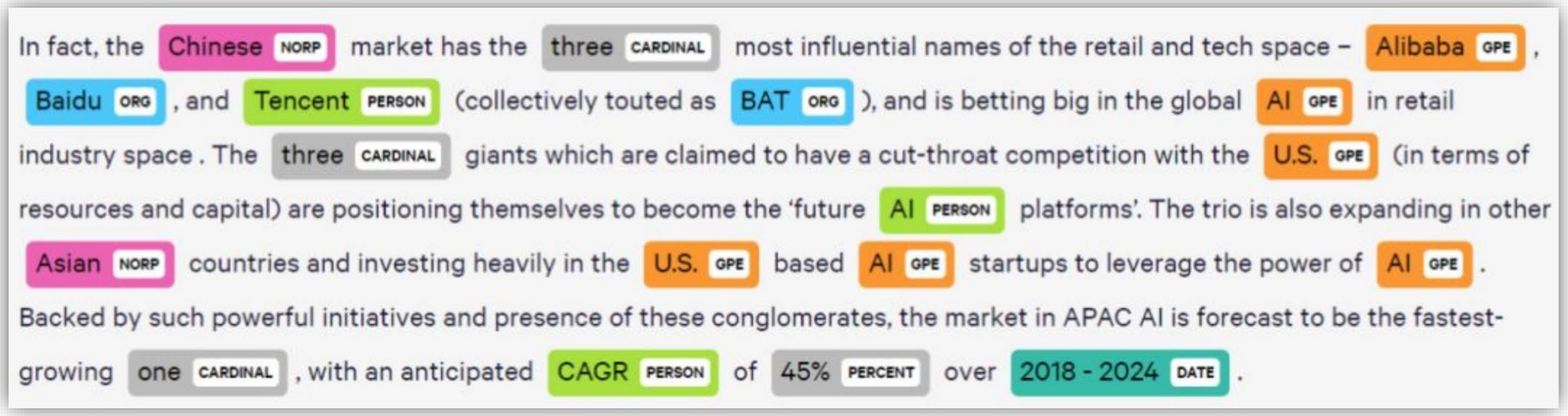
命名实体识别(NER)

汇报人：郑逸潇
汇报时间：2022.6.16

任务描述



难点：

1、数据集的标注
2、实体歧义：一些人名可能和地名一样
3、实体数量庞大
4、实体之间的边界难以确定（分词）：
   广州市长隆公园（其中的市长可能会被认为是一个实体）
5、嵌套重叠的实体：
   北京天安门可以分为（北京天安门/北京/天安门）三个实体
6、不连续的实体：多出现在英文中(I am having aching in legs and shoulders)
7、数据日新月异的增长

命名实体识别方法

- 基于规则

构建词典，一一对比句子中的词，看是否有词典中的词，这也是最朴素的办法，但是除了难点1，其他问题都没有解决。

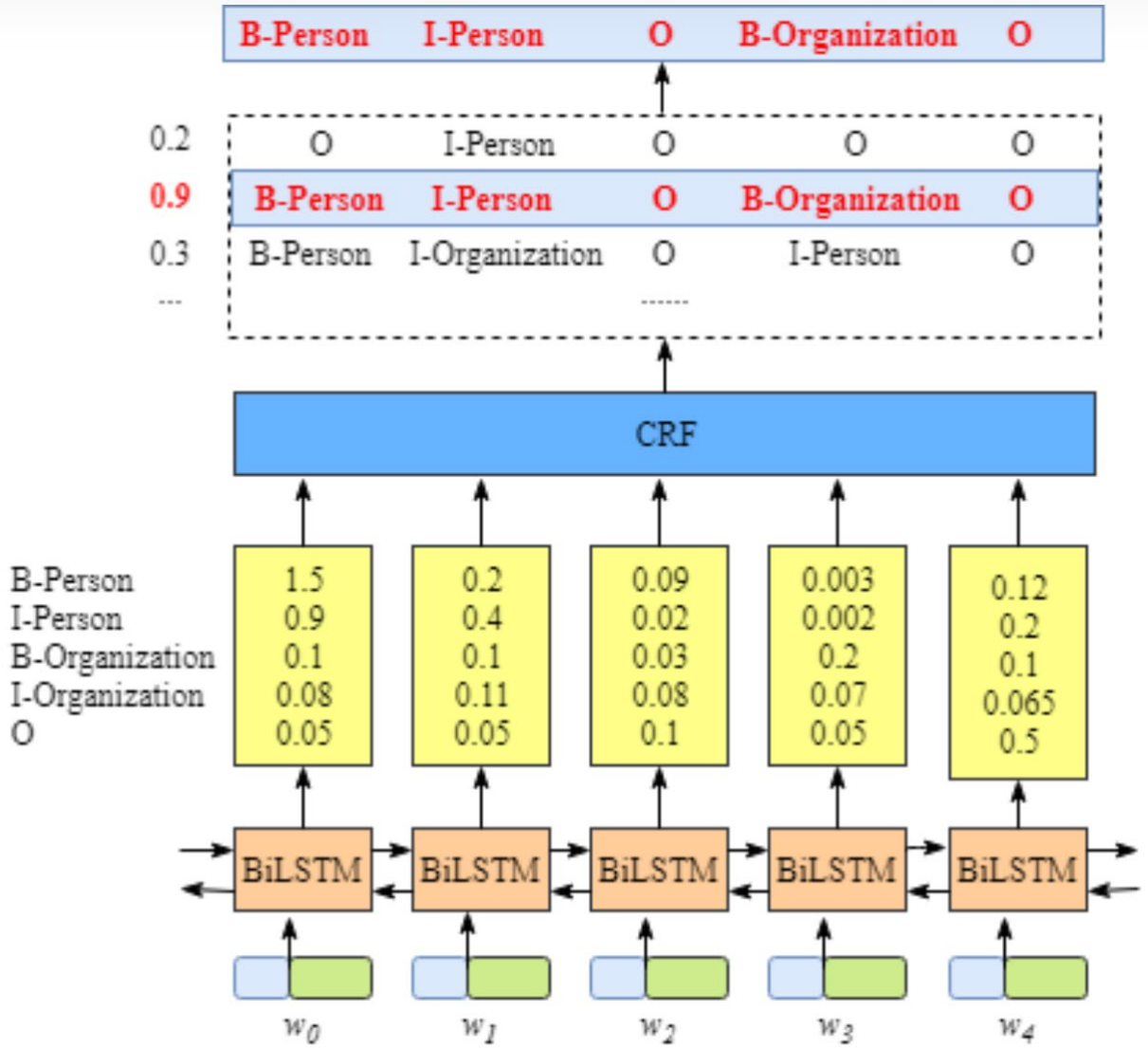
- 基于统计

HMM，MEMM、CRF、SVM等方法，拥有较好的理论基础，也一定程度解决了边界识别的问题，但是很难消除歧义，尽管CRF已经考虑了上下文关系，但是可能由于模型变量之间的依赖关系是人工定义的，效果不是很理想。

- 基于深度学习

LSTM/BiLSTM/BERT+BiLSTM→...一些结合图模型等...→W2NER等方法，LSTM方法较好的解决了歧义问题，BERT较好的解决了实体数据较大的问题，后面一系列新方法就是为了更好的解决边界问题和嵌套问题。

BiLSTM+CRF



- 整个模型分为：

WordEmbbeding层，将词(在中文里即单个字)以词向量的形式表征。

BiLSTM层，双向LSTM，一种RNN变体，可以很好的处理序列数据，提取出序列数据的隐藏信息。

全连接层，将BiLSTM的输出作为参数进行分类，这里实际上就可以输出标签了。

CRF层，将全连接层的输出视作观测序列，输出标注序列，加入CRF层能一定程度解决深度学习网络输出的标注序列不够“规则”的问题，如：一个BIOES序列BIIOOS可能会被边界划分得稍微规则点，即BIEOOS。实际的效果需要视数据集而定。

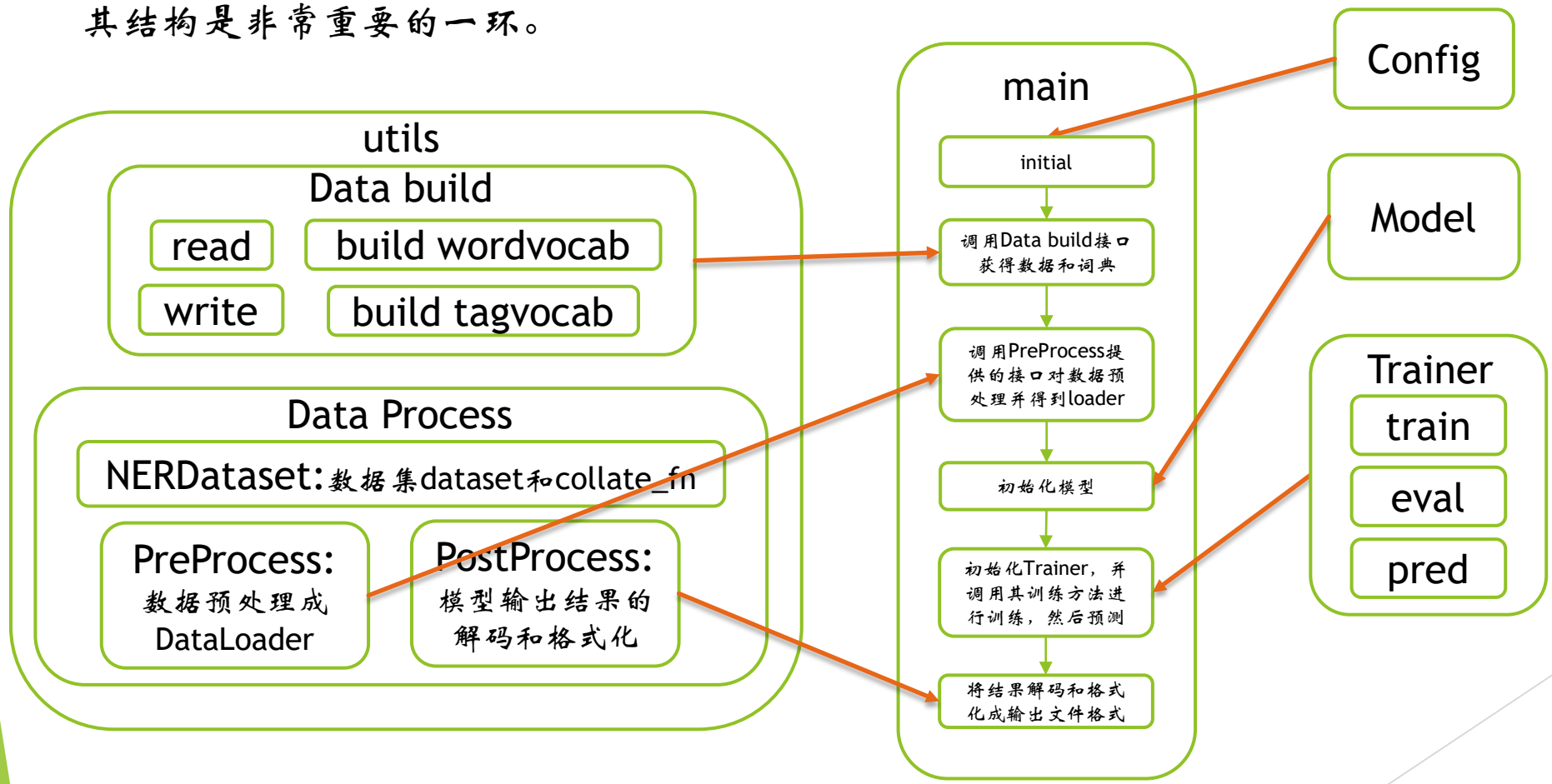
BiLSTM+CRF实现

Advanced: Making Dynamic Decisions and the Bi-LSTM CRF — PyTorch Tutorials 1.11.0+cu102 documentation

- 这里可以参考Pytorch官方的实现方法（也是老师课上讲的），在官方的代码中，CRF层代码是手动实现的，显得比较冗长，同时他是一个一个句子训练的，速度也比较慢。
- 因此我的第一个版本就是改进Pytorch官方的代码，调用了第三方库来实现CRF，同时加入了Dataset和Dataloader使之加速训练。
- 这个版本目前所需的知识完全就是已经学过的知识，所以在写代码的时候更改了很大一部分Pytorch官方的写法（代码框架在下一页）
- 当然由于都是已经学过的知识，做出来的算是一定意义上“过时”的模型（不是指知识技术真正过时，而是随着需求的进一步提高，模型已经不满足需求），结果肯定是不太好的，这份代码最后提交的F1只有0.45左右。

BiLSTM+CRF框架结构

作为一个期末项目，而且这次提供的框架也不像之前那样基本可以直接用，其结构是非常重要的一环。



BERT+BiLSTM+CRF

Hugging Face Transformers

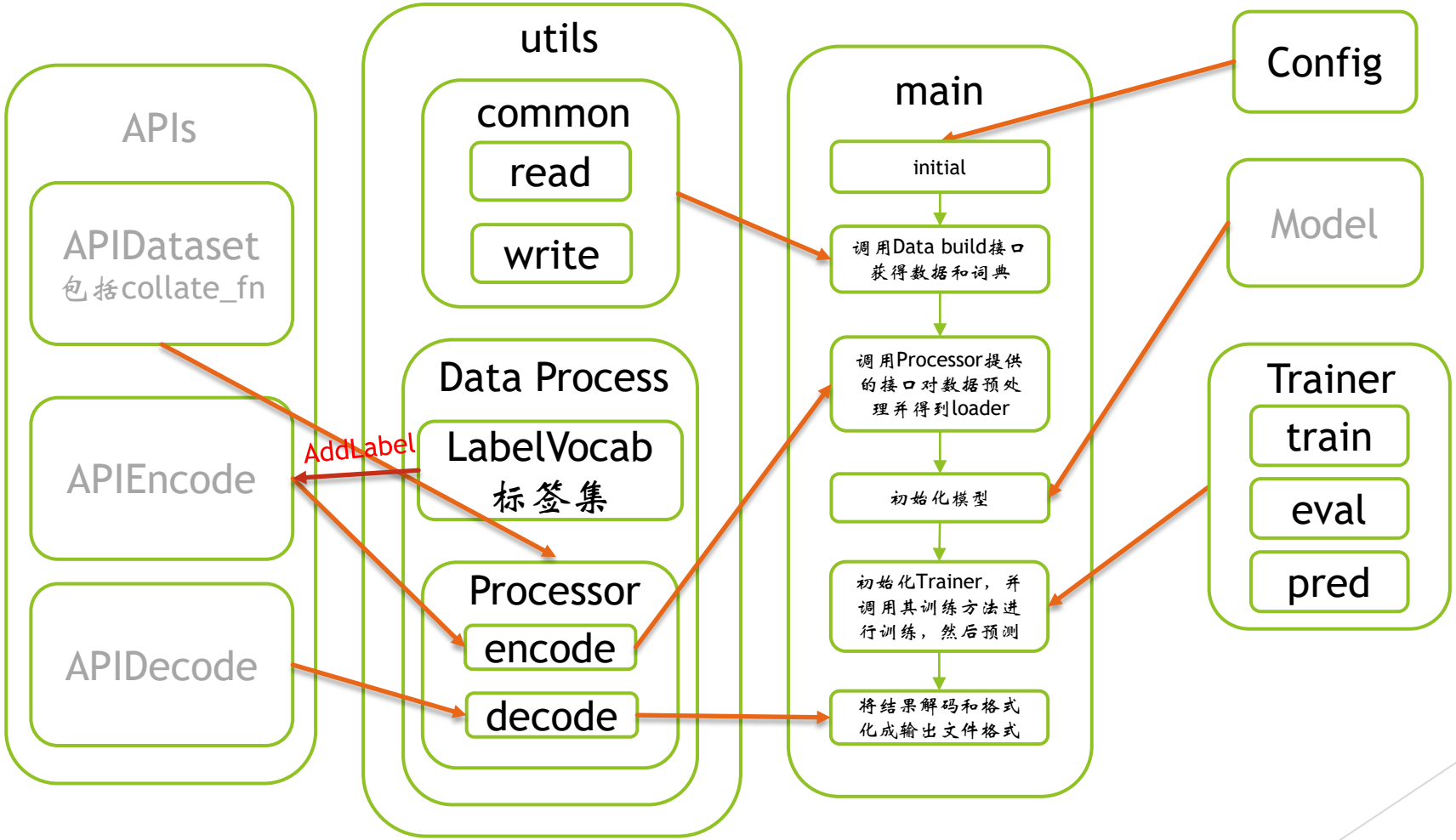
预训练模型，相当于数据增强，由于BERT的Embbeding本身就带有一定语义信息，效果可能要比直接训练随机Embbeding层要好，最后实际效果能达到66%左右F1。

- 实现BERT+BiLSTM+CRF相当于将BERTModel当成一种Embedding层使用即可。
- 在实现包含BERT的模型代码过程中，原本的代码结构修改起来不是很友好，而且模型是不断变化的，对于不同模型的预处理也是有所不同的，因此我稍微修改了原本的代码结构，使之更解耦合，方便修改模型（下一页）。
- 使用BERT训练的时候还需要注意BERT那一层是只需要Finetune的，而剩下的层数BiLSTM和CRF则要正常训练，因此这里优化器的训练参数不能像之前那样直接model.parameters()就行，需要对不同层使用不同的参数。

```
bert_params = set(self.model.bert.parameters())
other_params = list(set(self.model.parameters()) - bert_params)
no_decay = ['bias', 'LayerNorm.weight']
params = [
    {'params': [p for n, p in model.bert.named_parameters() if not any(nd in n for nd in no_decay)],
     'lr': config.bert_learning_rate, 'weight_decay': config.weight_decay},
    {'params': [p for n, p in model.bert.named_parameters() if any(nd in n for nd in no_decay)],
     'lr': config.bert_learning_rate, 'weight_decay': 0.0},
    {'params': other_params,
     'lr': config.learning_rate, 'weight_decay': config.weight_decay},
]
```

BERT+BiLSTM+CRF结构

一般替换灰色部分就可以迁移另一个模型使用，有时需要更改Config/Trainer。



思考

前面提到的模型能较好地解决前四个难点：标注、歧义、数据量、边界，但是重叠嵌套的词还是无法解决，导致指标一直上不去。

- 这里我首先想到的是暴力遍历法，既然输出了标注，那么将每个B到O之间的序列中所有可能的情况全部遍历。

```python
# 判断当前词的label(label指的是最终输出json的类别) 输入词的idx列表和对应tag的idx列表 输出类别
def distinguish(self, sentenceidx: list, tagidx: list) -> str:
    if self.classifier is None:
        return self.idx2tag.get(tagidx[-1], Config.NON_S_NONE).split(Config.TAG_SPLITBY, 1)[1]
    else:
        return self.idx2label.get(self.classifier(sentenceidx))

# 由[B(……I……B……)]O组成的短语，其中可能包含多个实体，此处列举几乎所有可能
def pharse2entities(self, start, pharseidx, tagidx):
    assert len(pharseidx) == len(tagidx)
    entities = []
    def apd(b, e):
        entities.append((self.distinguish(pharseidx[b: e], tagidx[b: e]), (start + b, start + e)))
    apd(0, len(tagidx))
    tagidx.append(Config.NONE_NUM)
    last_tagidx, last_position = tagidx[0], 0
    for idx in range(1, len(tagidx)):
        current_tag = self.idx2tag.get(tagidx[idx], Config.NON_S_NONE).split(Config.TAG_SPLITBY, 1)[0]
        current_tagidx = tagidx[idx]
        if current_tagidx == last_tagidx and current_tag == Config.INNER_TAG: pass
        if current_tagidx != last_tagidx and current_tag == Config.INNER_TAG:
            apd(last_position, idx)
        if current_tag == Config.BEGIN_TAG or current_tag == Config.NON_TAG:
            apd(last_position, idx)
            last_tagidx, last_position = current_tagidx, idx
    return entities
```

效果确实不理想，
这样做虽然能大大提高召回率，
但是精确率下降得离谱。

Nested/Overlapped NER

Neural architectures for nested NER through linearization
J Straková, M Straka, J Hajič - arXiv preprint arXiv:1908.06926, 2019 - arxiv.org
… We propose two neural network architectures for nested named entity recognition (NER), a … the nested labels using a linearized scheme. In our first proposed approach, the nested …
☆ 保存 99 引用 被引用次数：157 相关文章 所有 9 个版本 »

A span-based model for joint overlapped and discontinuous named entity recognition
F Li, ZC Lin, M Zhang, D Ji - arXiv preprint arXiv:2106.14373, 2021 - arxiv.org
… For overlapped NER, the earliest model to our … overlapped and discontinuous NER. In this work, we also aim to design a competitive model for both overlapped and discontinuous NER. …
☆ 保存 99 引用 被引用次数：11 相关文章 所有 5 个版本 »

Rethinking boundaries: End-to-end recognition of discontinuous mentions with pointer networks
H Fei, D Ji, B Li, Y Liu, Y Ren, F Li - … of the AAAI conference on artificial …, 2021 - ojs.aaai.org
… are often partially overlapped with each other, discontinuous NER covers nested NER but is … dataset, the recognition for overlapped mentions is more successful than that for non-…
☆ 保存 99 引用 被引用次数：5 相关文章 所有 2 个版本 »

[PDF] Unified named entity recognition as word-word relation classification
J Li, H Fei, J Liu, S Wu, M Zhang, C Teng, D Ji… - arXiv preprint arXiv …, 2021 - aaai.org
… as word-word relation classification, namely W2NER. The architecture resolves the kernel … Based on the W2NER scheme we develop a neural framework, in which the unified NER is …
☆ 保存 99 引用 被引用次数：2 相关文章 所有 4 个版本 »

经过在Google scholar中的搜索
发现了一些可以解决重叠嵌套词/不连续实体的方法
好几篇都基本是FeiLi/HaoLi等人的
这里就提供一下较新的，
并有代码库提供的两份论文：

A Span-Based Model for Joint Overlapped and Discontinuous Named Entity Recognition
https://arxiv.org/pdf/2106.14373.pdf
https://github.com/foxlf823/sodner

Unified Named Entity Recognition as Word-Word Relation Classification
https://arxiv.org/pdf/2112.10070.pdf
https://github.com/ljynlp/W2NER

Word-Word NER （W2NER）

最后我采用了W2NER的模型进行训练与预测。

- W2NER与之前的NER不同，它将NER表征为词-词之间的关系，在论文中为无关系、NNW(Next-Neighboring-Word)关系和THW(Tail-Head-Word-*)关系。
- 模型的输出也不是一串序列，而是一张关系表(有向图)，而有向图中的每一个环就是一个实体。

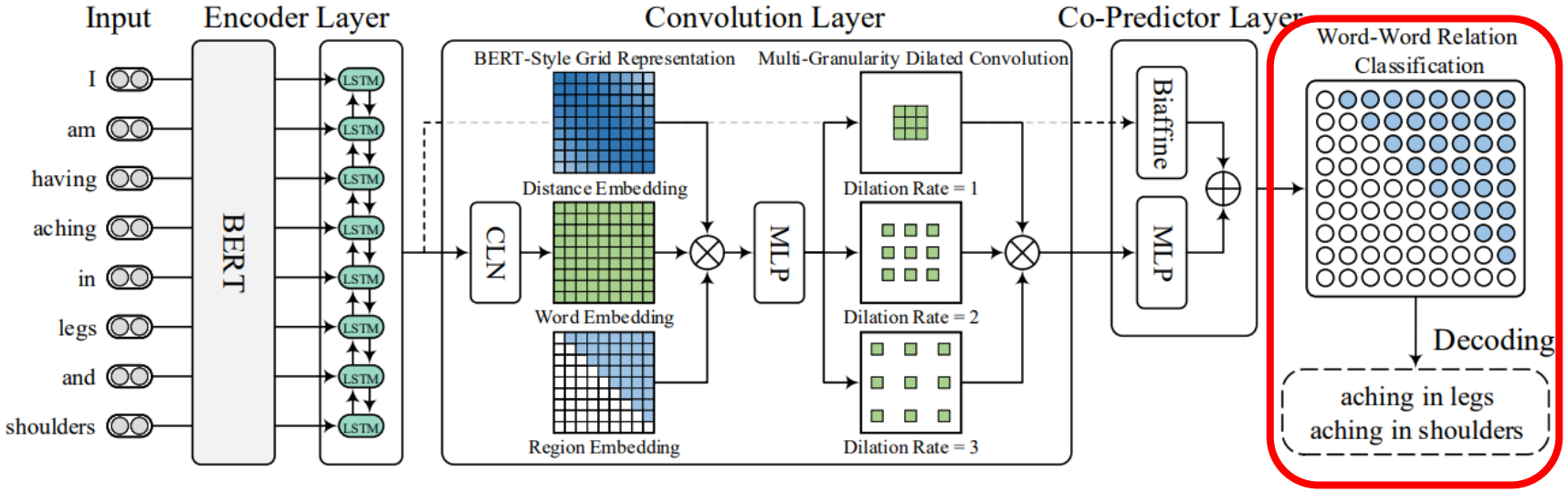


Figure 3: Overall NER architecture. CLN and MLP represent conditional layer normalization and multi-layer perceptron. ⊕ and ⊗ represent element-wise addition and concatenation operations.

W2NER Decode

对于每条边我们有标记：PAD 0，SUC 1，PER 2， GPE 3，...

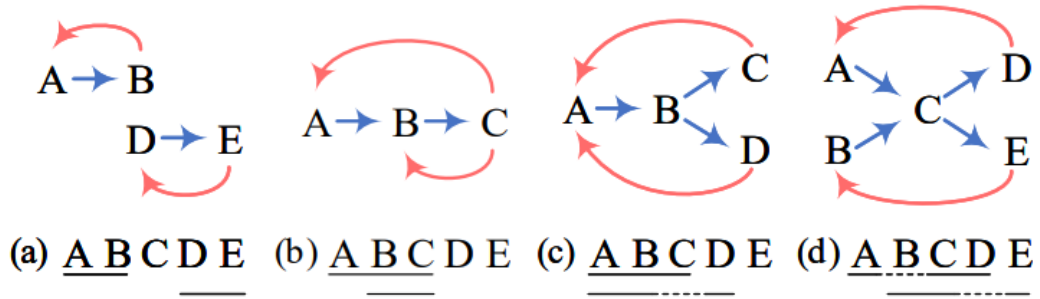


Figure 4: Four decoding cases for the word sequence "ABCDE". (a) "AB" and "DE" are flat entities. (b) The flat entity "BC" is nested in "ABC". (c) The entity "ABC" is overlapped with a discontinuous entity "ABD". (d) Two discontinuous entities "ACD" and "BCE" are overlapped. The blue and red arrows indicate NNW and THW relations.

每一条边可以表示不同的类别，在论文中红色边有类别之分，即在邻接矩阵中数值是不同的，蓝色边为固定类别SUC(表连续随后的)。

所以一个句子：...A...B...中，A和B的关系，即A和B之间的边只有如下几种可能：

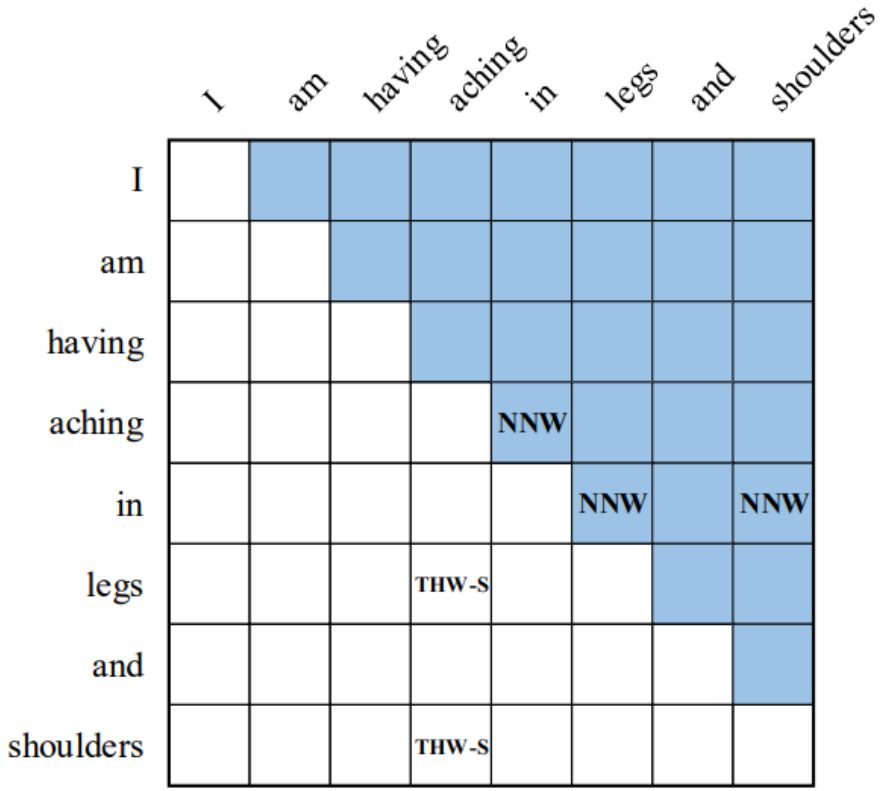
A→B：SUC 1 或 PAD 0 表示B是否为A的next neighbor。(代码中的判断是是否为1)

B→A：<=1 或 >1每个大于1的数字都有对应的类别。(代码中的判断是是否大于1)

所以类别是由THW来决定的。不成环的则舍弃。

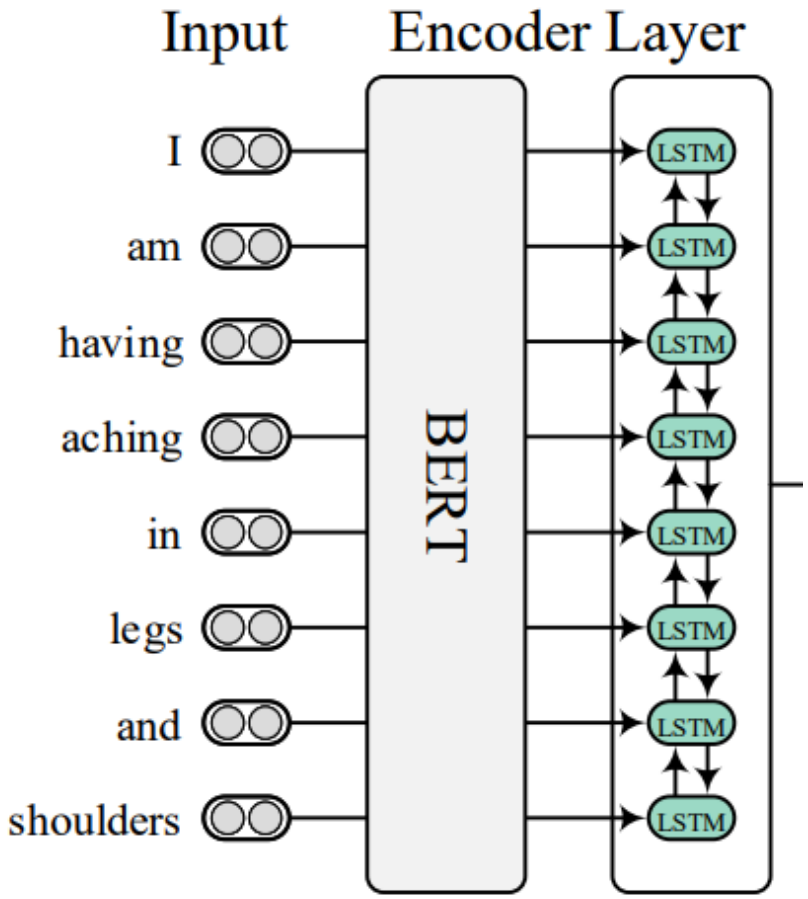
W2NER Decode

例子：



aching in legs
aching in shoulders

W2NER Encoder Layer



Encoder Layer事实上就是BERT + BiLSTM

BiLSTM的隐层输出作为了 Convolution Layer层的输入。

这里和之前的用法基本相同，不赘述。

W2NER Convolution Layer

Conditional Layer Normalization

$$\mathbf{V}_{ij} = \mathrm{CLN}(\mathbf{h}_i, \mathbf{h}_j) = \gamma_{ij} \odot (\frac{\mathbf{h}_j - \mu}{\sigma}) + \lambda_{ij}\,, \quad (1)$$

where $\mathbf{h}_i$ is the condition to generate the gain parameter $\gamma_{ij} = \mathbf{W}_\alpha \mathbf{h}_i + \mathbf{b}_\alpha$ and bias $\lambda_{ij} = \mathbf{W}_\beta \mathbf{h}_i + \mathbf{b}_\beta$ of layer normalization. μ and σ are the mean and standard deviation across the elements of $\mathbf{h}_j$, denoted as:

$$\mu = \frac{1}{d_h}\sum_{k=1}^{d_h} h_{jk}, \quad \sigma = \sqrt{\frac{1}{d_h}\sum_{k=1}^{d_h}(h_{jk} - \mu)^2}\,. \quad (2)$$

where h_{jk} denotes the k-th dimension of $\mathbf{h}_j$.

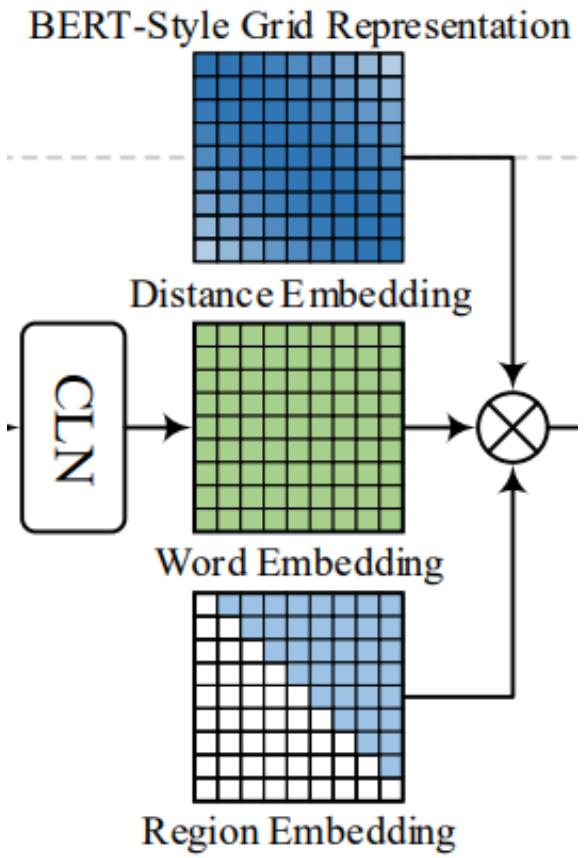
对于每个隐层向量h_i，得到一个增益系数γ_{ij}和偏差λ_{ij}，以这两个参数来对每个h_j进行特殊的LayerNorm，就可以形成一个seq×seq维度的矩阵(实际中是seq×seq×emb的张量)

这样就将一个句子的LSTM隐层二维化了。

该方法相当于把h_i当作一个condition，然后计算出与每个h_j的关联值，然后再看γ_{ij}和λ_{ij}的计算，感觉这个方法和self-attention有些类似。

W2NER Convolution Layer

BERT-Style Grid Representation



这一层参考BERT的实现：分为了与距离相关的二维矩阵、CLN的输出矩阵、region矩阵(用来区分上下三角)。

将这三者结合到一起来作为下一层输入，有利于后面的分类器将位置信息、区域信息(这里应该是句子从前到后的顺序)和语义信息同时考虑。

具体实现的时候，距离矩阵Distance Matrix的值是以根号距离来描述，应该是为了防止线性增加的距离差距过大导致后面分类过分依赖距离

具体实现的时候，结合这三者的方法就是直接cat到一起作为下一层输入。

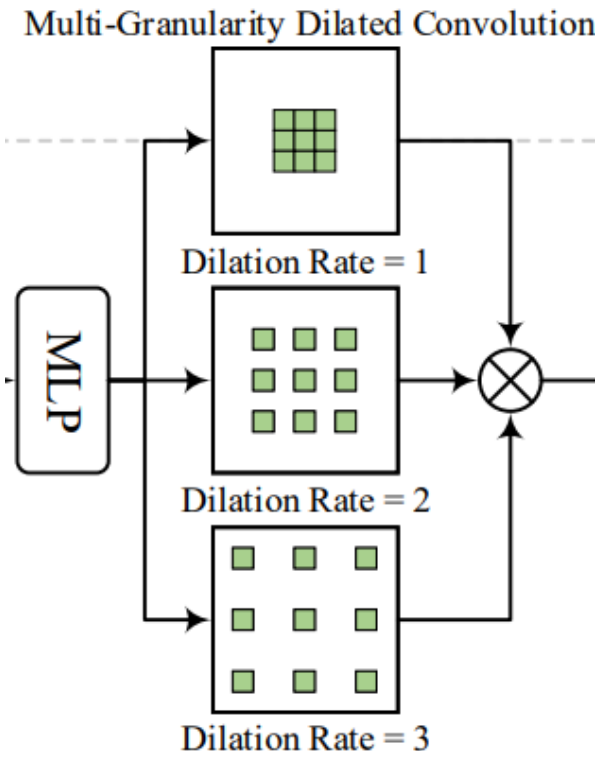
W2NER Convolution Layer

Multi-Granularity Dilated Convolution

Multi-Granularity Dilated Convolution Motivated by TextCNN (Kim 2014), we adopt multiple 2-dimensional dilated convolutions (DConv) with different dilation rates l (e.g., $l \in [1, 2, 3]$) to capture the interactions between the words with different distances, because our model is to predict the relations between these words. The calculation in one dilated convolution can be formulated as:

$$\mathbf{Q}^l = \sigma(\mathrm{DConv}_l(\mathbf{C})), \tag{4}$$

where $\mathbf{Q}^l \in \mathbb{R}^{N \times N \times d_c}$ denotes the output of the dilation convolution with the dilation rate l, σ is the GELU activation function (Hendrycks and Gimpel 2016). After that, we can obtain the final word-pair grid representation $\mathbf{Q} = [\mathbf{Q}^1, \mathbf{Q}^2, \mathbf{Q}^3] \in \mathbb{R}^{N \times N \times 3d_c}$.



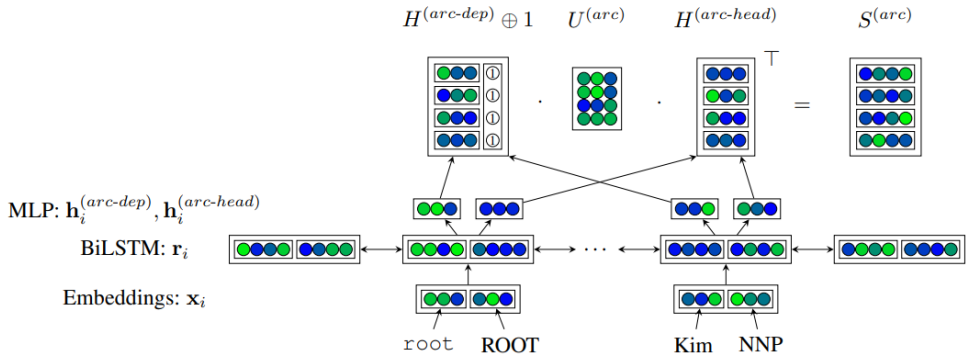
对上一层的输出进行多粒度卷积，让最后的模型能有不同程度感受野。

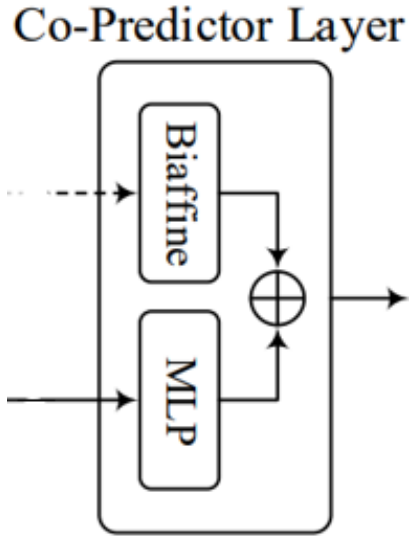
（这里有个疑问，LSTM模型在一定程度上已经包括了上下文的信息，为什么还需要多粒度卷积呢？可能是作者想要显式地捕获相邻词之间的关系，LSTM是通过隐式的隐层捕获的关系）

W2NER Co-Predictor Layer

这一层将BiLSTM的输出和Multi-Granularity Dilated Convolution的输出结合在一起进行预测，最后的到关系表（有向图）。

其一分类器使用了Biaffine方法，这主要是一种研究依存句法分析方法，作者将其套用在NER上可能是想借助语法来更好的判断哪些词可能是实体（像主谓宾中的谓语一般来说不会是实体）。





再结合上一层的输出使用全连接输出最后的关系表。

实际实现中是直接二者相加。

这部分没有深入去研究，只是初步涉略，且经尝试，去掉Biaffine 单一predict在本次实验数据集中也能达到不错的结果(验证集0.79 F1)

EVAL 9	F1	Precision	Recall
Label	0.4765	0.5031	0.5153
Entity	0.7944	0.7897	0.7992

数据预处理(APIEncode)

分析完上述W2NER模型之后，我们可以得知模型需要哪些输入：
X：BERT tokenizer后的句子，距离矩阵，region矩阵；y：label矩阵

0	-1	-2	-3
1	0	-1	-2
2	1	0	-1
3	2	1	0

19	10	11	12
1	19	10	11
2	1	19	10
2	2	1	19

Distance Matrix

- X部分：

  BERT tokenizer后的句子，也是BERT模型所需的输入(bert input_ids)，注意需要加入[CLS]和[SEP]，调用tokenizer的方法即可；

  距离矩阵(dist inputs)，就是将单词与单词之间的距离用某个值来刻画，如Distance Matrix，作为BERT-Style Grid Representation的其中一部分输入。

  region矩阵，一个下三角矩阵(具体实现中调用了torch.tril)

  参考BERT的思想，上面这些在最终实现的时候都会变成带有Embedding的张量。

- y部分：

  通过原有的数据集标注，

  构建一个关系表。

```
if instance.get(Keys.entity_mentions, None) is not None:
    for entity in instance[Keys.entity_mentions]:
        b, e, type = entity[Keys.start], entity[Keys.end], entity[Keys.type]
        if b < 0: continue
        if e > len(sentence): e = len(sentence)
        index = list(range(b, e))
        for i in range(len(index)):
            if i + 1 >= len(index):
                break
            _grid_labels[index[i], index[i + 1]] = vocab.label_to_id(vocab.SUC)
        _grid_labels[index[-1], index[0]] = vocab.label_to_id(type)
        _entity_text.append(convert_index_to_text(index, vocab.label_to_id(type)))
```

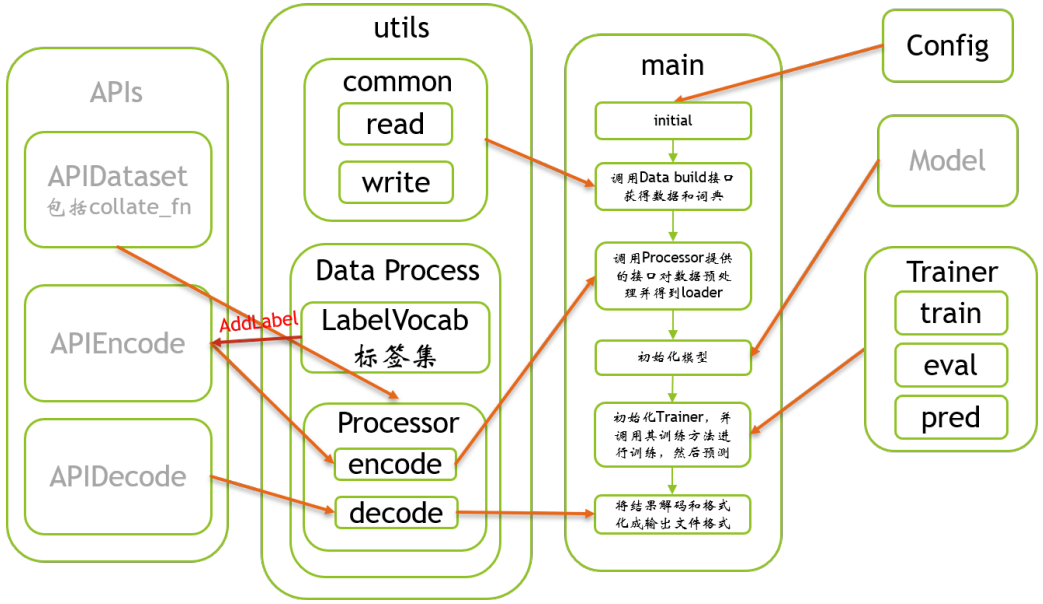
NNW

THW

注意：老师给出的数据是有一些问题的，比如：有些start和end小于0，有些end超过了句子长度，同时，在Pytorch的GPU训练中句子长度默认是不能超过512的（可以调整）。

W2NER 实现与结果

- 具体实现就是参考GitHub代码，将每一部分分割成之前的结构，这种结构也将各个接口定义的较为清楚，方便观察每一步的数据输入输出分别是什么样的。



PPT19

PPT12

```
2022-06-12 07:50:41 - INFO: EVAL Label F1 [0.99956962 0.96680706 0.84552846 0.8        0.91431793 0.89115646
 1.         0.73913043 0.89       0.79591837 0.         0.75958188
 0.78431373 0.65882353 0.74594595 0.77777778 0.77777778 0.61111111
 0.75       0.87804878 0.8691796  0.66666667 0.61176471 0.74025974
 0.5        0.67532468 0.62857143 0.77777778 0.83333333 0.52631579
 0.57142857 0.77419355 0.55       0.66666667 0.69230769 0.66666667
 0.46153846 1.         0.         0.         0.83018868 0.84444444
 0.66666667 0.45714286 0.71428571 0.66666667 0.73170732 0.66666667
 0.4        0.66666667 0.4        1.         1.        ]
2022-06-12 07:50:41 - INFO:
+--------+--------+-----------+--------+
| EVAL 9 |   F1   | Precision | Recall |
+--------+--------+-----------+--------+
| Label  | 0.6951 |   0.7281  | 0.7114 |
| Entity | 0.8257 |   0.8234  | 0.8280 |
+--------+--------+-----------+--------+
```

这里第0个是PAD也就是无关系，第1个是SUC就是NNW，剩下的都是THW（包含类别）

由于时间有限，论文和代码中还有很多不懂的地方，欢迎大家一起讨论。

4	<10192100406郑逸潇	28	2022/06/12 15:58:21	0.810963 (4)	0.816679 (7)	0.816415 (10)

事实上我们方便调试的话度量(p r f)应该也可以用一个api来更解耦(APIMetric)

Future NER

- 我们可以看到W2NER已经相当好地解决了重叠嵌套的词，不连续的实体。
- 目前还未解决的就是一个个新产生的实体，由于没有数据训练，模型可能无法识别出从没出现过的实体，只能不断地喂给模型新的数据。
- 针对不同领域的命名实体识别可能会稍有不同，可以特化某一领域进行NER，提高正确率和召回率。