**实 验 报 告**

**EXPERIMENT REPORT**

目录

[一 项目简介 1](#_Toc88399327)

[二 论文内容 1](#_Toc88399328)

[1.基本介绍 1](#_Toc88399329)

[2.所作改进 1](#_Toc88399330)

[3.数学模型 3](#_Toc88399331)

[4.结果比对 5](#_Toc88399332)

[三 项目实践 6](#_Toc88399333)

[1.运行环境 6](#_Toc88399334)

[2.项目结构 7](#_Toc88399335)

[3.参数介绍 7](#_Toc88399336)

[1) 数据加载参数 7](#_Toc88399337)

[2) bert相关参数（仅V1） 8](#_Toc88399338)

[3) 模型参数 8](#_Toc88399339)

[4) 训练参数 10](#_Toc88399340)

[4.模型 10](#_Toc88399341)

[1) 模型结构 10](#_Toc88399342)

[2) 模型训练 13](#_Toc88399343)

[3) 模型保存 14](#_Toc88399344)

[4) 模型加载 15](#_Toc88399345)

[5) 模型预测 16](#_Toc88399346)

[5.运行结果 18](#_Toc88399347)

[四 总结反思 20](#_Toc88399348)

[成员分工 20](#_Toc88399349)

[实验感想 20](#_Toc88399350)

# 一 项目简介

本小组对ACL2020中来自复旦大学邱锡鹏老师团队的 *FLAT: Chinese NER Using Flat-Lattice Transformer*进行论文复现，具体实现过程参考了官方开源代码Flat-Lattice-Transformer。接下来报告将大致从两部分来介绍本项目的内容，分别是对论文的理解和项目运行过程记录。

与项目相关性较强的论文一共三篇，放在相关论文文件夹中；语料及训练集较大，上传受限，所以存放在百度云盘，后附链接；开源项目代码也已打包，放在项目代码文件夹中，后附GitHub链接。

语料及训练集百度云链接

链接：https://pan.baidu.com/s/1iJDo\_iQnlcrMjxXXbPK-3Q

提取码：p55m

开源项目Flat-Lattice-Transformer

链接：<https://github.com/LeeSureman/Flat-Lattice-Transformer>

注：若用Jupyter Notebook 在浏览器中打开运行，直接运行Run.ipynb文件，V0版本运行，进入V0文件，运行Run\_V0.ipynb文件，V1版本运行，进入V1文件，运行Run\_V1.ipynb文件。用其他编译器运行可直接运行preprocess.py文件，以及flat\_main.py文件，完整运行步骤见README.MD文件。

# **二** 论文内容

## 1.基本介绍

本文在Lattice LSTM(ACL 2018)的基础上做出了这两方面的改进：

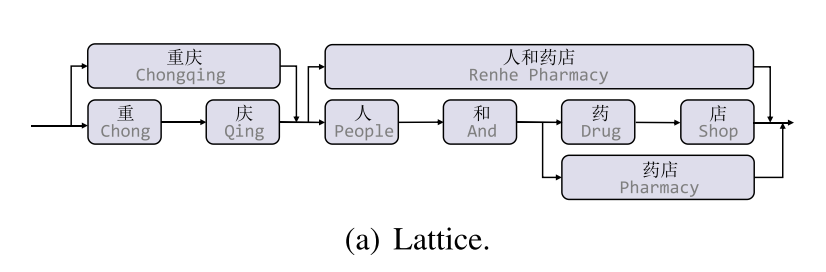
第一，作者提出了一种将Lattice图结构无损转换为扁平的Flat结构的方法，并将LSTM替换为了更先进的Transformer Encoder，该方法不仅弥补了Lattice LSTM无法并行计算(batchsize=1)的缺陷，而且更好地建模了序列的长期依赖关系；

第二，作者提出了一种针对Flat结构的相对位置编码机制，使得字符与词汇得到了更充分更直接的信息交互，在基于词典的中文NER（Named Entity Recognition，命名实体识别）模型中取得了SOTA。

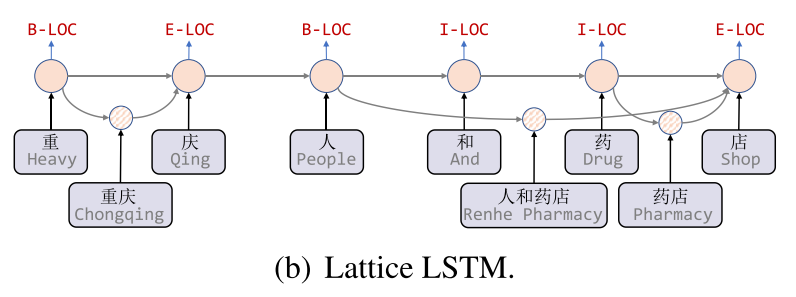
## 2.所作改进

由于中文词汇的稀疏性和模糊性，基于字符的序列标注模型往往比基于词汇的序列标注模型表现更好，但在基于字符的模型中引入分词信息往往能够带来性能的提升，尤其是对于NER任务来说，词汇能够提供丰富的实体边界信息。Lattice LSTM首次提出使用Lattice结构在NER任务中融入词汇信息。

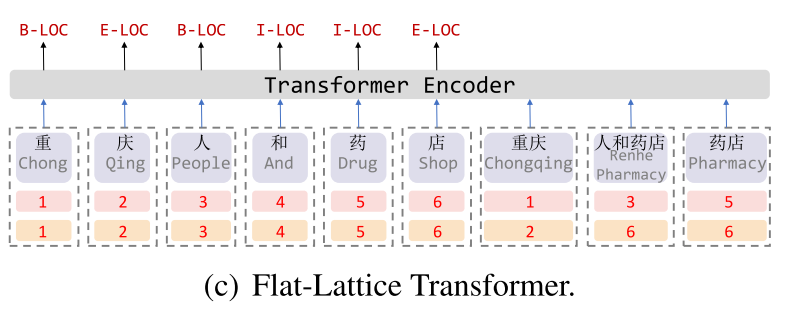
汉字格结构已被证实是一种有效的中文命名实体识别方法，格子结构被证明对利用词信息和避免分词的错误传播有很大的好处。我们可以将一个句子与一个词典进行匹配，得到其中的潜词。格是一个有向无环图，其中每个节点都是一个字符或一个潜在的字。格包括句子中的一系列字符和可能的单词。它们不是按顺序排列的，单词的第一个字符和最后一个字符决定了它的位置。汉字格中的一些词可能对NER很重要。如图(a)所示，一个句子的Lattice结构是一个有向无环图，每个节点是一个字或者一个词， “人和药店(Renhe Pharmacy)”可以用来区分地理实体“重庆(Chongqing)”和“重 庆人(Chongqing People)”。



在Lattice LSTM中，使用一个额外的词元对可能的词进行编码，并使用注意机制在每个位置融合可变数量的节点，如图(b)所示。LR-CNN使用CNN对不同窗口大小的潜在单词进行编码。但该方法存在一定局限性，由于Lattice结构的动态性，Lattice LSTM无法在GPU上并行训练；且RNN和CNN难以建模长距离的依赖关系，在Lattice LSTM中的字符只能获取前向信息，没有和词汇进行足够充分的全局交互。



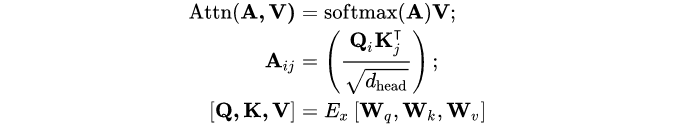
从Transformer的position representation得到启发，作者给每一个token/span(字、词)增加了两个位置编码，分别表示该span在sentence中开始(head)和结束(tail)的位置，对于字来说，head position和tail position是相同的。通过对注意力打分函数的简单改进，使得Transformer结构在NER任务上性能大幅提升，如图 (c)所示。从这样的标签序列中可以无损地重建Lattice结构。同时，扁平的结构允许使用Transformer Encoder，其中的self-attention机制允许任何字符和词汇进行直接的交互。



## 3.数学模型

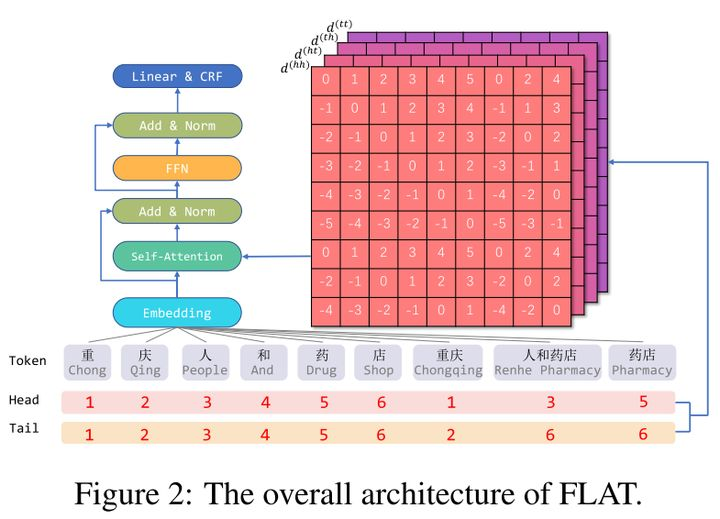
**Muiti-head self-attention**

有了位置编码，容易想到可以像原始Transformer那样将字向量直接和两个位置向量相加，然后参与后续的self-attention：

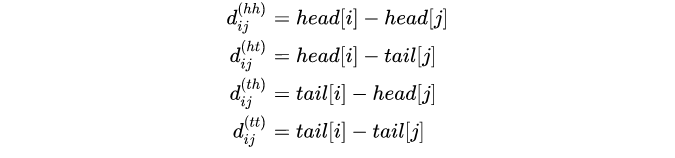


不过这样做肯定不算是有效编码了位置信息，这也是原始Transformer在NER任务上的性能比不过BiLSTM的原因之一。有效的位置编码一直是改进Transformer的重要方向，针对本文提出的Flat结构，作者借鉴并优化了Transformer-XL (ACL 2019)中的相对位置编码方法，有效地刻画了span之间的相对位置信息。

**Relative Position Encoding of Spans**



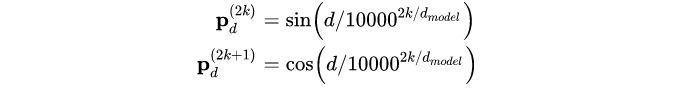
span是字符和词汇的总称，span之间存在三种关系：交叉、包含、分离，然而作者没有直接编码这些位置关系，而是将其表示为一个稠密向量。作者用*head*[*i*]和*tail*[*i*] 表示span的头尾位置坐标，并从四个不同的角度来计算和的距离：



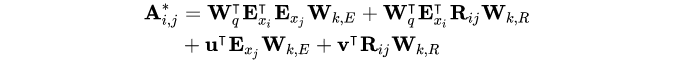
如图2所示，这会得到四个相对距离矩阵：，，，，其中表示的开始位置和的开始位置的距离。然后将这四个距离拼接后作一个非线性变换，得到和的位置编码向量：



其中是Transformer采用的绝对位置编码：



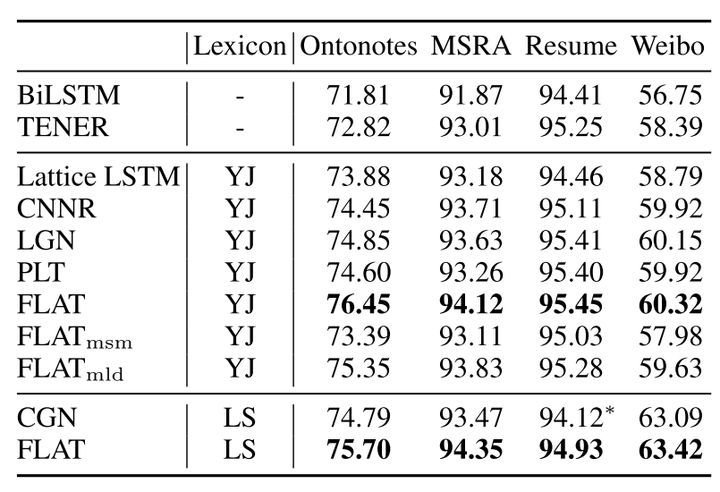
这样，每一个span都可以与任意span进行充分且直接的交互，然后作者采用了Transformer-XL (ACL 2019)中提出的基于相对位置编码的self-attention：

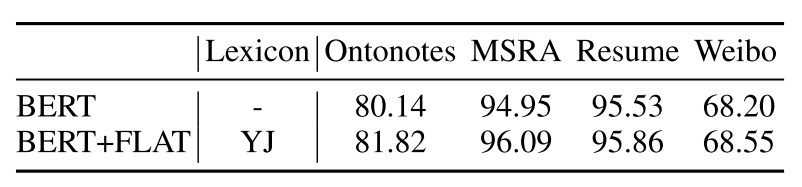


可以直观地将前两项分别看作是两个span之间的内容交互和位置交互，后两项为全局内容和位置bias，在Transformer-XL中是根据绝对位置编码直接计算得出的，而这里的经过了非线性变换的处理。最后，用替换式(1)中的，取出字的编码表示，将其送入CRF层进行解码得到预测的标签序列。

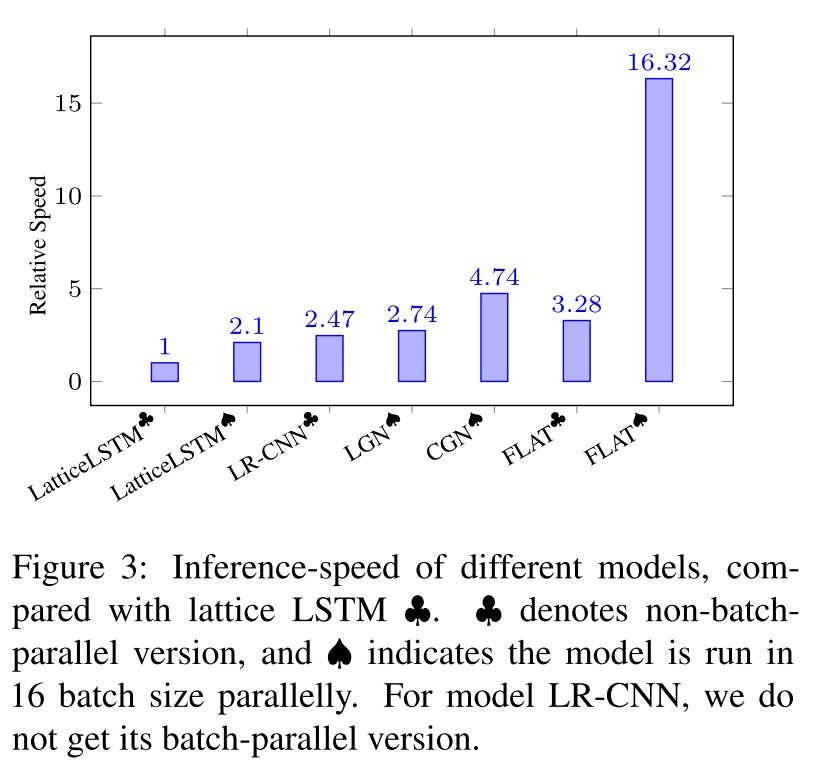
## 4.结果比对

下图给出了论文的实验结果，从图中可以看出，引入词汇信息的方法，都相较于baseline模型biLSTM+CRF有较大提升。可见引入词汇信息可以有效提升中文NER性能。采用相同词表（词向量）时，FLAT好于其他词汇增强方法；FLAT如果mask字符与词汇间的attention，性能下降明显，这表明FLAT有利于捕捉长距离依赖。且FLAT结合BERT效果会更佳。





在推断速度方面，FLAT论文也与其他方法进行了对比，FLAT仅仅采用1层Transformer,在指标领先的同时、推断速度也明显优于其他方法。



# 三 项目实践

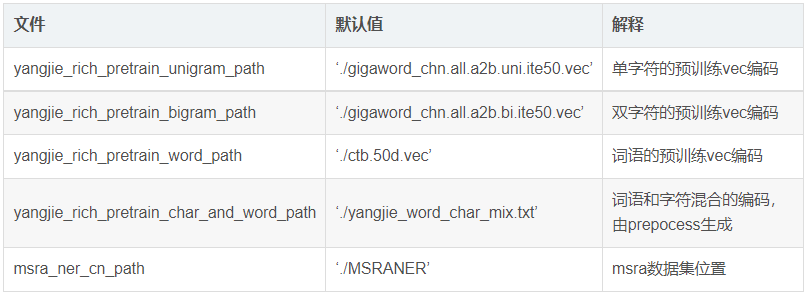
## 1.运行环境



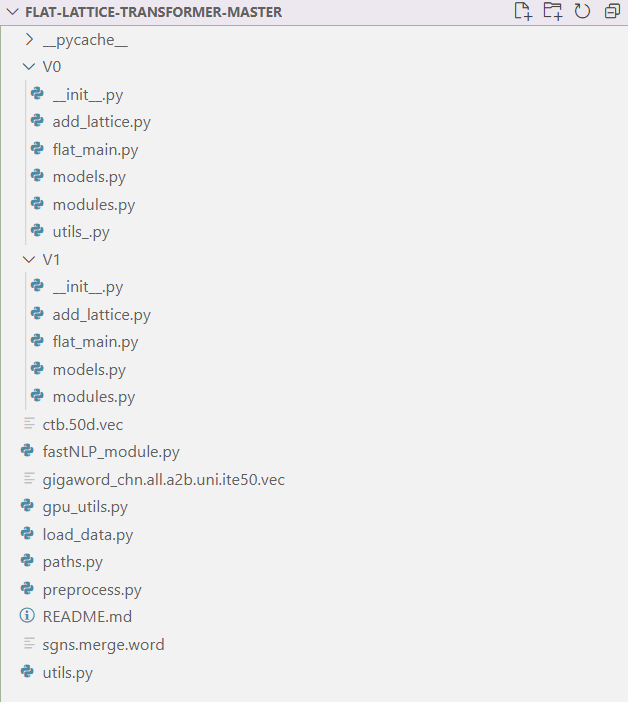
FastNLP是作者团队自己做的一个NLP工具包，和本项目较贴合，使用流畅，FLAT的代码中很多类都已定义在FastNLP中。与本项目匹配的是0.5.0的版本，建议安装这个版本，严格按照作者的指导，否则会遇到运行问题，需要重装。

## 2.项目结构

1需要的词向量文件和数据集如下



2项目结构



## 3.参数介绍

项目中出现了大量的可调参数，且多数参数并未必给出含义的介绍，阅读代码时很影响流畅性，代码能够找到这些参数，并推断出参数的含义如下总结。

### 数据加载参数

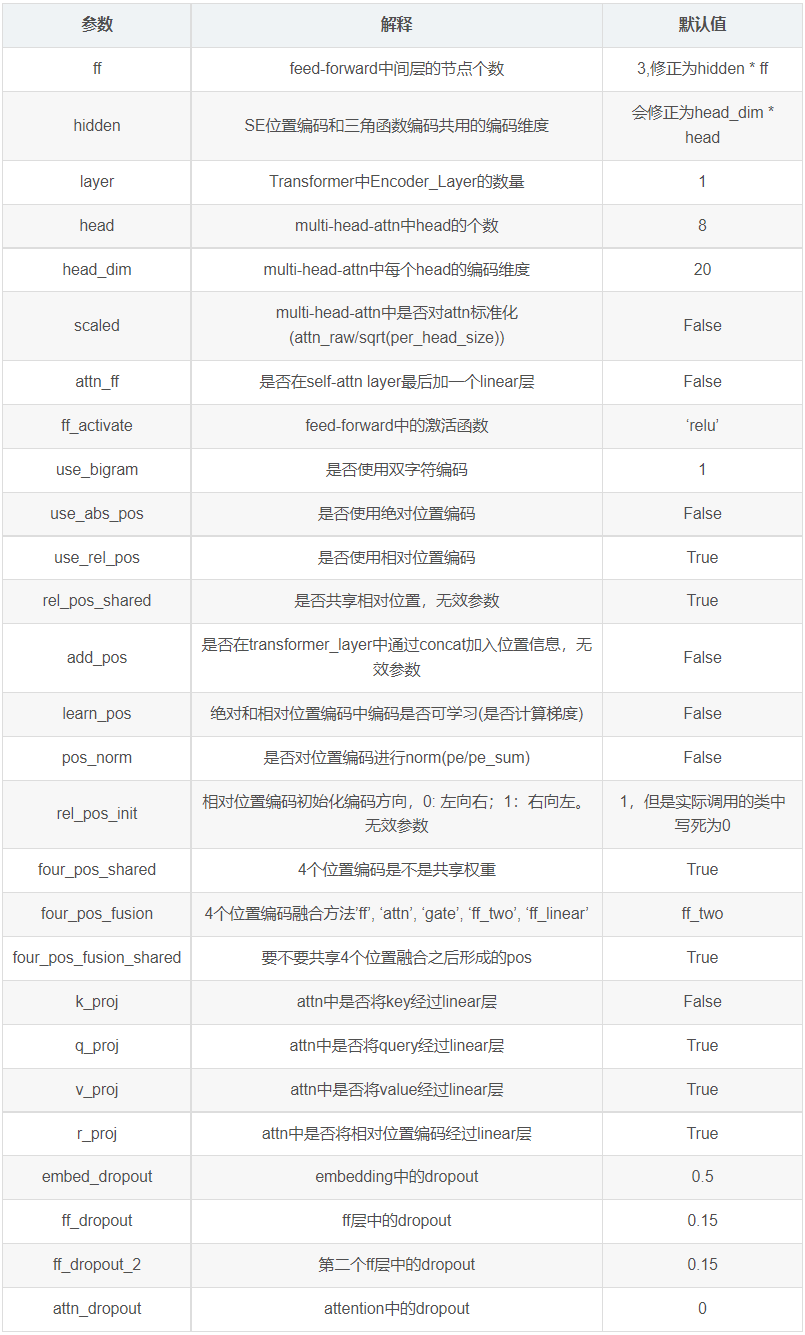


### bert相关参数（仅V1）

bert相关的参数只有在V1版本中才会用到。



### 模型参数



### 训练参数

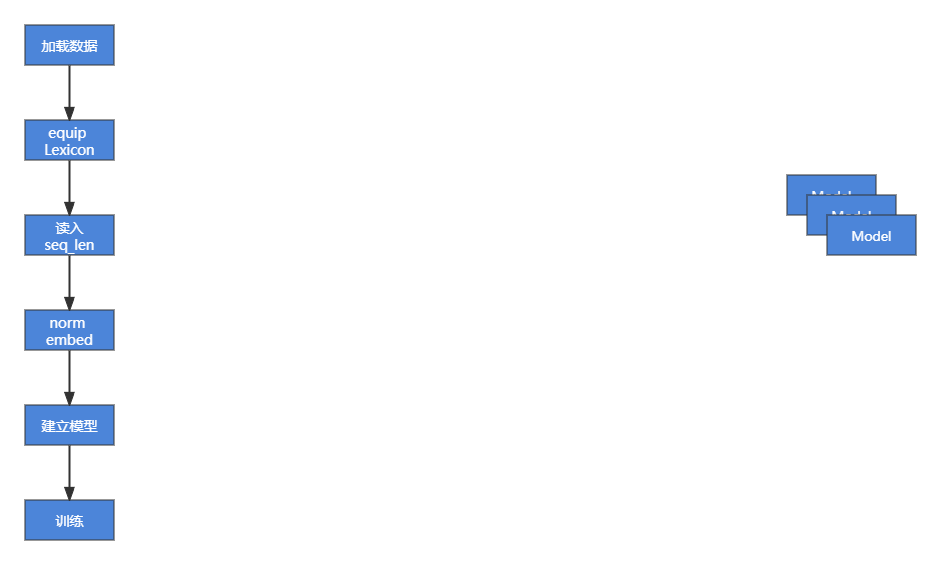
训练参数的默认值列出来后，如下。



## 4.模型

### 模型结构

模型分为V0和V1两个版本。主要区别在于是否使用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。然后每一个数据集中，又包含如下字段。

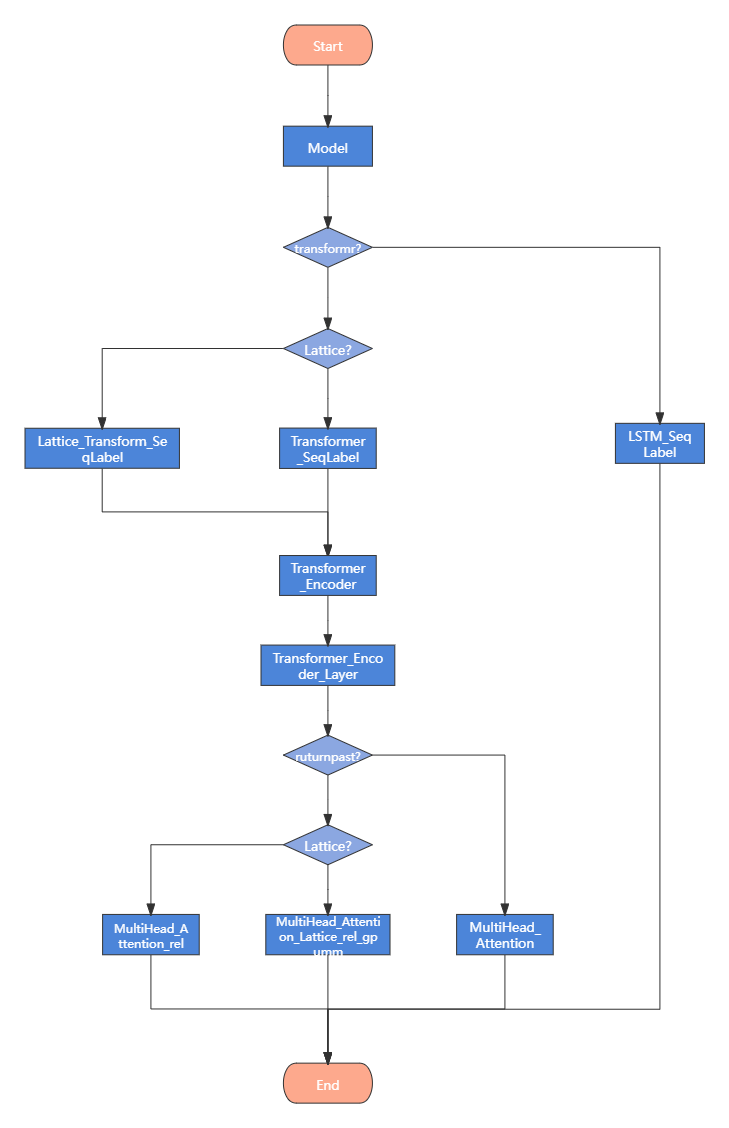


equip lexicon是给数据集添加lattice的过程，即在词库内匹配词汇，然后放在原来的token embedding之后。

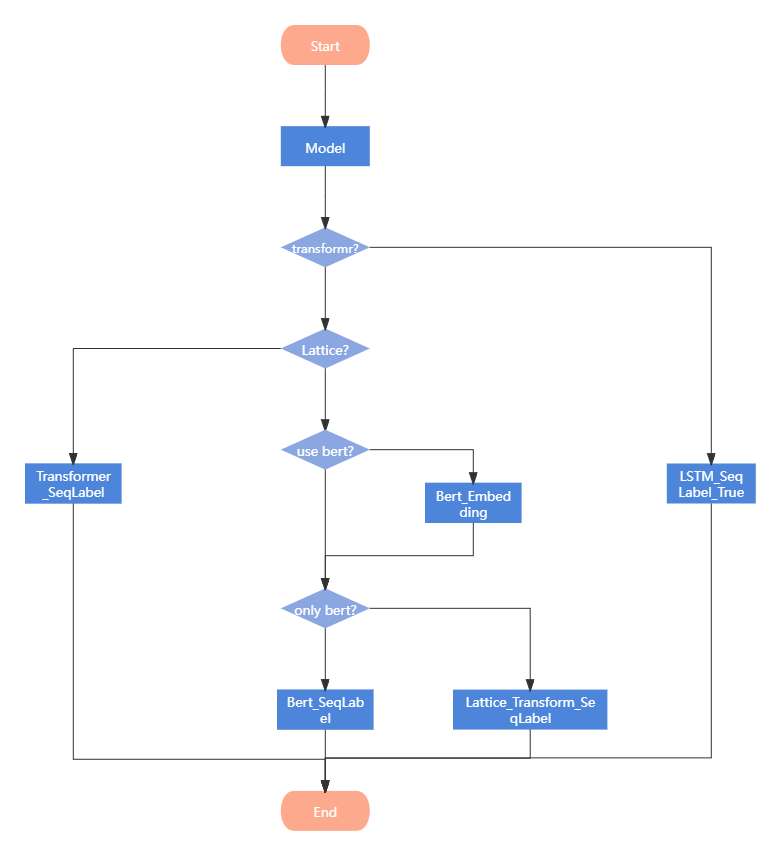
接下来将读入数据的长度，向dataset添加seq\_len的字段，再对所有的embedding进行normalization。

上述准备工作完成后，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。针对不同模型，图示见后。

V0版本的模型结构如下：



V1版本的模型结构如下：



### 模型训练

以MSRA数据集为例，训练只需要在项目目录下执行

就可以根据训练参数去传参。

对遇到的问题进行总结：   
**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中就可以解决。  
**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。  
**3 缺少数据类型‘chain’**

这个来自于\JetBrains\PyCharm\python\_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。在报错的脚本中添加：

**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，更改名字，放入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\modules\tokenizer\bert\_tokenizer.py里边相关的代码复制到fastNLP\_Module\_v1.py就可以解决。  
**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。

### 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，这个类是已写保存方法。  
只需在flat\_main.py中，将生成Trainer的位置加一个参数save\_path。

修改了之后，在保存时仍然报错，所以又对Trainer的save和load方法进行修改。修改后的代码如下：



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。

### 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。



### 模型预测

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：



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：



这里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\_lsit: 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'),*

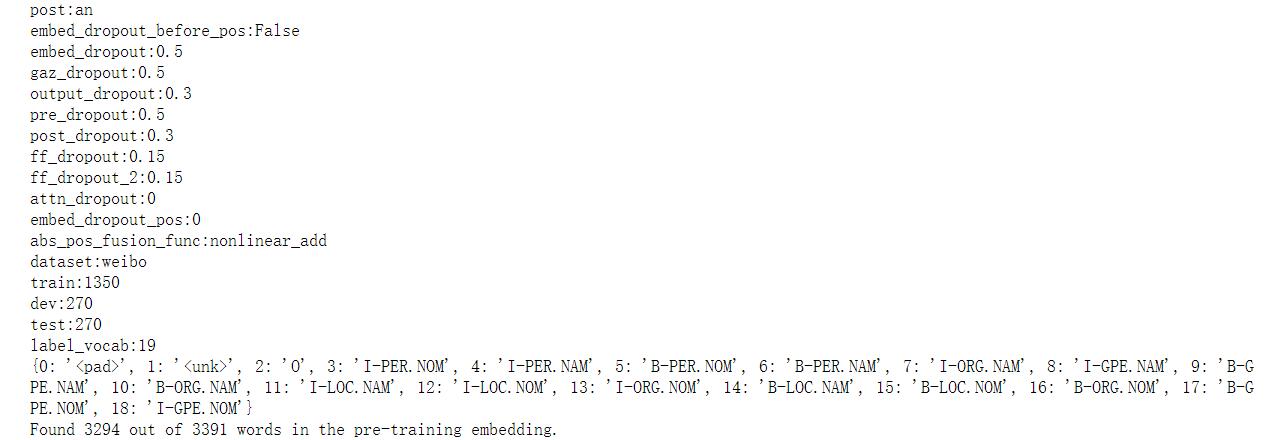
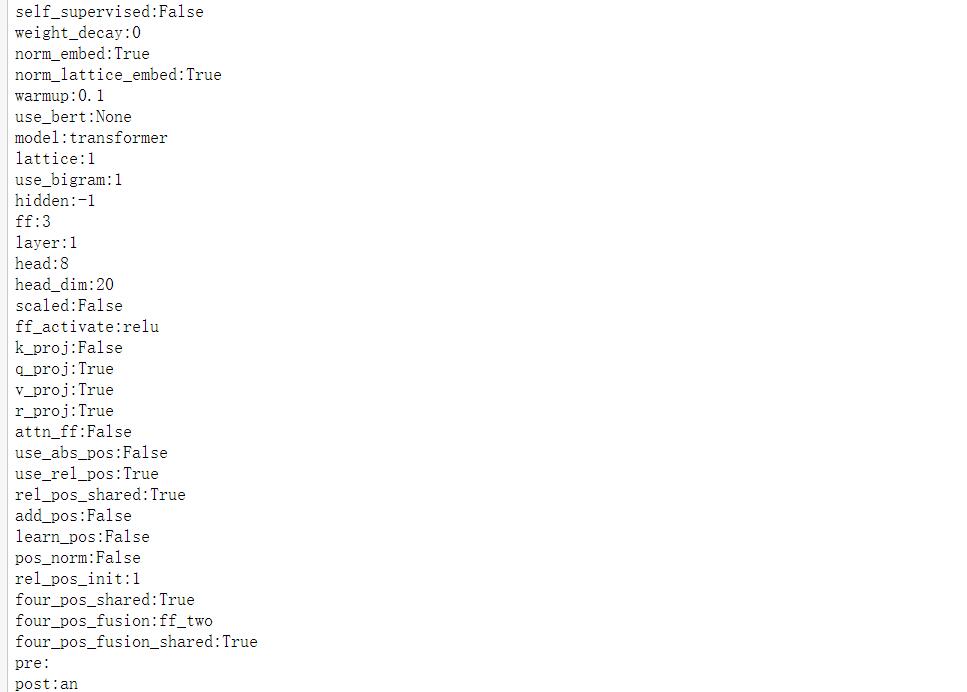
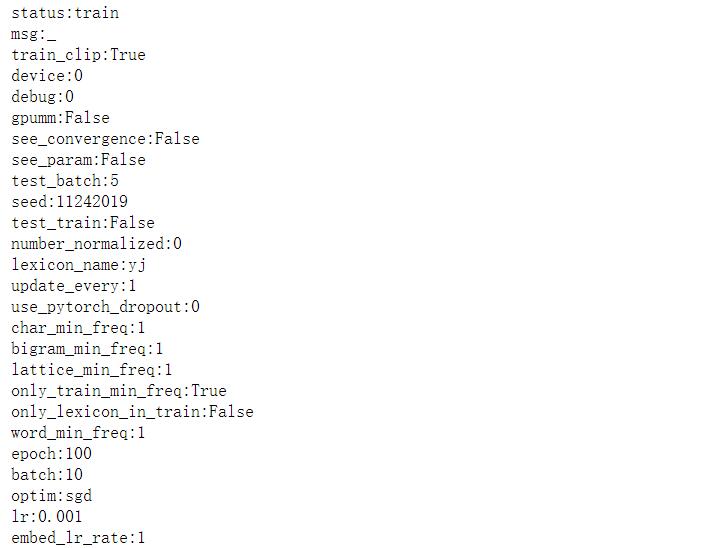
*#  ('中国致公党', 'ORG'),*

*#  ('中国共产党中央委员会', 'ORG'),*

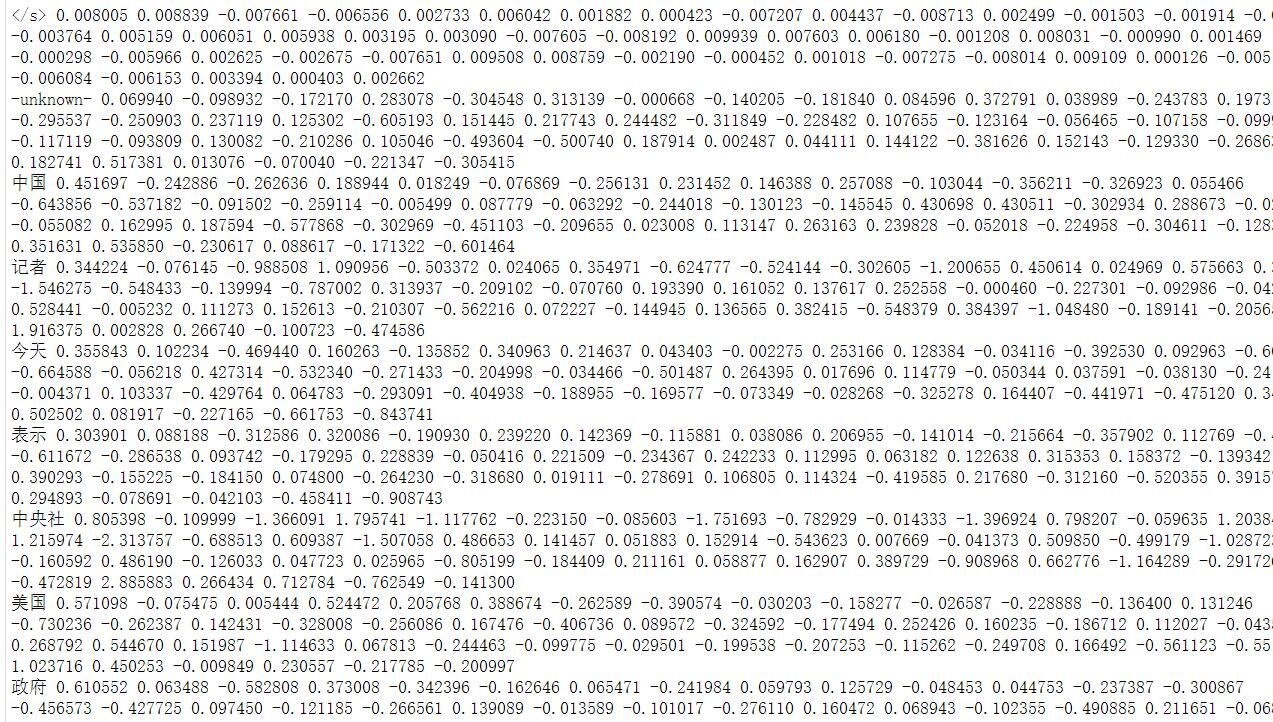
*#  ('致公党', 'ORG')]*

## 5.运行结果

V0版本的运行结果



preprocess的导出文件



# 四 总结反思

## 成员分工

赵睿：实验报告撰写 论文搜集分析

马昌盛：代码调试和运行 论文分析

严志涵：实验报告撰写 资料收集整理

## 实验感想

经过这次项目我们获益匪浅，收获良多。

关于团队的优点，首先就是团队意识，在开始项目前对彼此的能力进行一定的了解，根据自己搜集的资料交流想法，也能更好地明确分工，效率也会随之提高。在这次项目中大家的发挥都非常不错，成员之间也能互相学习，互相给人的感觉也很可靠。

关于团队的不足，我们这次时间安排的不是很合理，导致我们任务完成有些粗糙，这也可能是我们第一次合作，能力有限导致的。最开始的定题也花去了不少时间。

就做项目的经验而言，我们经过讨论，一个项目首先是要有好的想法，然后才是方案设计，不要一股脑地直接就上手码代码。在实施中，不要为了学习某种方法，而去应用它； 也不要因为不熟悉，就直接舍弃某种方案，网上参考别人用过的，学习学习都能大概掌握，还能加经验值，做项目也是一个学习的过程。实施过程中遇到问题，也不能直接规避，贪图一时方便后面可能越来越复杂，最后直接掉坑里，项目的实施要以全局考虑。好的编译习惯很重要，编写代码的过程中，一串代码的往往需要反复的测试，执行，才能完善，在编译报错的情况下，又需要反复的去检查。而也有可能在编译通过之后，执行后的结果并不是想要的结果。这就可能是代码中出了其他的错误，又需要反复的检查与改动。所以，这要求我们在编写代码的过程中要更加的细心。

这次项目了解了一些课上没有学过的知识，再确定选题的过程中也对NLP的应用方面有了一定的了解，既增长了见识，又锻炼了能力，对小组合作的模式也更加适应。