个作为结果。

双向最大匹配分词是综合了前两者的算法。先根据标点对文档进行粗切分，把文档分解成若干个句子，然后再对这些句子用正向最大匹配法和逆向最大匹配法进行扫描切分。如果两种分词方法得到的匹配结果相同，则认为分词正确，否则，按最小集处理。准确的结果往往是词数切分较少的那种。

2.2 基于统计学习的分词算法模块

2.2.1 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。

2.2.2 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。

2.2.3 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。

2.2.4 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。

2.2.5 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 系统实现

3.1 基于词典的分词算法

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
```

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
```

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
```

3.2 基于统计学习的分词算法

使用 BERT 模型进行预训练

```
    def get_masked_lm_output(bert_config, input_tensor, output_weights,
positions,
                             label_ids, label_weights):
        """
        Get loss and log probs for the masked LM.
        :param bert_config:
        :param input_tensor: bert 模型 encoder 的输出, [batch_size, seq_len,
```

```
hidden_size]
      :param output_weights: bert 的 embedding lookup table , [vocab_size,
hidden_size]
      :param positions: 被 mask 的位置 index， [batch_size,
max_predictions_per_seq]
      :param label_ids: 被 mask 的 token id, [batch_size,
max_predictions_per_seq]
      :param label_weights: max_predictions_per_seq 个位置中，被 mask 的标记为
1，否则为 0.
       [batch_size, max_predictions_per_seq]
      :return:
      """
      input_tensor = gather_indexes(input_tensor, positions)

      with tf.variable_scope("cls/predictions"):
        # We apply one more non-linear transformation before the output layer.
        # This matrix is not used after pre-training.
        with tf.variable_scope("transform"):
          input_tensor = tf.layers.dense(
              input_tensor,
              units=bert_config.hidden_size,
              activation=modeling.get_activation(bert_config.hidden_act),
              kernel_initializer=modeling.create_initializer(
                  bert_config.initializer_range))
          input_tensor = modeling.layer_norm(input_tensor)

        # The output weights are the same as the input embeddings, but there is
        # an output-only bias for each token.
        output_bias = tf.get_variable(
            "output_bias",
```

```
            shape=[bert_config.vocab_size],
            initializer=tf.zeros_initializer())
        logits = tf.matmul(input_tensor, output_weights, transpose_b=True)
        logits = tf.nn.bias_add(logits, output_bias)
        log_probs = tf.nn.log_softmax(logits, axis=-1)

        label_ids = tf.reshape(label_ids, [-1])
        label_weights = tf.reshape(label_weights, [-1])

        one_hot_labels = tf.one_hot(
            label_ids, depth=bert_config.vocab_size, dtype=tf.float32)

        # The `positions` tensor might be zero-padded (if the sequence is too
        # short to have the maximum number of predictions). The `label_weights`
        # tensor has a value of 1.0 for every real prediction and 0.0 for the
        # padding predictions.
        per_example_loss = -tf.reduce_sum(log_probs * one_hot_labels, axis=[-
1])
        numerator = tf.reduce_sum(label_weights * per_example_loss)
        denominator = tf.reduce_sum(label_weights) + 1e-5
        loss = numerator / denominator

      return (loss, per_example_loss, log_probs)
```

使用 BiLSTM 训练模型

```
use_cuda = args.cuda and torch.cuda.is_available()


    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)


    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 = []
```

最终训练结果如下

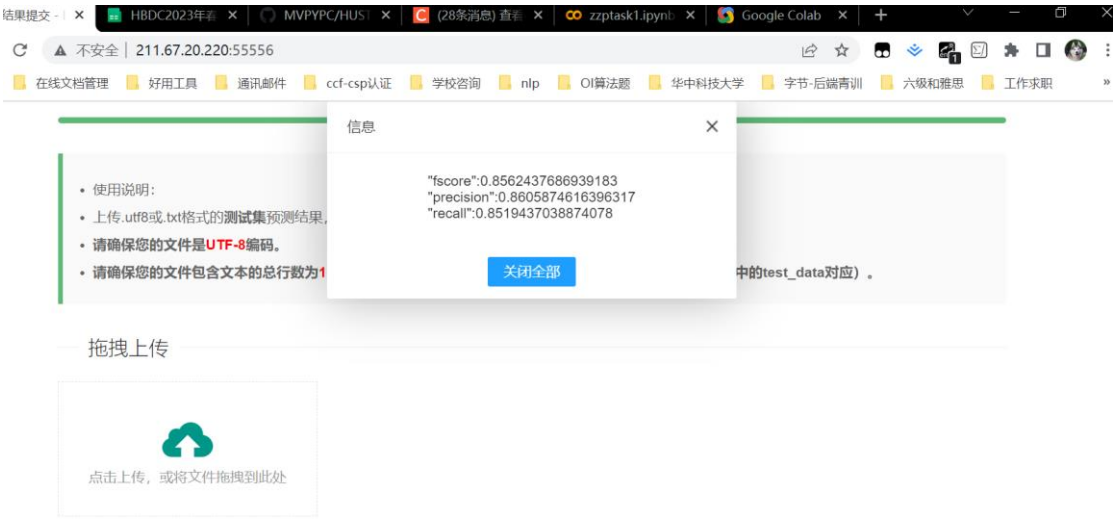

图 3.1 基于统计学习的分词算法模型评估结果

4 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，我在原有基础上加入了 Bert 预训练模型,融合了词向量的应用,效果比原有基础上提升了一点点性能,不算特别明显,值得思考用更高效的模型加以训练。

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 中文分词实现的源程序

#data_u.py

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

INPUT_DATA = "train.txt"
SAVE_PATH = "./datasave.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
    '''
    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:
    '''
    x_data = []
    y_data = []
    wordnum = 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]] = wordnum
                    line_x.append(wordnum)
                    wordnum = wordnum + 1
            x_data.append(line_x)

            lineArr = line.split()
            line_y = []
            for item in lineArr:
                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 outp:
        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_test, outp)
        pickle.dump(y_test, outp)
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]
        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
```

#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(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/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 = []

        # 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("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())
```

#infer.py

```
import torch
import pickle

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

    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)

    with open('data/test.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)
```