al



**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201801**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2018年 月 日**

**计算机科学与技术学院**

目 录

[1 中文分词 2](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 基础模块 2](#_Toc458159882)

[1.3 系统实现 2](#_Toc458159883)

[1.4 实验小结 2](#_Toc458159884)

[2 中文命名实体识别 2](#_Toc458159885)

[2.1 问题描述 2](#_Toc458159886)

[2.2 基础模块 2](#_Toc458159887)

[2.3 系统实现 2](#_Toc458159888)

[2.4 实验小结 2](#_Toc458159889)

[参考文献 2](#_Toc458159900)

[附录A 中文分词实现的源程序 2](#_Toc458159901)

[附录B 中文命名实体识别的源程序 2](#_Toc458159902)

# 1 中文分词实现

## 1.1 问题描述

## 1.1.1 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。分词就是将连续的字序列按照一定的规范重新组合成词序列的过程；在英文中，单词之间是以空格作为自然分界符，汉语中词没有一个形式上的分界符。中文分词是很多自然语言处理系统中的基础模块和首要环节。

## 1.1.2 问题难点

正因为缺乏形式上的分界符，导致我们对词的认定会出现很大的偏差。1996年Sproat[1]等通过对6个母语为汉语的人进行调研，让这6人对同一篇中文文本进行人工切分，文本包括100个句子，最后统计认同率，AG和GR分别代表最短匹配和最长匹配的分词结果，ST代表论文提出的分词算法结果，M开头的是大陆人，T开头的为台湾人，结果如图1-1：

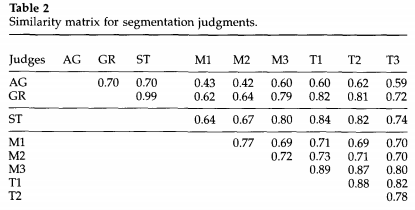


图1-1 中文分词认同率

结果体现了中文分词问题的困难性之一，分词标准问题，不同普通人对词语的认识有差异，导致对不同人分词结果的认同度只有八成左右。即便是中文语言专家也没有达成关于分词标准的共识。国家标准委曾发布过信息处理用现代汉语分词规范GB13715[2]作为分词建议，但因该建议弹性太大，未被广泛采用。其他的分词标准有北大发布的北大语料库加工规范[3]等。

中文分词的其他难点还包括歧义问题和新词问题，歧义问题可以分为组合型歧义、交集型歧义和真歧义三种类型。

1. 组合型歧义：因为分词粒度的不同，一个词可能由多个更细粒度的词组成。例：“秦皇岛钢铁工业集团”可以细分为“秦皇岛/钢铁/工业/集团”。
2. 交集型歧义：句子中间的一个字可以和左右两边的字组成不同的词。例：“天和服装厂”种“和”字可以组成“天和”，“和服”两个词。
3. 真歧义：句子本身就具有歧义。例：“美国会通过对台售武法案”，既可以切分成“美国/会/通过对台售武法案”，又可以切分成“美/国会/通过对台售武法案”。

新词问题指句子中包含未被词典收录的词，新词问题的解决依赖人们对汉语语言结构和分词算法的更深层次研究。

## 1.1.3 中文分词算法

目前的中文分词算法主要有：

1. 基于词表的算法，如：正向最大匹配，逆向最大匹配和双向最大匹配等
2. 基于统计模型的算法，如：N-gram语言模型分词等
3. 基于序列标注的分词算法，如：基于HMM的分词，基于深度学习的端到端分词等

## 1.2 基础模块

## 1.2.1 系统模块划分

系统主要分为数据预处理，模型和结果后处理三个模块，系统各模块组成和层次结构如图1-2。每个模块具体设计将在下一节给出。

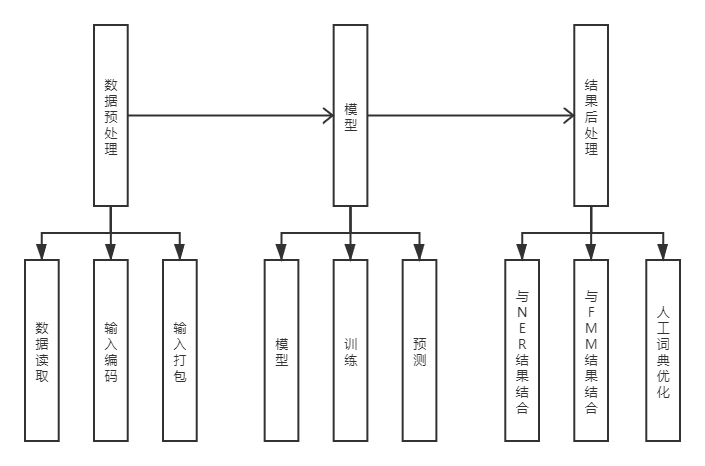


图1-2 系统层次结构图

## 1.2.2 系统模块设计

系统主要分为数据预处理，模型和结果后处理三个部分，下面介绍每个模块的详细设计。

数据预处理模块主要负责原始数据的读取，对原始数据的编码和编码结果的打包三个部分。原始训练数据以分词结果的形式给出，文件中每行是一个句子，句子中的各个词用空格隔开。数据读取部分把训练数据从文件中读出，为每个字生成相应的BMES标签。编码部分将以文字表示的句子中的每个字替换成其在字典中对应的编码，方便之后模型的处理，这一步为了配合bert系模型的设计，会在每个句子的开头和结尾加上[CLS]和[SEP]两个标志，未出现在字典中的字将被替换为[UNK]，同时将所有句子padding成相同的长度，方便后续处理，padding的部分使用符号[PAD]填充，为了保证后续训练和预测时能正确的忽略特殊符号，同时为每个句子生成input mask和output mask用于输入和输出时的遮罩。之后输入打包部分将编码结果保存到文件中，方便之后训练直接读取。

模型模块主要包含神经网络结构的定义，模型训练和模型预测三个部分。在课程提供的代码框架中，主要使用的Bi-LSTM+CRF的模型结构，在此之上，本实验将编码层替换成了transformer系的预训练模型，通过attention机制获得语义信息更加丰富的句子编码，并得到了更好的分词效果。在介绍本实验的神经网络结构前，先简单介绍一下几种基于transformer的预训练模型：

1. BERT

BERT（BERT:Pre-training of Deep Bidirectional Transformers）[4]是Google在2018年发表的一个语言预训练模型，模型一经发布，就在SQuAD测试中全面超越人类指标，并且在多种NLP测试上更新最好成绩。BERT模型在transformer的基础上引入了MLM和NSP用于构造双向语言模型，MLM(Masked Language Model)随机选择语料中15%的单词，使用[MASK]标记替换这个单词，要求模型正确预测被替换掉的单词。为了避免模型对[MASK]这个标记产生依赖，被选中的词中只有80%是被替换为[MASK]，10%被替换成一个其他单词，还有10%则没有做任何替换。NSP(Next Sentence Prediction)从语料中选择一些真正顺序相连的两个句子，再选取一些分散的两个句子，并把它们前后连接在一起，要求模型预测这些被选出来的句子对是否真实前后相连。引入这两个任务的目的是达到论文标题中的“Deep Bidirectional”，MLM让模型学习到了整个句子当中的前后文信息，而NSP让模型关注句子级别的上下文联系。

1. ERNIE

ERNIE(ERNIE: Enhanced Language Representation with Informative Entities)[5]是百度推出的语言预训练模型，在16种中英文任务上超越了BERT，ERNIE对于中文做了相应的优化，在中文任务上的表现更好。ERNIE相对于BERT主要做了以下几点改进：分别构建了词级别，句级别和语义级别的预训练任务，让模型在不同的预训练任务上连续训练，学习更多的语料信息，在模型输入上增加了一个task embedding用于表示不同的预训练任务；在词级别的训练任务上，ERNIE把BERT字级别的mask更换为了词组级别的mask和实体级别的mask来引入更多的知识，增加了一个判断单词是否大写的任务来学习大写词语特有的特征，加入了一个预测词语是否会在段落中出现的任务，用于增强模型获取文章关键词的能力；在句级别上，引入了一个重新组合段落中被拆散的句子的任务，和一个判断两个句子间距离的任务，让模型学习到段落中句子间的联系；在语义级别上，ERNIE通过判断句对间的修辞关系和基于百度日志的IR Relevance Task来获取数据中的语义信息。

1. ALBERT

ALBERT(ALBERT: A Lite BERT for Self-supervised Learning of Language Representations)[6]是Google推出的基于BERT的轻量语言预训练模型，ALBERT在BERT的基础上优化了模型结构，在BERT中，模型的embedding size E和transformer的hidden size H是完全相等的，但是实际上embedding中保存的信息是不依赖于当前上下文的，独立的词信息，而transformer中会保存上下文信息，理论上H应远大于E，将E和H设为同样的值会增加模型参数大小，所以ALBERT使用了较小的E值。同时ALBERT使用了跨层参数共享，稳定了网络参数。最后ALBERT替换掉了在BERT中作用较小的NSP任务，转换成了预测两个句子顺序是否颠倒的SOP(Sentence Order Prediction)任务，实验表明新任务有一个点的提升。ALBERT因为更轻量的设计，在相同的训练时间下，可以获得比BERT更好的结果，降低了使用BERT的成本，也是本次实验主要考虑的模型之一。

1. RoBERTa

RoBERTa(RoBERTa: A Robustly Optimized BERT Pretraining Approach)[7]是一个基于BERT的语言预训练模型，在BERT的基础上进行了部分调整，主要包括，将BERT的静态mask替换成动态mask，即在每次训练输入前再随机生成mask，而非在准备训练集时同时确定mask，避免了同样数据的重复输入，其次是抛弃了NSP任务获得了更好的性能。本次实验中使用的RoBERTa-wwm-ext还使用了哈工大讯飞实验室提供的wwm(whole word mask)版本模型，由于中文不像英文天然分词，所以BERT在中文语料上采用了字级别的编码，这样会忽略掉词级别的语义信息，wwm在对句子进行分词的基础上，将每个词整体进行mask，更加适应中文的语言结构。

1. XLNet

XLNet(XLNet: Generalized Autoregressive Pretraining for Language Understanding)[8]是一个基于transformer的预训练模型，在训练方式上，XLNet尝试在传统的从左到右的自回归语言模型的结构上引入自编码语言模型的特点，让模型可以同时看到上下文信息的同时，保持从左到右的训练方式，在结构同样是从左到右的一些语言任务上获得更好的表现。XLNet采用的方法是PLM(Permutation Language Model)，即排列语言模型，其中排列的意思是，固定要预测单词的位置，打乱句子中其他位置的单词，使得部分在预测单词下文的单词移动到预测单词的上文中，这样保留了从左到右预测结构的同时又获得了单词下文的信息。通过采用和BERT不同的策略，XLNet在一些从左到右的语言生成任务中取得了更好的效果。

1. GPT-2

GPT-2[9]是一个基于transformer的语言预训练模型，相比于BERT，GPT-2只使用了transformer的decoder，并且使用了更大的模型参数和更多的预训练数据，模型有15亿参数，训练数据集达40GB，包含八百万篇高质量文档。最近推出的GPT-3更是达到了1700亿参数的恐怖规模，训练语料45TB。在语言生成任务上达到了恐怖的高水平，可以说完全掌握了英语语法。但是由于参数量过大，本实验完全没有条件尝试。

结果后处理模块以模型输出的标签序列，NER模型输出的标签序列和人工词典输出的标签序列为输入，按照严格粗粒度优先的规则合并标签序列，作为最后的分词标签。

## 1.3 系统实现

## 1.3.1 系统模块实现

系统在实验提供的框架基础上引入了语言预训练模型作为编码层，配以相应的数据预处理，并进行了简单的后处理。中文分词任务可以转换为一个序列标注任务，即为句子中的每一个字分配一个“BMES”之中的标签，其中B代表一个词的开始，M代表一个词的中间部分，E代表一个词的结束，S则代表一个单独成词的字。整体实验按照序列标注的思想实现。

数据预处理模块中的handle\_data函数处理所有训练数据，首先创建input\_mask\_l，input\_id\_l等作为保存处理结果的列表，之后加载预训练模型的tokenizer，将输入句子转换成id，之后加入特殊符号[CLS]和[SEP]，并根据max length进行相应的填充或者剪裁，然后使用get\_list函数得到句子的标签序列。作为输入的标签，并生成一个output\_mask用于屏蔽掉填充部分的标签。最后将处理好的训练数据保存到datasave.pkl中，方便后续直接读取。

在训练数据集的选取上，出去实验提供的训练数据，还对其进行了扩充，增加了PKU训练集[10]中的训练数据，共计11000余条。

模型模块主要分为预训练编码器，LSTM，CRF三部分，其中预训练编码器加载预训练模型，读取input\_id，并输出编码后的句子，LSTM和CRF接受编码后的句子，序列标注知识的学习，最后CRF输出一个预测的标签序列。训练过程首先读取预处理后的训练数据，将训练数据分为训练集和验证集，数据量为9:1，并分别打包成batch为单位的包，每个epoch结束后进行一次验证集的验证。预测部分则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之后输入加载好的训练模型，得到输出后将标签序列保存到cws\_result.txt中，等待结果后处理。

预训练模型部分，本实验选用了多种预训练模型，对比其效果，包括bert-base，albert-tiny，albert-base，roberta-wwm-ext等，均来自开源网站Hugging Face[11]。

结果后处理阶段读取分词结果和实体识别结果的标签序列，按照严格粗粒度优先的策略合并结果，即除非词的数量有所减少，否则不改变分词结果。最后按照得到的标签序列，为测试文件进行分割。同时引入了基于词典的分词结果作参考，基于词典的分词主要使用了FMM,RMM和BMM三种分割方法，结果同样合并到最终结果中。

## 1.3.2 实验结果

本次实验中尝试了不同的预训练模型和超参数，最后得到的实验结果如表3-1：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模型 | F1 | LR | BS | EP |
| bert-base | 0.88 | 2e-5 | 12 | 3 |
| albert-tiny | 0.87 | 2e-5 | 32 | 10 |
| albert-base | 0.93 | 5e-5 | 64 | 15 |
| roberta-wwm-ext | 0.92 | 5e-5 | 24 | 5 |

表3-1 实验结果

其中F1为实验结果的F1-score，LR 为模型的学习率，BS为训练的batch size， EP为得到最优结果的epoch，由于算力的限制，没办法对所有模型进行彻底的训练，例如bert-base模型由于训练时间太长只能进行三个epoch所以得到的结果并不能代表模型本身的效果，因此之后选择了参数较小的albert-tiny，之后训练的结果表明albert-tiny因为模型太小，很难学习完全，在10个epoch的时候loss就不再下降但是模型本身仍然欠拟合，于是选用了稍大的albert-base在三十多个小时的训练后得到了最佳的效果，之后选用参数更大的roberta-wwm-ext，但是仍然受限于算力没能训练完全，根据训练的趋势可以期待roberta有相比于albert更好的效果。

结果后处理部分在实际实验的表现并不明显，在严格粗粒度优先的策略下，NER结果甚至没有一条被采用到最后结果中，而降低合并门槛后，NER结果对整体结果做了负优化。通过观察发现NER模型对于MISC类实体的识别率较低，且整体来讲NER结果的粒度要更细，在实体嵌套上的表现也不够好。所以最后并没有采用后处理得到的结果，而是直接提交了模型输出的结果。

这里要感谢ACM班师宇哲同学提供的通过贝叶斯优化得到的最佳学习率，因为算力有限没法进行全面的超参数优化，师宇哲同学提供的学习率参数大大减少了本次实验的工作量。

## 1.4 实验小结

本次实验在提供的框架基础上引入了语言预训练模型作为编码层，在获得了更好的句子表征之后，中文分词的结果相比于baseline有了将近十个点的提升。可以看出在NLP研究的领域，语言预训练模型的出现是一次巨大的革新，在本次实验中并有对语言预训练模型做一个完整的fine-tuning，如果算力足够，针对下游任务和预料做过fine-tuning的模型将会有更好的表现。对于NLP领域今后的发展来说，更好，更轻，更可理解的语言预训练模型应该是一个重要的方向。本次实验进行了一个完整的针对某个NLP任务的数据分析处理，模型搭建，调参，模型优化的过程，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。

# 2 中文命名实体识别实现

## 2.1 问题描述

## 2.2 基础模块

## 2.3 系统实现

## 2.4 实验小结

# 参考文献

1. Sproat, R. W., Shih, C., Gale, W. & Chang, N., 1996. A Stochastic Finite-State Word-Segmentation Algorithm for Chinese. *Computational Linguistics,* 卷 22, p. 377–404.
2. 全国信息技术标准化技术委员会, 1993. *信息处理用现代汉语分词规范* 北京.
3. Yu, S., Duan, H. & Wu, Y., 2018. *Corpus of Multi-level Processing for Modern Chinese.* Peking University Open Research Data Platform.
4. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K., 2018. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.*
5. Zhang, Z. et al, 2019. *ERNIE: Enhanced Language Representation with Informative Entities.*
6. Lan, Z. et al, 2019. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations.*
7. Liu, Y. et al, 2019. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.*
8. Yang, Z. et al, 2019. *XLNet: Generalized Autoregressive Pretraining for Language Understanding.*
9. Radford, A. et al, 2019. *Language Models are Unsupervised Multitask Learners.*
10. Emerson, T., 2005. *The Second International Chinese Word Segmentation Bakeoff.*
11. Anon., *Hugging Face. https://huggingface.co/*

# 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

import torch

from transformers import AlbertModel, BertTokenizer

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 = []

tokenizer = BertTokenizer.from\_pretrained("hfl/chinese-roberta-wwm-ext")

def get\_list(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savepath

:return:

"""

input\_ids\_l = []

input\_mask\_l = []

label\_l = []

output\_mask\_l = []

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

words = line.split()

sent = ''.join(words)

tokens = [i for i in sent]

label = []

for item in words:

label.extend(get\_list(item))

if len(tokens) > 512 - 2:

tokens = tokens[: (512 - 2)]

label = label[: (512 - 2)]

tokens\_cs = '[CLS]' + ' '.join(tokens) + '[SEP]'

tokenized\_text = tokenizer.tokenize(tokens\_cs)

input\_ids = tokenizer.convert\_tokens\_to\_ids(tokenized\_text)

input\_mask = [1] \* len(input\_ids)

label = [3] + label + [3]

while len(input\_ids) < 512:

input\_ids.append(0)

input\_mask.append(0)

while len(label) < 512:

label.append(-1)

output\_mask = [1] \* len(tokens)

output\_mask = [1] + output\_mask + [1]

while len(output\_mask) < 512:

output\_mask.append(0)

assert len(input\_ids) == 512

assert len(input\_mask) == 512

assert len(label) == 512

assert len(output\_mask) == 512

input\_ids\_l.append(input\_ids)

input\_mask\_l.append(input\_mask)

label\_l.append(label)

output\_mask\_l.append(output\_mask)

print(tokenizer.convert\_ids\_to\_tokens(input\_ids\_l[0]))

print(input\_ids\_l[0])

print(input\_mask\_l[0])

print(label\_l[0])

print(output\_mask\_l[0])

with open(SAVE\_PATH, 'wb') as outp:

pickle.dump(tag2id, outp)

pickle.dump(id2tag, outp)

pickle.dump(input\_ids\_l, outp)

pickle.dump(label\_l, outp)

pickle.dump(input\_mask\_l, outp)

pickle.dump(output\_mask\_l, outp)

if \_\_name\_\_ == "\_\_main\_\_":

handle\_data()

infer.py:

import torch

import pickle

from transformers import AlbertModel, BertTokenizer

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

model = torch.load('save/model\_epoch15.pkl', map\_location=torch.device('cpu'))

output = open('cws\_result.txt', 'w', encoding='utf-8')

label\_output = open('./data/cws\_res.txt', 'w', encoding='utf8')

tokenizer = BertTokenizer.from\_pretrained("voidful/albert\_chinese\_base")

with open('data/datasave.pkl', 'rb') as inp:

tag2id = pickle.load(inp)

id2tag = pickle.load(inp)

input\_ids\_l = pickle.load(inp)

label\_l = pickle.load(inp)

input\_mask\_l = pickle.load(inp)

output\_mask\_l = pickle.load(inp)

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

for test in f:

flag = False

test = test.strip()

tokens = [i for i in test]

tokens\_cs = '[CLS]' + ' '.join(tokens) + '[SEP]'

tokenized\_text = tokenizer.tokenize(tokens\_cs)

input\_ids = tokenizer.convert\_tokens\_to\_ids(tokenized\_text)

input\_mask = [1] \* len(input\_ids)

while len(input\_ids) < 512:

input\_ids.append(0)

input\_mask.append(0)

output\_mask = [1] \* len(tokens)

output\_mask = [1] + output\_mask + [1]

while len(output\_mask) < 512:

output\_mask.append(0)

input\_ids = torch.LongTensor(input\_ids).view(1, -1)

input\_mask = torch.BoolTensor(input\_mask).view(1, -1)

output\_mask = torch.BoolTensor(output\_mask).view(1, -1)

predict = model.infer(input\_ids, input\_mask, output\_mask)[0]

for i in range(len(test)):

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

print(id2tag[predict[i+1]], end=' ', file=label\_output)

if id2tag[predict[i+1]] in ['E', 'S']:

print(' ', end='', file=output)

print(file=output)

print(file=label\_output)

output.close()

label\_output.close()

merge.py：

import sys

with open('data/cws\_res.txt', 'r', encoding='utf8') as cws:

with open('data/ner\_res.txt', 'r', encoding='utf8') as ner:

with open('data/test\_data.txt', 'r', encoding='utf8') as test:

with open('cws\_result.txt', 'w', encoding='utf8') as out:

for cws\_label, ner\_label, line in zip(cws, ner, test):

cws\_label = cws\_label.split(" ")

ner\_label = ner\_label.split(" ")

length = len(line)

assert len(cws\_label) == len(ner\_label)

assert len(cws\_label) == length

if len(sys.argv) > 2 and sys.argv[1] == 'on':

i = 0

while i < length:

if line[i] == '《':

print('title')

j = i + 1

i = i + 1

while line[j] != '》':

j = j + 1

j = j - 1

if j - i == 0:

cws\_label[i] = 'S'

break

cws\_label[i] = 'B'

while i < j:

i = i + 1

cws\_label[i] = 'M'

cws\_label[j] = 'E'

if ner\_label[i][0] == 'B':

# print('encounter entity')

j = i

needReplace = False

count = 0

while ner\_label[j][0] != 'E':

if cws\_label[j] == 'E':

count = count + 1

if count >= 2:

needReplace = True

break

j = j + 1

if cws\_label[i] == 'E' and count == 1:

needReplace = True

if needReplace:

print('replace')

if cws\_label[i] == 'M' or cws\_label[i] == 'E':

if cws\_label[i - 1] == 'B':

cws\_label[i - 1] = 'S'

elif cws\_label[i - 1] == 'M':

cws\_label[i - 1] = 'E'

cws\_label[i] == 'B'

elif cws\_label[i] == 'S':

cws\_label[i] = 'B'

i = i + 1

while ner\_label[i][0] != 'E':

cws\_label[i] = 'M'

i = i + 1

if cws\_label[i] == 'B' or cws\_label[i] == 'M':

if cws\_label[i + 1] == 'M':

cws\_label[i + 1] = 'B'

elif cws\_label[i + 1] == 'E':

cws\_label[i + 1] = 'S'

cws\_label[i] = 'E'

elif cws\_label[i] == 'S':

cws\_label[i] == 'E'

else:

i = j

i = i + 1

# print(' '.join(cws\_label))

# exit(0)

for i in range(len(line)):

print(line[i], end='', file=out)

if cws\_label[i] in ['E', 'S']:

print(' ', end='', file=out)

# print(file=out)

model.py:

from math import e

import torch

import torch.nn as nn

from torchcrf import CRF

from torch.nn.utils.rnn import pack\_padded\_sequence, pad\_packed\_sequence

from transformers import BertModel, BertTokenizer

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.tokenizer = BertTokenizer.from\_pretrained("hfl/chinese-roberta-wwm-ext")

self.word\_embeds = BertModel.from\_pretrained("hfl/chinese-roberta-wwm-ext")

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, input\_ids, input\_mask):

batch\_size = input\_ids.size(0)

embeds = self.word\_embeds(input\_ids=input\_ids, attention\_mask=input\_mask, output\_hidden\_states=True, output\_attentions=True)

# idx->embedding

all\_hidden\_states, all\_attention = embeds[-2:]

embeds = all\_hidden\_states[-2]

# LSTM forward

self.hidden = self.init\_hidden(batch\_size, input\_ids.device)

lstm\_out, self.hidden = self.lstm(embeds, self.hidden)

lstm\_feats = self.hidden2tag(lstm\_out)

return lstm\_feats

def forward(self, input\_ids, label, input\_mask, output\_mask):

# print(input\_ids)

# print(label)

# print(input\_mask)

# print(output\_mask)

emissions = self.\_get\_lstm\_features(input\_ids, input\_mask)

loss = -self.crf(emissions, label, output\_mask, reduction='mean')

return loss

def infer(self, input\_ids, input\_mask, output\_mask):

emissions = self.\_get\_lstm\_features(input\_ids, input\_mask)

return self.crf.decode(emissions, output\_mask)

# 附录B 命名实体识别实现的源程序