

**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202001**

**学 号： U202015533**

**姓 名： 徐瑞达**

**指导教师： 魏巍**

**报告日期： 2023年 6 月 5 日**

**计算机科学与技术学院**

目 录

[1 中文分词实现 2](#_Toc137402127)

[1.1 问题描述 2](#_Toc137402128)

[1.1.1 基于词典匹配的分词算法 2](#_Toc137402129)

[1.1.2 基于统计学习的分词算法 2](#_Toc137402130)

[1.2 基础模块 2](#_Toc137402131)

[1.3 系统实现 3](#_Toc137402132)

[1.3.1 总体实现 3](#_Toc137402133)

[1.3.2 数据处理 3](#_Toc137402134)

[1.3.3 数据加载 3](#_Toc137402135)

[1.3.4 构建模型 4](#_Toc137402136)

[1.3.5 训练及验证 4](#_Toc137402137)

[1.3.6 模型预测与融合 4](#_Toc137402138)

[1.4 实验小结 5](#_Toc137402139)

[参考文献 6](#_Toc137402140)

[附录A 中文分词实现的源程序 7](#_Toc137402141)

# 1 中文分词实现

## 1.1 问题描述

本实验需要实现中文分词任务，可以考虑基于统计、基于词典的分词方法等，也可以考虑使用Bi-LSTM+CRF的深度学习模型进行分词，通过对比不同算法分词效果和性能，加深对中文分词算法的理解。

## 基于词典匹配的分词算法

基于词典匹配的分词算法依赖人工建立的词库（词典）进行，包括正向最大匹配法、逆向最大匹配法以及双向最大匹配法。

## 1.1.2 基于统计学习的分词算法

将中文分词视作序列标注任务，给句子中的每个词打上合适的标签即可完成分词任务，而本实验使用BIES标注：

B：代表该字是一个词的开头

I：代表该字在一个词的内部

E：代表该字是一个词的结尾

S：代表该字独立成词

使用上述标注方式对句子进行标注，例如：

B E / S / B E / B I E / B E

小明 / 在 / 中国 / 科学院 / 工作

序列标注方法可以使用HMM等传统统计方法或者Bi-LSTM等深度学习方法，本实验使用基于双向长短神经网络+条件随机场（Bi-LSTM+CRF）的中文分词模型。

## 1.2 基础模块

本实验采用Bert+Bi-LSTM+CRF模型实现，主要由数据处理、数据加载、构建模型、训练及验证、预测等模块组成。(如图1-1所示)

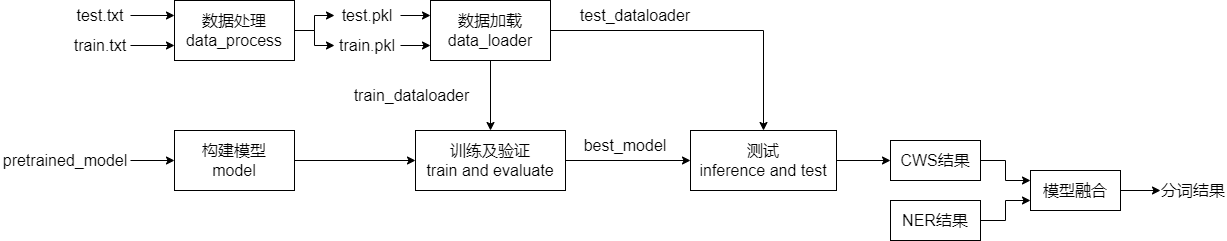


图1-1 模块流程图

## 1.3 系统实现

## 1.3.1 总体实现

该系统可划分为数据处理、模型训练、模型预测三部分。本系统在Bi-LSTM+CRF的模型基础上，将数据编码部分替换为预训练模型Bert，利用Transfomer的自注意力机制得到包含位置信息、语义信息的编码，从而获得更好的分词效果。在模型预测部分，根据CWS输出的标签序列和NER输出的标签序列进行模型融合，以减少CWS任务中对命名实体的错误划分。

run.py文件为系统入口，首先加载预处理后的数据，并将其划分为训练集和验证集（此处划分比例为9:1），然后从预训练模型中构建得到模型，设置优化率和学习率衰减，进行训练。每轮训练完成后，将得到的最优模型结果保存，当若干轮次的验证集损失不再有明显降低后即停止训练。根据最优模型结果，进行预测得到CWS任务的分词结果，然后使用同样的模型结构进行NER任务，得到命名实体标注结果，进行二者之间的模型融合得到最终分词结果。

## 1.3.2 数据处理

在data\_process.py中，预处理训练集。在处理时，根据空格位置将每个字符转换为BIES标注。每行的标注结果存储在一个列表中，而由于Bert中输入序列的最长长度为512字符，因此当某行的长度大于max\_len时，需要进行切分，切分后的多个子句需要在句尾添加分隔符以表示其属于同一个输入序列。

将预处理后的将预处理后的结果输出到二进制文件training.pkl中。

## 1.3.3 数据加载

在data\_loader.py中，加载预处理后的数据到Dataset对象中。

首先从预训练模型tacl-bert-base-chinese中加载得到BertTokenizer，将预处理得到的words\_list的每个字符转换为token，并在每一行的开头加上[CLS]表示句子开头，每一句的结尾加上[SEP]表示句子分隔，经过以上处理后得到dataset。

其次在collate\_fn函数中获取batch数据时，还需要进行数据对齐。首先将每行数据填充至统一长度，然后将token与label对齐，最后将数据转换为张量。

## 1.3.4 构建模型

在model.py中，构建模型结构。模型依次分为BertModel、Bi-LSTM、CRF三层，首先根据输入的token与label获取bert模型的输出，然后去除[CLS]等标签，依次输入到Bi-LSTM、CRF获取损失值和输出。

BertModel使用了HuggingFace[2]提供的transfomers库中的bert模型，使用到的预训练模型有bert-base-chinese、chinese-macbert-base、tacl-bert-base-chinese、RoBERTa-wwm-ext-large-chinese、LTP-tiny、LTP-small、LTP-base等模型，最终结合模型性能和训练算力限制使用了tacl-bert-base-chinese模型。

## 1.3.5 训练及验证

在train.py中，进行模型的训练和验证。在每一轮训练过程中，依次进行模型预测、计算损失、反向传播、梯度裁剪、学习率衰减等过程，每轮训练完成后，即再验证集上评估当前模型性能，计算得到验证集损失和F1-score。每得到更好的F1-score值，即保存模型参数，当F1-score变化不大时即停止训练。

## 1.3.6 模型预测与融合

在test.py中，进行CWS任务的模型预测。在merge.py中，进行CWS任务和NER任务结果的模型融合。进行模型融合时，先需要将CWS结果和NER结果输出为标注格式的文件；然后遍历两个文件，如果某个字符的NER标注为I则说明该字符属于某个命名实体，直接输出字符，否则根据CWS标注结果决定是否要在字符后输出空格。进行模型融合后，在线测试性能提升了0.2个百分点。

## 1.4 实验小结

1. 经过模型训练，最终选取的主要超参数如下：

lr = 0.005

epoch = 14

clip\_grad = 5

batch\_size = 32

hidden\_dim = 200

embedding\_dim = 100

weight\_decay = 0.01

pretained\_model = tacl-bert-base-chinese

最终得到的最高F1值为0.912，相较基础模型提高了约6个百分点。

1. 训练过程中的遇到的问题如下：

由于微调预训练模型需要较大的算力，因此需要选择合适的预训练模型，对于LTP-tiny、LTP-small等模型，由于其参数规模较小，往往耗费算力训练十余个epoch之后仍较欠拟合，而对于参数规模较大的模型，其训练时间又会较长。经过综合考量，选取了tacl-bert-base-chinese模型，其使用了TaCL (Token-aware Contrastive Learning)技术[3]，能够得到更具各向同性和判别性的语义信息。

1. 小结

在本实验中，通过预训练语言模型引入更好的语义编码，大大提升了中文分词的准确性。通过对预训练语言模型在训练集上进行微调，即可使其在下游任务上得到很好的表现。然而，表现较好的预训练语言模型参数规模也较大，微调时使用的算法也较大，因此，开发轻量级的预训练语言模型是NLP领域的重要研究方向。

通过本次实验，我了解并掌握了NLP任务的处理流程——数据处理、模型构建、训练优化、模型测试等步骤，也让我对NLP领域产生了更加浓厚的兴趣。

# 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

[2] Hugging Face.https://huggingface.co/

[3] Yixuan Su, Fangyu Liu, Zaiqiao Meng, Tian Lan, Lei Shu, Ehsan Shareghi, Nigel Collier. TaCL: Improving BERT Pre-training with Token-aware Contrastive Learning

# 附录A 中文分词实现的源程序

### data\_process.py

import os

import pickle

import re

import config

class Processor:

"""

数据处理器

"""

def process(self):

self.process\_data("train")

self.process\_data("test")

@staticmethod

def cut(raw, max\_len, sep):

"""

将raw按照max\_len切分，分隔符为sep\_word

"""

list\_groups = zip(\*(iter(raw),) \* max\_len)

end\_list = [list(i) + list(sep) for i in list\_groups]

count = len(raw) % max\_len

if count != 0:

end\_list.append(raw[-count:])

else:

end\_list[-1] = end\_list[-1][:-1]

return end\_list

@staticmethod

def toTag(input):

"""

将每个字转换为BMES标注

"""

output\_str = []

if len(input) == 1:

output\_str.append('S')

elif len(input) == 2:

output\_str = ['B', 'E']

else:

M\_num = len(input) - 2

M\_list = ['M'] \* M\_num

output\_str.append('B')

output\_str.extend(M\_list)

output\_str.append('E')

return output\_str

def process\_data(self, mode):

# 判断训练集或测试集结果是否已存在

if mode == "train":

path = config.TRAINING\_PATH

origin\_path = config.TRAINING\_DATA

else:

path = config.TESTING\_PATH

origin\_path = config.TEST\_DATA

if os.path.exists(path) is True:

return

with open(origin\_path, 'r', encoding='utf-8') as f:

words\_list = []

tags\_list = []

# 行数

count = 0

# 切分次数

sep\_count = 0

for line in f:

words = []

tags = []

line = line.strip()

if not line:

continue

# 读取每个字到words

for i in range(len(line)):

if line[i] == " ":

continue

words.append(line[i])

# 将每个字转换为BIES标注,记录在labels

line = line.split(" ")

for item in line:

if item == "":

continue

tags.extend(Processor.toTag(item))

# 如果字数大于设定的最大长度，则进行切分

if len(words) > config.max\_len:

# 获取切分后的words列表

sub\_words\_list = Processor.cut(words, config.max\_len - 5, config.sep\_word)

sub\_labels\_list = Processor.cut(tags, config.max\_len - 5, config.sep\_label)

words\_list.extend(sub\_words\_list)

tags\_list.extend(sub\_labels\_list)

sep\_count += 1

else:

words\_list.append(words)

tags\_list.append(tags)

count += 1

# 将结果输出为二进制

with open(path, 'wb') as outp:

pickle.dump(words\_list, outp)

pickle.dump(tags\_list, outp)

class Parser:

"""

数据读取器

"""

def analysis(self, mode='training'):

len\_list, word\_list = self.read\_file(mode)

lens = {'<100': 0, '100-200': 0, '200-500': 0, '500-1000': 0, '>1000': 0}

print(len(len\_list), "sentences in the", mode, "file.")

for i in len\_list:

if i <= 100:

lens['<100'] += 1

elif 100 < i <= 200:

lens['100-200'] += 1

elif 200 < i <= 500:

lens['200-500'] += 1

elif 500 < i <= 1000:

lens['500-1000'] += 1

elif i > 1000:

lens['>1000'] += 1

return lens

def read\_file(self, mode='training'):

"""

读取文件

"""

word\_list = []

len\_list = []

if mode == "train":

origin\_path = config.TRAINING\_DATA

else:

origin\_path = config.TEST\_DATA

with open(origin\_path, 'r', encoding='utf-8') as f:

for line in f:

words = []

line = line.strip()

if not line:

continue

for i in range(len(line)):

if line[i] == " ":

continue

words.append(line[i])

if len(words) > config.max\_len:

sub\_word\_list = self.get\_sep\_list(words, config.sep\_word)

for wl in sub\_word\_list:

if len(wl) > config.max\_len or len(wl) == 0:

continue

word\_list.append(wl)

len\_list.append(len(wl))

else:

word\_list.append(words)

len\_list.append(len(words))

return len\_list, word\_list

def get\_sep\_list(self, init\_list, sep\_word):

"""

按标点切分

"""

w = "".join(init\_list)

s = re.split(r"(。)", w)

s = self.add\_sep\_word(s, sep\_word)

s.append("")

s = ["".join(i) for i in zip(s[0::2], s[1::2])]

r = []

for sub\_list in s:

r.append(list(sub\_list))

return r

def add\_sep\_word(self, s\_, sep\_word):

"""

add sep word to string

"""

new = []

for i, item in enumerate(s\_):

if item == "，" or item == "。" or item == "；":

if i == len(s\_) - 2:

if s\_[-1] == '':

new.append(item)

continue

item += sep\_word

new.append(item)

s\_ = new

return s\_

def run():

if os.path.exists(config.TRAINING\_PATH):

os.remove(config.TRAINING\_PATH)

if os.path.exists(config.TESTING\_PATH):

os.remove(config.TESTING\_PATH)

Processor().process()

def anlysis():

print(Parser().analysis("train"))

print(Parser().analysis("test"))

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

run()

anlysis()

### data\_loader.py

import numpy as np

import torch

import pickle

from torch.utils.data import Dataset, DataLoader

from transformers import BertTokenizer

import config

class Sentence(Dataset):

"""

数据集对象

"""

def \_\_init\_\_(self, words, tags):

self.tokenizer = BertTokenizer.from\_pretrained(config.BERT\_MODEL\_PATH, do\_lower\_case=True)

self.dataset = self.preprocess(words, tags)

self.word\_pad\_idx = 0

self.label\_pad\_idx = -1

def preprocess(self, origin\_words\_list, origin\_tags\_list):

data = []

words\_list = []

tags\_list = []

for origin\_words in origin\_words\_list[0:1]:

words = []

words\_len = []

for token in origin\_words:

words.append(self.tokenizer.tokenize(token))

words\_len.append(len(token))

words = ['[CLS]'] + [item for token in words for item in token]

token\_start\_idxs = 1 + np.cumsum([0] + words\_len[:-1])

words\_list.append(((self.tokenizer.convert\_tokens\_to\_ids(words), token\_start\_idxs), origin\_words))

for origin\_tags in origin\_tags\_list[0:1]:

tags = [config.tag2id.get(t) for t in origin\_tags]

tags\_list.append(tags)

for words, tags in zip(words\_list, tags\_list):

data.append((words, tags))

return data

def \_\_len\_\_(self):

return len(self.dataset)

def \_\_getitem\_\_(self, idx):

word = self.dataset[idx][0]

label = self.dataset[idx][1]

return [word, label]

def collate\_fn(self, batch):

"""

生成batch

process batch data, including:

1. padding: 将每个batch的data padding到同一长度（batch中最长的data长度）

2. aligning: 找到每个sentence sequence里面有label项，文本与label对齐

3. tensor：转化为tensor

"""

# 处理后句子

sentences = [x[0][0] for x in batch]

# 原始句子

ori\_sentences = [x[0][1] for x in batch]

# 标签

labels = [x[1] for x in batch]

# batch大小

batch\_len = len(sentences)

# 最长的句子长度

max\_len = max([len(s[0]) for s in sentences])

max\_label\_len = 0

# 初始化对齐后数据

batch\_data = self.word\_pad\_idx \* np.ones((batch\_len, max\_len))

batch\_label\_starts = []

# 对齐数据

for j in range(batch\_len):

cur\_len = len(sentences[j][0])

batch\_data[j][:cur\_len] = sentences[j][0]

word\_indexs = sentences[j][-1]

label\_starts = np.zeros(max\_len)

# 标记label开始位置

label\_starts[[idx for idx in word\_indexs if idx < max\_len]] = 1

batch\_label\_starts.append(label\_starts)

max\_label\_len = max(int(sum(label\_starts)), max\_label\_len)

# 初始化对齐后标签

batch\_labels = self.label\_pad\_idx \* np.ones((batch\_len, max\_label\_len))

# 对齐标签

for j in range(batch\_len):

cur\_tags\_len = len(labels[j])

batch\_labels[j][:cur\_tags\_len] = labels[j]

# 转化为tensor

batch\_label\_starts = np.array(batch\_label\_starts)

batch\_data = torch.LongTensor(batch\_data)

batch\_label\_starts = torch.LongTensor(batch\_label\_starts)

batch\_labels = torch.LongTensor(batch\_labels)

return [batch\_data, batch\_label\_starts, batch\_labels, ori\_sentences]

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

with open(config.TRAINING\_PATH, 'rb') as inp:

words = pickle.load(inp)

labels = pickle.load(inp)

dataset = Sentence(words, labels)

train\_dataloader = DataLoader(dataset, batch\_size=config.batch\_size, shuffle=True,

collate\_fn=dataset.collate\_fn)

for batch\_data, batch\_label\_starts, batch\_labels, ori\_sentences in train\_dataloader:

print(batch\_data, batch\_label\_starts, batch\_labels, ori\_sentences)

break

### model.py

from torch import nn

from transformers.models.bert import \*

from torch.nn.utils.rnn import pad\_sequence

from torchcrf import CRF

class BertSeg(BertPreTrainedModel):

def \_\_init\_\_(self, config):

super(BertSeg, self).\_\_init\_\_(config)

self.num\_labels = config.num\_labels

self.bert = BertModel(config)

self.dropout = nn.Dropout(config.hidden\_dropout\_prob)

self.bilstm = nn.LSTM(

input\_size=config.lstm\_embedding\_size, # 768

hidden\_size=config.hidden\_size // 2, # 1024 / 2

batch\_first=True,

num\_layers=2,

dropout=config.lstm\_dropout\_prob, # 0.5

bidirectional=True

)

self.classifier = nn.Linear(config.hidden\_size, config.num\_labels)

self.crf = CRF(config.num\_labels, batch\_first=True)

self.init\_weights()

def forward(self, input\_data, token\_type\_ids=None, attention\_mask=None, labels=None,

position\_ids=None, inputs\_embeds=None, head\_mask=None):

# input\_data: (input\_ids, input\_token\_starts)

input\_ids, input\_token\_starts = input\_data

# 获取bert模型的输出

outputs = self.bert(input\_ids,

attention\_mask=attention\_mask,

token\_type\_ids=token\_type\_ids,

position\_ids=position\_ids,

head\_mask=head\_mask,

inputs\_embeds=inputs\_embeds)

sequence\_output = outputs[0]

# 去除[CLS]标签等位置，获得与label对齐的pre\_label表示

origin\_sequence\_output = [layer[starts.nonzero().squeeze(1)]

for layer, starts in zip(sequence\_output, input\_token\_starts)]

# 将sequence\_output的pred\_label维度padding到最大长度

padded\_sequence\_output = pad\_sequence(origin\_sequence\_output, batch\_first=True)

# dropout pred\_label的一部分feature

padded\_sequence\_output = self.dropout(padded\_sequence\_output)

# 将padded\_sequence\_output输入到bilstm中

lstm\_output, \_ = self.bilstm(padded\_sequence\_output)

# 得到判别值

logits = self.classifier(lstm\_output)

outputs = (logits,)

if labels is not None:

loss\_mask = labels.gt(-1)

# 计算损失

loss = self.crf(logits, labels, loss\_mask) \* (-1)

outputs = (loss,) + outputs

return outputs

### config.py

import os

# 文件路径

BASE\_DIR = os.getcwd() + "/"

DATA\_DIR = BASE\_DIR + "data/"

MODEL\_DIR = BASE\_DIR + "model/"

BEST\_MODEL\_DIR = BASE\_DIR + "model/best/"

MODEL\_PREFIX = MODEL\_DIR + "model\_epoch"

LOG\_PATH = BASE\_DIR + "log.txt"

TRAINING\_DATA = DATA\_DIR + "train\_pku.txt"

TEST\_DATA = DATA\_DIR + "test\_data.txt"

TRAINING\_PATH = DATA\_DIR + "training.pkl"

TESTING\_PATH = DATA\_DIR + "testing.pkl"

BERT\_MODEL\_PATH = BASE\_DIR + "pretrained/chinese-macbert-base/"

MODEL\_PATH=MODEL\_DIR+"mode.pkl"

# 超参数

lr = 0.005

max\_epoch = 10

min\_epoch = 5

clip\_grad = 5

batch\_size = 1

hidden\_dim = 200

embedding\_dim = 100

weight\_decay = 0.01

# worker数量

num\_workers = 2

# 是否使用gpu

cuda = True

# 是否对Bert进行微调

full\_fine\_tuning = True

# 每行最大长度

max\_len = 500

# 切分分隔符

sep\_word = '@'

sep\_label = 'S'

# 训练集和验证集划分

train\_test\_split\_size = 0.1

random\_state = 43

# 是否重新处理数据

reprocess\_data = False

patience=5

# 标注

tag2id = {'B': 0, 'M': 1, 'E': 2, 'S': 3}

id2tag = {\_id: \_label for \_label, \_id in list(tag2id.items())}

### run.py

import pickle

import logging

from train import train

from utils import set\_logger

from transformers import get\_cosine\_schedule\_with\_warmup

import config

import torch

from torch.utils.data import DataLoader

from data\_process import Processor

from data\_loader import Sentence

from model import BertSeg

from torch.optim import AdamW

from sklearn.model\_selection import train\_test\_split

def run():

# 设置日志

set\_logger()

# 是否开启cuda

enable\_cuda = config.cuda and torch.cuda.is\_available()

# 处理数据

if config.reprocess\_data:

logging.info("Process data...")

Processor().process()

# 加载数据

logging.info("Load data...")

with open(config.TRAINING\_PATH, 'rb') as inp:

words = pickle.load(inp)

labels = pickle.load(inp)

# 划分训练集和验证集

word\_train, word\_test, label\_train, label\_test = train\_test\_split(words, labels,

test\_size=config.train\_test\_split\_size,

random\_state=config.random\_state)

# 训练集

train\_dataset = Sentence(word\_train, label\_train)

train\_dataloader = DataLoader(

dataset=train\_dataset,

shuffle=True,

batch\_size=config.batch\_size,

collate\_fn=train\_dataset.collate\_fn,

drop\_last=False,

num\_workers=config.num\_workers

)

# 验证集

test\_dataset = Sentence(word\_test, label\_test)

test\_dataloader = DataLoader(

dataset=test\_dataset,

shuffle=False,

batch\_size=config.batch\_size,

collate\_fn=test\_dataset.collate\_fn,

drop\_last=False,

num\_workers=config.num\_workers

)

train\_size = len(train\_dataset)

# 加载模型

logging.info("Load pretrained model...")

model = BertSeg.from\_pretrained(config.BERT\_MODEL\_PATH, num\_labels=len(config.tag2id))

# 是否使用cuda

if enable\_cuda:

model = model.cuda()

if config.full\_fine\_tuning:

# 全部参数参与训练

bert\_optimizer = list(model.bert.named\_parameters())

classifier\_optimizer = list(model.classifier.named\_parameters())

no\_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']

optimizer\_grouped\_parameters = [

{'params': [p for n, p in bert\_optimizer if not any(nd in n for nd in no\_decay)],

'weight\_decay': config.weight\_decay},

{'params': [p for n, p in bert\_optimizer if any(nd in n for nd in no\_decay)],

'weight\_decay': 0.0},

{'params': [p for n, p in classifier\_optimizer if not any(nd in n for nd in no\_decay)],

'lr': config.lr \* 5, 'weight\_decay': config.weight\_decay},

{'params': [p for n, p in classifier\_optimizer if any(nd in n for nd in no\_decay)],

'lr': config.lr \* 5, 'weight\_decay': 0.0},

{'params': model.crf.parameters(), 'lr': config.lr \* 5}

]

else:

# 只有分类器参与训练

classifier\_optimizer = list(model.classifier.named\_parameters())

optimizer\_grouped\_parameters = [{'params': [p for n, p in classifier\_optimizer]}]

# 优化器

optimizer = AdamW(optimizer\_grouped\_parameters, lr=config.lr)

# 训练步数

train\_steps\_per\_epoch = train\_size // config.batch\_size

# 学习率衰减

scheduler = get\_cosine\_schedule\_with\_warmup(optimizer,

num\_warmup\_steps=2 \* train\_steps\_per\_epoch,

num\_training\_steps=config.max\_epoch \* train\_steps\_per\_epoch)

# 训练

train(train\_dataloader,test\_dataloader, model, optimizer, scheduler, enable\_cuda)

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

run()

### test.py

import torch

import pickle

import config

from data\_loader import Sentence

from torch.utils.data import DataLoader

from model import BertSeg

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

# 加载模型

model = BertSeg.from\_pretrained(

config.BERT\_MODEL\_PATH, num\_labels=len(config.tag2id))

model.eval()

# 是否使用cuda

enable\_cuda = config.cuda and torch.cuda.is\_available()

# 输出文件

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

# 加载测试集

with open(config.TESTING\_PATH, 'rb') as inp:

word\_test = pickle.load(inp)

label\_test = pickle.load(inp)

# 测试集

test\_dataset = Sentence(word\_test, label\_test)

test\_loader = DataLoader(

dataset=test\_dataset,

shuffle=False,

batch\_size=config.batch\_size,

collate\_fn=test\_dataset.collate\_fn,

drop\_last=False,

num\_workers=config.num\_workers

)

id2tag = config.id2tag

pred\_labels = []

print("Start predict...")

with torch.no\_grad():

for idx, batch\_samples in enumerate(test\_loader):

batch\_data, batch\_token\_starts, batch\_labels, ori\_data = batch\_samples

if enable\_cuda:

batch\_data = batch\_data.cuda()

batch\_token\_starts = batch\_token\_starts.cuda()

batch\_labels = batch\_labels.cuda()

# 获取mask

batch\_masks = batch\_data.gt(0)

label\_masks = batch\_labels.gt(-1)

# 计算模型预测值

batch\_output = model((batch\_data, batch\_token\_starts),

token\_type\_ids=None, attention\_mask=batch\_masks)[0]

batch\_output = model.crf.decode(batch\_output, mask=label\_masks)

pred\_labels.extend([[id2tag.get(idx) for idx in indices]

for indices in batch\_output])

print("Start output...")

# 输出结果

with open(config.TEST\_DATA, 'r', encoding='utf-8') as f:

for idx, test in enumerate(f):

if (idx < pred\_labels.\_\_len\_\_()):

for i in range(len(test)):

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

if (i < pred\_labels[idx].\_\_len\_\_() and pred\_labels[idx][i] in ['E', 'S']):

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

print(file=output)

print("Done!")

### train.py

import logging

import torch

from tqdm import tqdm

import config

from torch import nn

from utils import f1\_score

def train(train\_dataloader,test\_dataloader, model, optimizer, scheduler,enable\_cuda):

best\_f1=0

patience=0

# 开始训练

for epoch in range(config.max\_epoch):

# 训练

train\_epoch(train\_dataloader, model, optimizer, scheduler, epoch, enable\_cuda)

# 验证

result = evaluate(test\_dataloader, model, enable\_cuda)

logging.info(

"Epoch: {}, test loss: {}, fscore: {}, percision: {},recall: {}".format(epoch, result['loss'],

result['f'],

result['p'], result['r']))

f1 = result['f']

improve = f1 - best\_f1

if improve > 1e-5:

best\_f1 = f1

model.save\_pretrained(config.BEST\_MODEL\_DIR)

if improve < config.patience:

patience += 1

else:

patience = 0

else:

patience += 1

if (patience >= config.patience and epoch > config.min\_epoch) or epoch == config.max\_epoch:

break

# 保存模型

# torch.save(model, config.MODEL\_PREFIX + str(epoch) + ".pkl")

def train\_epoch(train\_dataloader, model, optimizer, scheduler, epoch, enable\_cuda):

"""

进行一轮训练

:param train\_dataloader: 数据

:param model: 模型

:param optimizer: 优化器

:param scheduler: 学习率预热

:param epoch: 轮数

:param enable\_cuda: 是否使用cuda

"""

torch.cuda.empty\_cache()

logging.info("Epoch: {} start...".format(epoch))

model.train()

train\_losses = 0

for idx, batch\_samples in enumerate(train\_dataloader):

# 获取数据

batch\_data, batch\_token\_starts, batch\_labels, \_ = batch\_samples

# 是否使用cuda

if enable\_cuda:

batch\_data = batch\_data.cuda()

batch\_token\_starts = batch\_token\_starts.cuda()

batch\_labels = batch\_labels.cuda()

# 获取mask

batch\_masks = batch\_data.gt(0)

# 模型预测

loss = model((batch\_data, batch\_token\_starts),

token\_type\_ids=None, attention\_mask=batch\_masks, labels=batch\_labels)[0]

# 计算损失

train\_losses += loss.item()

# 梯度清零

model.zero\_grad()

# 反向传播

loss.backward()

# 梯度裁剪

nn.utils.clip\_grad\_norm\_(parameters=model.parameters(), max\_norm=config.clip\_grad)

# 更新参数

optimizer.step()

# 更新学习率

scheduler.step()

if idx%100==0:

logging.info("Epoch: {}, step: {}".format(epoch, idx))

# 计算平均损失

train\_loss = float(train\_losses) / len(train\_dataloader)

logging.info("Epoch: {}, train loss: {}".format(epoch, train\_loss))

def evaluate(test\_dataloader, model, enable\_cuda):

"""

评估模型性能

:param test\_dataloader: 验证集数据

:param model: 模型

:param enable\_cuda: 是否使用cuda

"""

# 模型评估

model.eval()

id2tag = config.id2tag

true\_labels = []

pred\_labels = []

test\_losses = 0

# 不计算梯度

with torch.no\_grad():

for batch\_samples in test\_dataloader:

batch\_data, batch\_token\_starts, batch\_labels, ori\_data = batch\_samples

# 是否使用cuda

if enable\_cuda:

batch\_data = batch\_data.cuda()

batch\_token\_starts = batch\_token\_starts.cuda()

batch\_labels = batch\_labels.cuda()

# 获取mask

batch\_masks = batch\_data.gt(0)

label\_masks = batch\_labels.gt(-1)

# 模型预测

loss = model((batch\_data, batch\_token\_starts),

token\_type\_ids=None, attention\_mask=batch\_masks, labels=batch\_labels)[0]

# 计算损失

test\_losses += loss.item()

# 获取预测结果

batch\_output = model((batch\_data, batch\_token\_starts),

token\_type\_ids=None, attention\_mask=batch\_masks)[0]

# 解码

batch\_output = model.crf.decode(batch\_output, mask=label\_masks)

batch\_labels = batch\_labels.to('cpu').numpy()

pred\_labels.extend([[id2tag.get(idx) for idx in indices] for indices in batch\_output])

true\_labels.extend([[id2tag.get(idx) for idx in indices if idx > -1] for indices in batch\_labels])

# 计算f1值

res = {}

f, p, r = f1\_score(true\_labels,pred\_labels)

res['f'] = f

res['p'] = p

res['r'] = r

res['loss'] = float(test\_losses) / len(test\_dataloader)

return res

### utils.py

import transformers

import config

import os

import logging

def set\_logger():

"""

设置日志

"""

transformers.logging.set\_verbosity\_error()

log\_file = os.path.join(config.MODEL\_DIR, config.LOG\_PATH)

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 get\_entities(seq):

"""

Gets entities from sequence.

"""

# for nested list

if any(isinstance(s, list) for s in seq):

seq = [item for sublist in seq for item in sublist + ['O']]

prev\_tag = 'O'

begin\_offset = 0

chunks = []

for i, chunk in enumerate(seq + ['O']):

tag = chunk[0]

if end\_of\_chunk(prev\_tag, tag):

chunks.append((begin\_offset, i - 1))

if start\_of\_chunk(prev\_tag, tag):

begin\_offset = i

prev\_tag = tag

return chunks

def end\_of\_chunk(prev\_tag, tag):

"""Checks if a chunk ended between the previous and current word.

Args:

prev\_tag: previous chunk tag.

tag: current chunk tag.

Returns:

chunk\_end: boolean.

"""

chunk\_end = False

if prev\_tag == 'S':

chunk\_end = True

if prev\_tag == 'E':

chunk\_end = True

# pred\_label中可能出现这种情形

if prev\_tag == 'B' and tag == 'B':

chunk\_end = True

if prev\_tag == 'B' and tag == 'S':

chunk\_end = True

if prev\_tag == 'B' and tag == 'O':

chunk\_end = True

if prev\_tag == 'M' and tag == 'B':

chunk\_end = True

if prev\_tag == 'M' and tag == 'S':

chunk\_end = True

if prev\_tag == 'M' and tag == 'O':

chunk\_end = True

return chunk\_end

def start\_of\_chunk(prev\_tag, tag):

"""Checks if a chunk started between the previous and current word.

Args:

prev\_tag: previous chunk tag.

tag: current chunk tag.

Returns:

chunk\_start: boolean.

"""

chunk\_start = False

if tag == 'B':

chunk\_start = True

if tag == 'S':

chunk\_start = True

if prev\_tag == 'O' and tag == 'M':

chunk\_start = True

if prev\_tag == 'O' and tag == 'E':

chunk\_start = True

if prev\_tag == 'S' and tag == 'M':

chunk\_start = True

if prev\_tag == 'S' and tag == 'E':

chunk\_start = True

if prev\_tag == 'E' and tag == 'M':

chunk\_start = True

if prev\_tag == 'E' and tag == 'E':

chunk\_start = True

return chunk\_start

def f1\_score(y\_true, y\_pred):

true\_entities = set(get\_entities(y\_true))

pred\_entities = set(get\_entities(y\_pred))

nb\_correct = len(true\_entities & pred\_entities)

nb\_pred = len(pred\_entities)

nb\_true = len(true\_entities)

p = nb\_correct / nb\_pred if nb\_pred > 0 else 0

r = nb\_correct / nb\_true if nb\_true > 0 else 0

score = 2 \* p \* r / (p + r) if p + r > 0 else 0

return score, p, r

### merge.py

output\_cws = open('cws\_result2.txt', 'w', encoding='utf-8')

cws\_label = open('cws\_label.txt', 'r', encoding='utf-8')

ner\_label = open('ner\_label.txt', 'r', encoding='utf-8')

cws\_label\_lines=cws\_label.readlines()

ner\_label\_lines=ner\_label.readlines()

for i in range(len(cws\_label\_lines)):

cws\_label\_lines[i]=cws\_label\_lines[i].replace('\n','')

ner\_label\_lines[i]=ner\_label\_lines[i].replace('\n','')

cws=cws\_label\_lines[i].split(' ')

ner=ner\_label\_lines[i].split(' ')

if (i==len(cws\_label\_lines)-1):

next\_ner=['','']

else:

next\_ner=ner\_label\_lines[i+1].replace('\n','').split(' ')

if len(next\_ner)<2:

next\_ner=['','']

if(len(cws)<2):

print(file=output\_cws)

else:

if ner[1]=='I' and next\_ner[1]=='I':

print(cws[0],end='',file=output\_cws)

else:

if cws[1]=='E' or cws[1]=='S':

print(cws[0],end=' ',file=output\_cws)

else:

print(cws[0],end='',file=output\_cws)