

## NLP项目实践——中文序列标注Flat Lattice代码解读、运行与使用

笔记本： 机器学习

创建时间： 2021-11-15 23:48

更新时间： 2021-11-16 01:23

---

## NLP项目实践——中文序列标注Flat Lattice代码解读与使用

- 1. 原文与项目地址
  - 1.1 原文
  - 1.2 项目
- 2. 运行环境
- 3. 项目结构
- 4. 参数介绍
  - 4.1 数据加载参数：
  - 4.2 bert相关参数 (V1)：
  - 4.3 模型参数：
  - 4.4 训练参数
- 5. 模型结构
- 6. 模型训练
- 7. 模型保存、加载与预测
  - 7.1 模型保存
  - 7.2 模型加载
  - 7.3 生成预测
- 8. 应用在自己的数据集

好久没有更新了，有好几个NLP的实践项目一直没有整理，以后会尽量加大更新频率，给大家带来更多的深度学习应用方面的分享。

关于之前几篇博客的评论和私信，一般我都会回复，如果没有回的话可能是我也不懂的，或者当时回不了的，后来又忘记了。我目前主要是做NLP相关的研究，CV上的很多东西没有跟进，平时CV的项目接触的也少了很多，所以很多问题我也不明白，或者没有时间去仔细查代码。对于我没有及时回复的问题和解答不了的问题，在这里道个歉。

进入正题之前，先声明一下，这篇文章是我在阅读原文和项目代码的基础上自己理解和修改的，可能存在不正确不恰当的地方，不能代表原项目本意，包括一些参数的解释，未必完全正确。如果要深入了解这个项目，请仔细阅读原文与代码。

# 1. 原文与项目地址

## 1.1 原文

原文是《FLAT: Chinese NER Using Flat-Lattice Transformer》，解决的是中文命名实体识别的任务，在这里不过多的介绍原理。如果了解原理上的细节或者Lattice在NER上的应用，可以直接在站内或者某乎搜索关键词，已经有很多人详细介绍过了。文章提出的方法在多个数据集上达到了SOTA结果，目前是中文NER的一个主流的方法。

这篇博客虽然叫代码解读，但是我不会把每一个类、每一个方法都解释一遍，因为项目还是比较复杂的，我只会大体介绍其中相对重要的内容，把代码的运行方法讲清楚。如果有细节上的问题，可以留言或私信与我讨论。如果是原理方面的问题，建议先去多看看别人的介绍，把原理搞清楚，不清楚原理的话，想看懂代码是很难的。另外，不能完全以论文为准，有些处理细节在论文原文中没有详细的介绍。

如果我写的有不正确的地方，还请大家帮忙指正。

## 1.2 项目

项目地址：<https://github.com/LeeSureman/Flat-Lattice-Transformer>

项目分为V0和V1两个版本，其中V0是没有Bert的版本，V1是有Bert的版本。由于我个人的疏忽，一开始看的项目是原来的旧版本，只有V0，所以这次代码解读以V0为主，V1在代码结构上与V0比较相似，我在尝试复现论文结果时，在MSRA数据集上f1只达到了91，没有到论文的96，回去看git才发现原来还有个V1版本，后来在加入bert编码之后，复现了论文中96的结果。

原项目中并没有给出模型如何进行预测等使用，在这篇博客中将会给出简单的预测方法。

# 2. 运行环境

代码是使用pytorch实现的，依赖的模块如下：

```
Python: 3.7.3
PyTorch: 1.2.0
```

FastNLP: 0.5.0  
Numpy: 1.16.4

#### (1) 关于pytorch

我使用的pytorch 1.5.0, 只要是1.0以上的版本应该都没有问题。

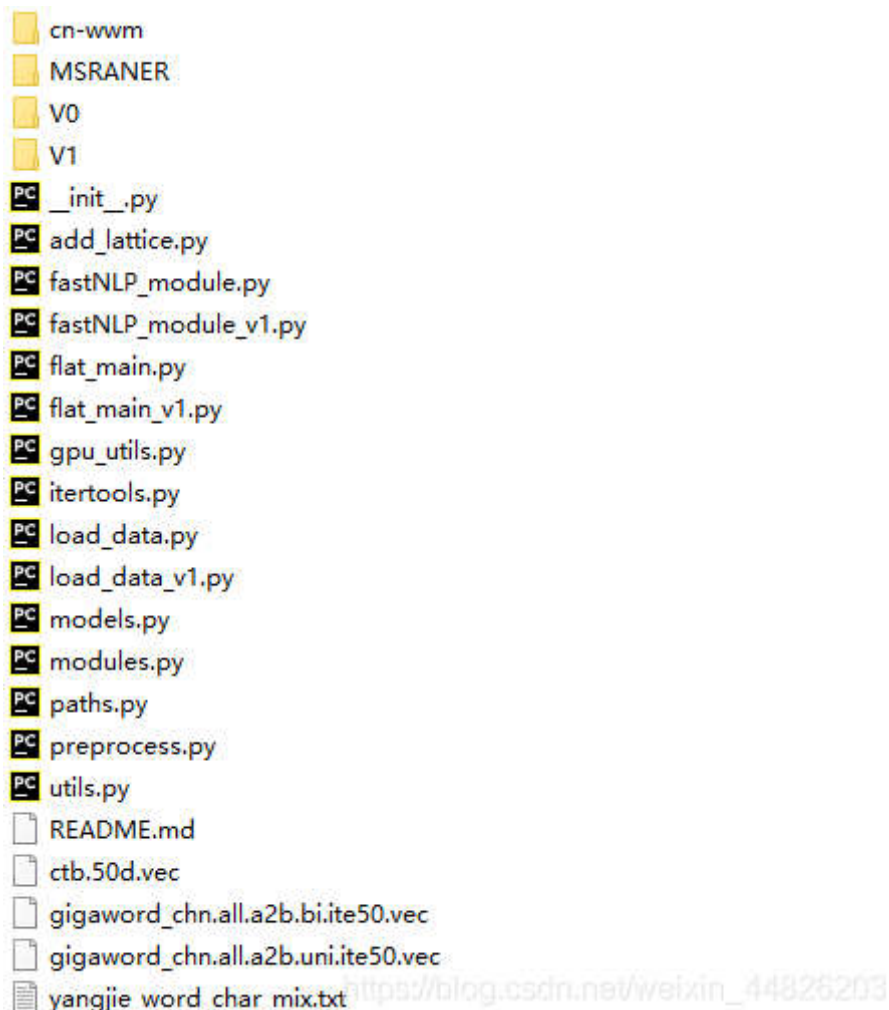
#### (2) 关于FastNLP

FastNLP是作者团队自己做的一个NLP工具包, 跟这个项目比较贴合, 用起来还挺好用的, FLAT的代码中很多类都是定义在FastNLP中的。

与这个项目匹配的是0.5.0的版本, 建议安装这个版本, 严格按照作者的指导, 会比较省事儿。如果你跟我一样安装了0.5.5的新版的FastNLP, 在之后的代码执行过程中可能会遇到些问题。不过没有关系, 都是比较容易解决的, 我在下面也会进行说明。

## 3. 项目结构

项目的整体目录结构如下:



- cn-wwm
- MSRANER
- V0
- V1
- \_\_init\_\_.py
- add\_lattice.py
- fastNLP\_module.py
- fastNLP\_module\_v1.py
- flat\_main.py
- flat\_main\_v1.py
- gpu\_utils.py
- iterutils.py
- load\_data.py
- load\_data\_v1.py
- models.py
- modules.py
- paths.py
- preprocess.py
- utils.py
- README.md
- ctb.50d.vec
- gigaword\_chn.all.a2b.bi.ite50.vec
- gigaword\_chn.all.a2b.uni.ite50.vec
- yangjie\_word\_char\_mix.txt

这个目录并不是原作者git项目的结构, 而是我整理之后的, 直接下载我百度云上传的文件, 解压之后就是这样的。

链接: <https://pan.baidu.com/s/1TGLb44HNQ2ypotxIYbsVJw>

提取码: 2c2y

所做的整理主要如下：

(1) 下载了相关的词向量文件数据集  
需要下载几个词向量文件和数据集如下

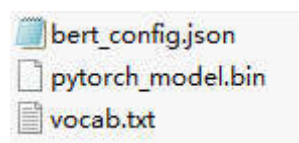
| 文件                                       | 默认值                                    | 解释                       |
|--|--|--------------------------|
|  |  | 单字符的                     |
| yangjie_rich_pretrain_unigram_path       | './gigaword_chn.all.a2b.uni.ite50.vec' | 预训练vec<br>编码             |
|  |  | 双字符的                     |
| yangjie_rich_pretrain_bigram_path        | './gigaword_chn.all.a2b.bi.ite50.vec'  | 预训练vec<br>编码             |
|  |  | 词语的预                     |
| yangjie_rich_pretrain_word_path          | './ctb.50d.vec'                        | 训练vec编<br>码              |
|  |  | 词语和字<br>符混合的             |
| yangjie_rich_pretrain_char_and_word_path | './yangjie_word_char_mix.txt'          | 编码，由<br>preprocess<br>生成 |
| msra_ner_cn_path                         | './MSRANER'                            | msra数据<br>集位置            |

(2) 把V0和V1两个文件夹里边的main搬出来了，防止import的问题

(3) load\_data和fastNLP\_model两个脚本是有V0和V1的区别的。如果执行的是flat\_main.py，则会调用不带v1的脚本，如果是flat\_main\_v1.py，则会调用带v1的脚本。

(4) MSRA是MSRA的序列标注数据集

(5) cn-wwm是一个空文件夹，需要下载中文whole-word-mask的bert预训练权重文件的pytorch版，要求以bin为后缀名，该文件夹下还需要有json格式的bert config文件和词表。



方便起见我也都传百度云了。链接如下：

链接：<https://pan.baidu.com/s/1LdSYaFvgKdhLMXuSK7Fx-w>

提取码: thct

## 4. 参数介绍

项目中出现了大量的可调参数，而且，多数参数并没有给出含义的介绍，这让我在阅读代码时非常困扰，我在代码中挨个去找了这些参数，推断出参数的含义如下总结。

### 4.1 数据加载参数：

| 参数                    | 解释   | 默认值    |
|-----------------------|--|--------|
| dataset               | 数据集名称  | 'msra' |
| bigram_min_freq       | bigram编码时考虑的最小词频                               | 1      |
| char_min_freq         | 单个汉字编码时考虑的最小词频                                 | 1      |
| word_min_freq         | 词语编码时考虑的最小词频                                   | 1      |
| lattice_min_freq      | 添加lattice编码时考虑的最小词频                            | 1      |
| train_clip            | 是否将训练集裁剪到200以下                                 | False  |
| only_train_min_freq   | 仅对train中的词语使用min_freq筛选                        | True   |
| only_lexicon_in_train | 只加载在train中出现过的词汇                               | False  |
| number_normalized     | 0:不<br>norm;1:char;2:char&bi;3:char&bi&lattice | 0      |
| load_dataset_seed     | 随机种子   | 100    |

### 4.2 bert相关参数（V1）：

bert相关的参数只有在V1版本中才会用到。

| 参数             | 解释                     | 默认值 |
|----------------|------------------------|-----|
| use_bert       | 是否使用bert编码             | 1   |
| only_bert      | 是否只使用bert编码            | 0   |
| fix_bert_epoch | 多少轮之后开始训练              | 20  |
|                | bert                   |     |
| after_bert     | 如果只使用bert,<br>bert之后的层 | mlp |

### 4.3 模型参数：

| 参数 | 解释                  | 默认值              |
|----|---------------------|------------------|
| ff | feed-forward中间层的节点个 | 3,修正为hidden * ff |

|                        |   |                     |
|------------------------|---|---------------------|
|                        | 数   |                     |
| hidden                 | SE位置编码和三角函数编码共用的编码维度                                  | 会修正为head_dim * head |
| layer                  | Transformer中Encoder_Layer的数量                          | 1                   |
| head                   | multi-head-attn中head的个数                               | 8                   |
| head_dim               | multi-head-attn中每个head的编码维度                           | 20                  |
| scaled                 | multi-head-attn中是否对attn标准化                            | False               |
| attn_ff                | 是否在self-attn layer最后加一个linear层                        | False               |
| ff_activate            | feed-forward中的激活函数                                    | 'relu'              |
| use_bigram             | 是否使用双字符编码   | 1                   |
| use_abs_pos            | 是否使用绝对位置编码  | False               |
| use_rel_pos            | 是否使用相对位置编码  | True                |
| rel_pos_shared         | 是否共享相对位置，无效参数   | True                |
| add_pos                | 是否在transformer_layer中通过concat加入位置信息，无效参数              | False               |
| learn_pos              | 绝对和相对位置编码中编码是否可学习(是否计算梯度)                             | False               |
| pos_norm               | 是否对位置编码进行norm(pe/pe_sum)                              | False               |
| rel_pos_init           | 相对位置编码初始化编码方向，0: 左向右；1: 右向左。无效参数                      | 1，但是实际调用的类中写死为0     |
| four_pos_shared        | 4个位置编码是不是共享权重   | True                |
| four_pos_fusion        | 4个位置编码融合方法'ff', 'attn', 'gate', 'ff_two', 'ff_linear' | ff_two              |
| four_pos_fusion_shared | 要不要共享4个位置融合之后形成的pos                                   | True                |
| k_proj                 | attn中是否将key经过linear层                                  | False               |

|               |                         |      |
|---------------|-------------------------|------|
| q_proj        | attn中是否将query经过linear层  | True |
| v_proj        | attn中是否将value经过linear层  | True |
| r_proj        | attn中是否将相对位置编码经过linear层 | True |
| embed_dropout | embedding中的dropout      | 0.5  |
| ff_dropout    | ff层中的dropout            | 0.15 |
| ff_dropout_2  | 第二个ff层中的dropout         | 0.15 |
| attn_dropout  | attention中的dropout      | 0    |

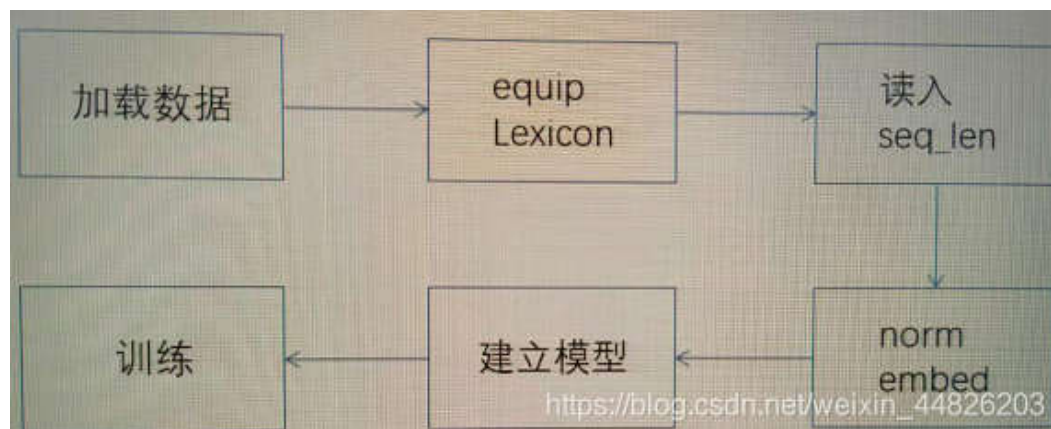
## 4.4 训练参数

训练参数相对容易理解，就不过多解释了（偷懒），把默认值列出来如下。

```
epoch = 10
batch = 1
optim = 'sgd' # sgd|adam
lr = 1e-3
warmup = 0.1
embed_lr_rate = 1
momentum = 0.9
update_every = 1
init = 'uniform' # 'norm|uniform'
self_supervised = False
weight_decay = 0
norm_embed = True
norm_lattice_embed = True
test_batch = batch // 2
```

## 5. 模型结构

模型分为V0和V1两个版本。主要区别就是有没有使用Bert，所调用的类的名称也有所不同，但main脚本的流程没有太大的区别，都是按照如下的过程进行的：



首先加载数据，创建fastNLP中Dataset类型的数据集，可以通过Dataset['train']的方式去索引训练集、验证集和测试集。然后每一个数据集里边，又包含几个字段。

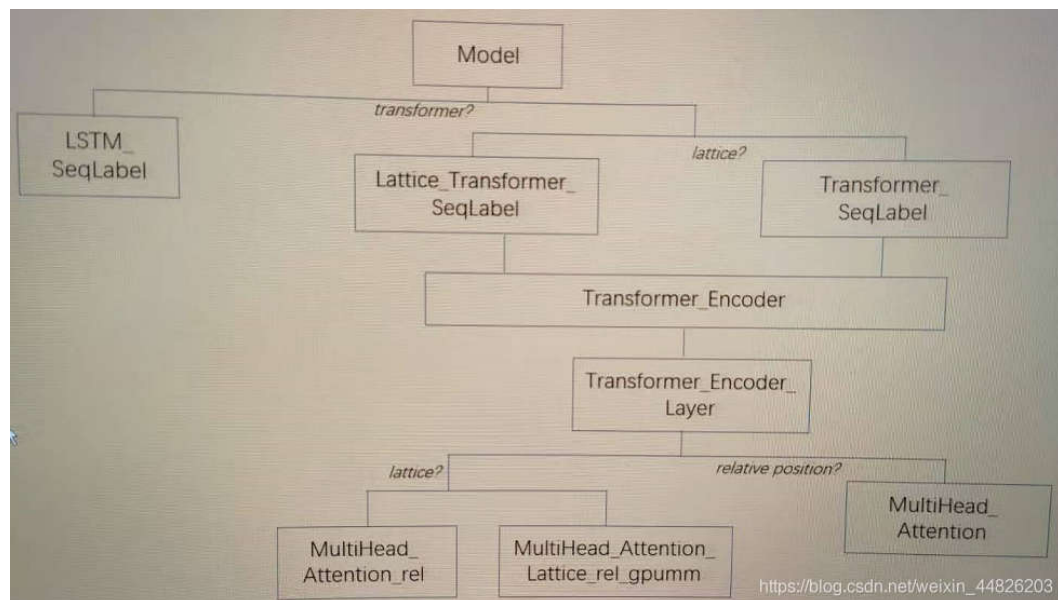
| 字段名    | 含义        |
|--------|-----------|
| char   | 原文本，以字为分割 |
| target | 标签        |
| bigram | 两两连续分割的字符 |

equip lexicon是给数据集添加lattice的过程，就是去词库里边匹配词汇，然后放在原来的token embedding后边。

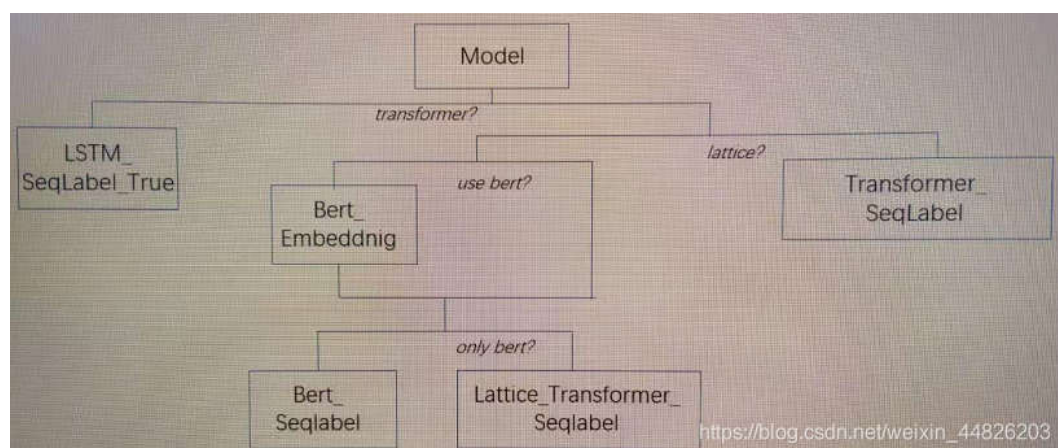
接下来会读入数据的长度，给dataset添加要给seq\_len的字段。然后对所有的embedding进行normalization。

上述准备工作完成之后，就会建立模型，建立哪一种模型会根据用户输入的args进行判断。简单画了两个图，如下。

V0版本的模型结构如下：



V1版本的模型结构如下：



## 6. 模型训练



以MSRA数据集为例，训练只需要在项目目录下执行

```
python flat_main.py --dataset msra
```

可以根据训练参数去传参。

在这里总结一下可能遇到的问题。如果你是直接执行我上传在百度云中的代码，可能不会遇到这些问题，因为我都已经修改过了。

(1) import 报错。无法import `_get_file_name_base_on_postfix`函数。

这个错误很明显，函数名以下短线开头，不能被import。只需要把这个函数复制到报错的脚本`load_data.py`中就可以了。

(2) `ncoding_type` 不对。

在代码里改一下就好了，例如MSRA数据集，改成`bioes`，CLUE数据集，改成`bio`，就可以了。

(3) 缺少数据类型`'chain'`。

这个来自于JetBrainsPyCharmpython\_stubs-1902731831`itertools.py`这个脚本，把这个脚本复制到项目根路径，然后import它就可以了。在报错的脚本中添加：

```
from itertools import chain
```

(4) Bert相关的问题

如果是用的V1版本代码，可以设置使用Bert编码，如果使用0.5.5的FastNLP则会遇到若干问题，问题和解决方法如下。

BertModel的名称不对：

这是因为FastNLP中这个类的名字变了，原本是`_WordBertModel`，改成了`_BertWordModel`，直接复制这个类，改一下名字，丢在`fastNLP_Module_v1.py`里边就可以了。

然后会报缺少bert tokenizer，也是直接把fastNLP里边的`moduletokenizerbert_tokenizer.py`里边相关的代码复制到`fastNLP_Module_v1.py`就可以解决了。

(5) tqdm引发的报错

如果是在jupyter中执行训练，可能会由于tqdm版本的问题引发报错，遇到这种情况只需要把训练参数中的`use tqdm`给关掉就可以了。

## 7. 模型保存、加载与预测

由于作者提供的只包含了训练，没有设置模型的保存、加载与预测相关的功能，所以我在这补充一下这几方面。

## 7.1 模型保存

模型在训练的时候，利用了fastNLP中的一个名为Trainer的类，通过查看这个类的代码可以发现，这个类是写了保存方法的。

只需要在flat\_main.py中，生成Trainer的位置加一个参数save\_path就可以了。

```
trainer = Trainer(datasets['train'], model, optimizer, loss, args.batch,
n_epochs=args.epoch,
dev_data=datasets['dev'],
metrics=metrics,
device=device, callbacks=callbacks, dev_batch_size=args.test_batch,
test_use_tqdm=False, check_code_level=-1,
update_every=args.update_every,
save_path='your_path/flat_lattice/{}/'.format(args.dataset))
```

但是我这样修改了之后，在保存的时候还是报错了，所以我又把Trainer的save和load方法修改了一下。修改后的代码如下：

```
def _save_model(self, model, model_name):
    if self.save_path is not None:
        model_path = os.path.join(self.save_path, model_name)
        if not os.path.exists(self.save_path):
            os.makedirs(self.save_path, exist_ok=True)
        if _model_contains_inner_module(model):
            model = model.module
        torch.save(model.state_dict(), model_path) # 只改了这一行

def _load_model(self, model, model_name):
    if self.save_path is not None:
        model_path = os.path.join(self.save_path, model_name)
        model.load_state_dict(torch.load(model_path))
    elif hasattr(self, "_best_model_states"):
        model.load_state_dict(self._best_model_states)
    else:
        return False
    return True
```

这样改好了之后，执行flat\_main.py脚本进行训练之后，就会在save\_path路径下保存一个模型权重文件。

## 7.2 模型加载

模型加载很简单，只需要在flat\_main.py中，实例化model之后，load之前保存的权重文件就可以了。

```
model_path = '/msra/best_Lattice_Transformer_Seqlabel_f_2021-03-03-14-55-31-899501'
states = torch.load(model_path).state_dict()
model.load_state_dict(states)
```

## 7.3 生成预测

作者并没有给出如何预测，但是在fastNLP中实际上是定义了用于预测的类的，名为predictor，去看一下代码的话，这个类其实写的很简单，但是很实用。使用方法如下：

```
from fastNLP.core.predictor import Predictor
predictor = Predictor(model) # 这里的model是加载权重之后的model

test_label_list = predictor.predict(datasets['test'][:1])['pred'][0] #
    预测结果
test_raw_char = datasets['test'][:1]['raw_chars'][0] # 原始文字
```

上面代码中的test\_label\_list就在test上预测出来的label，label对应的BIO可以通过以下代码查看：

```
for d in vocabs['label']:
    print(d)
```

然后我写了一个简单的方法把label转换成实体（仅适用于MSRA数据集），如下所示：

```
def recognize(label_list, raw_chars):
    """
    根据模型预测的label_list，找出其中的实体
    label_list: array
    raw_chars: list of raw_char
    return: entity_list: list of tuple(ent_text, ent_type)
    -----
    ver: 20210303
    by: changhongyu
    """
    if len(label_list.shape) == 2:
        label_list = label_list[0]
    elif len(label_list) > 2:
        raise ValueError('please check the shape of input')
    assert len(label_list.shape) == 1
    assert len(label_list) == len(raw_chars)
    # 其实没有必要写这个
    # 但是为了将来可能适应bio的标注模式还是把它放在这里了
    starting_per = False
    starting_loc = False
    starting_org = False
    ent_type = None
    ent_text = ''
    entity_list = []
    for i, label in enumerate(label_list):
        if label in [0, 1, 2]:
            ent_text = ''
            ent_type = None
            continue
        # begin
```

```

elif label == 10:
    ent_type = 'PER'
    starting_per = True
    ent_text += raw_chars[i]
elif label == 4:
    ent_type = 'LOC'
    starting_loc = True
    ent_text += raw_chars[i]
elif label == 6:
    ent_type = 'ORG'
    starting_org = True
    ent_text += raw_chars[i]
    # inside
elif label == 9:
    if starting_per:
        ent_text += raw_chars[i]
elif label == 8:
    if starting_loc:
        ent_text += raw_chars[i]
elif label == 3:
    if starting_org:
        ent_text += raw_chars[i]
    # end
elif label == 11:
    if starting_per:
        ent_text += raw_chars[i]
        starting_per = False
elif label == 5:
    if starting_loc:
        ent_text += raw_chars[i]
        starting_loc = False
elif label == 7:
    if starting_org:
        ent_text += raw_chars[i]
        starting_org = False
elif label == 13:
    ent_type = 'PER'
    ent_text = raw_chars[i]
elif label == 12:
    ent_type = 'LOC'
    ent_text = raw_chars[i]
elif label == 14:
    ent_type = 'PER'
    ent_text = raw_chars[i]
else:
    ent_text = ''
    ent_type = None
    continue
if not (starting_per or starting_loc or starting_org) and len(ent_text):
    # 判断实体已经结束，并且提取到的实体有内容
    entity_list.append((ent_text, ent_type))
return entity_list

recognize(test_label_list, test_raw_char)
# Out:
# [('中共中央', 'ORG'),
# ('中国致公党', 'ORG'),

```

```
# ('中国致公党', 'ORG'),  
# ('中国共产党中央委员会', 'ORG'),  
# ('致公党', 'ORG')]
```

## 8. 应用在自己的数据集

在这里我以中文命名实体识别数据集CLUE NER 2020为例，介绍怎样将FLAT用在自己的数据集上。

### (1) 数据格式转换。

首先，不管你的数据格式原来是什么样子，都要转成一行一个字符的格式，例如：

```
浙 B-CMP  
商 I-CMP  
银 I-CMP  
行 I-CMP  
企 O  
业 O  
信 O  
贷 O  
部 O
```

并且命名为train.char.bmes，然后创建一个名为CLUE2020的文件夹，把train，dev和test都放进去。

### (2) 设置路径

打开paths.py，增加一行：

```
clue_2020_ner_path = './CLUE2020'
```

### (3) 写加载数据方法

打开load\_data\_v1.py，仿照load\_ontonotes4ner，写一个新的方法

load\_clue\_2020，几乎就是复制粘贴，改一下加载数据的路径和缓存路径就好了。

### (4) 在main中修改加载的数据集

打开flat\_main\_v1.py，修改dataset为clue，以及读取数据的时候，加一个判断，如果dataset为clue，调用（3）中刚写的load\_clue的方法。

完成了这几步之后，训练时就可以用自己的数据集啦。

```
python flat_main_v1.py --dataset clue
```

然后就可以根据7中的步骤去进行预测了，记得自己写一下recognize的方法，这个识别方法是由数据集的标注格式决定的。

注意一点，缓存的名字是跟着dataset的名字走的，如果你换了数据集的实际内容，而数据集的名称没有改的话，记得去cache里边把缓存清理掉，不然模型运行的时候会有限去找缓存，而缓存存的还是原来的数据集。