流水的NLP铁打的NER：命名实体识别实践与探索

原创 王岳王院长 AI小白入门 2020-08-16

作者：王岳王院长

知乎：https://www.zhihu.com/people/wang-yue-40-21

github：https://github.com/wavewangyue

编辑：yuquanle

前言

最近在做命名实体识别（Named Entity Recognition，NER）的工作，也就是序列标注（Sequence Tagging），老 NLP task 了，就是从一段文本中抽取到找到任何你想要的东西，可能是某个字，某个词，或者某个短语

为什么说流水的NLP铁打的NER？NLP四大任务嘛，分类、生成、序列标注、句子对标注。分类任务，面太广了，万物皆可分类，各种方法层出不穷；句子对标注，经常是体现人工智能（zhang）对人类语言理解能力的标准秤，孪生网络、DSSM、ESIM 各种模型一年年也是秀的飞起；生成任务，目前人工智障 NLP 能力的天花板，虽然经常会处在说不出来人话的状态，但也不断吸引 CopyNet、VAE、GAN 各类选手前来挑战；唯有序列标注，数年如一日，不忘初心，原地踏步，到现在一提到 NER，还是会一下子只想到 LSTM-CRF，铁打不动的模型，没得挑也不用挑，用就完事了，不用就是不给面子

虽然之前也做过 NER，但是想细致地捋一下，看一下自从有了 LSTM-CRF 之后，NER 在做些什么，顺便记录一下最近的工作，中间有些经验和想法，有什么就记点什么

因为能力有限，还是跟之前一样，就少讲理论少放公式，多画模型图多放代码，还是主要从工程实现角度记录和分享下经验，也记录一些个人探索过程。如果有新人苦于不知道怎么实现一个 NER 模型，不知道 LSTM-CRF、BERT-CRF 怎么写，看到代码之后便可以原地起飞，从此打开新世界的大门；或者有老 NLPer 从我的某段探索过程里感觉还挺有意思的，那我就太开心了。就这样

还是先放结论

命名实体识别虽然是一个历史悠久的老任务了，但是自从2015年有人使用了BI-LSTM-CRF模型之后，这个模型和这个任务简直是郎才女貌，天造地设，轮不到任何妖怪来反对。直到后来出现了BERT。在这里放两个问题：

> 2015-2019年，BERT出现之前4年的时间，命名实体识别就只有 BI-LSTM-CRF 了吗？
>
> 2019年BERT出现之后，命名实体识别就只有 BERT-CRF（或者 BERT-LSTM-CRF）了吗？

经过我不完善也不成熟的调研之后，好像的确是的，一个能打的都没有

既然模型打不动了，然后我找了找 ACL2020 做NER的论文，看看现在的NER还在做哪些事情，主要分几个方面：

> 多特征：实体识别不是一个特别复杂的任务，不需要太深入的模型，那么就是加特征，特征越多效果越好，所以字特征、词特征、词性特征、句法特征、KG表征等等的就一个个加吧，甚至有些中文 NER 任务里还加入了拼音特征、笔画特征。。？心有多大，特征就有多多
>
> 多任务：很多时候做 NER 的目的并不仅是为了 NER，而是服务于一个更大的目标，比如信息抽取、问答系统等等的，如果把整个大任务做一个端到端的模型，就需要做成一个多任务模型，把 NER 作为其中一个子任务；另外，如果单纯为了 NER，本身也可以做成多任务，比如实体类型多的时候，单独用一个任务来识别实体，另一个用来判断实体类型
>
> 时令大杂烩：把当下比较流行的深度学习话题或方法跟NER结合一下，比如结合强化学习的NER、结合 few-shot learning 的NER、结合多模态信息的NER、结合跨语种学习的NER等等的，具体就不提了

所以沿着上述思路，就在一个中文NER任务上做一些实践，写一些模型。都列在下面了，首先是 LSTM-CRF 和 BERT-CRF，然后就是几个多任务模型， Cascade 开头的（因为实体类型比较多，把NER拆成两个任务，一个用来识别实体，另一个用来判断实体类型），后面的几个模型里，WLF 指的是 Word Level Feature（即在原本字级别的序列标注任务上加入词级别的表征），WOL 指的是 Weight of Loss（即在loss函数方面通过设置权重来权衡Precision与Recall，以达到提高F1的目的），具体细节后面再讲

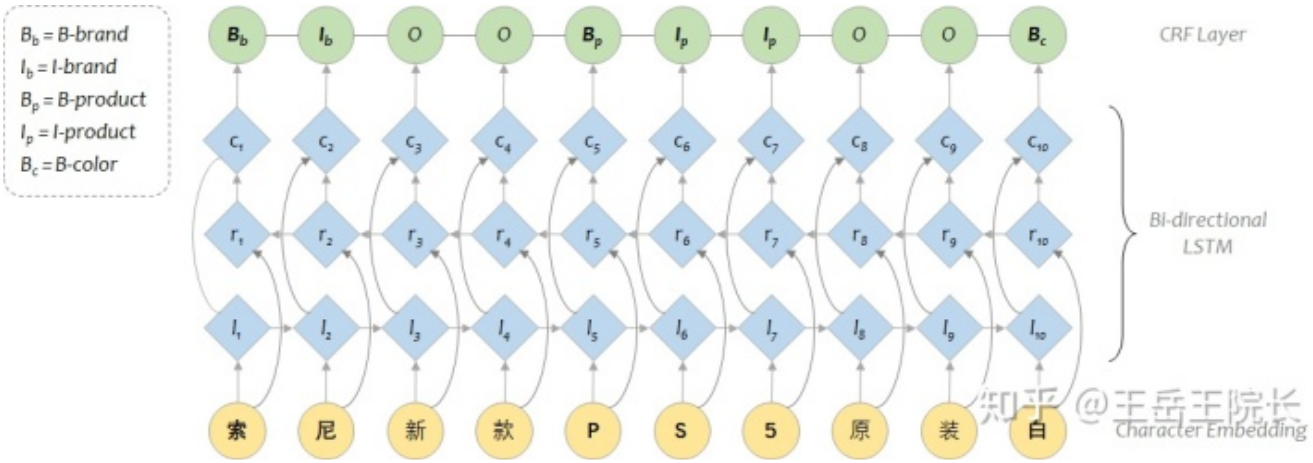
Model	Precision / Recall / F1	Training Time - 1 epoch, GPU: Tesla P40
BiLSTM	72.01 / 72.49 / 72.25	30 min
BiLSTM + CRF	82.59 / 78.36 / 80.42	400 min
BERT	86.35 / 76.14 / 80.92	140 min
BERT + CRF	86.23 / 77.43 / 81.60	410 min
BERT + BiLSTM + CRF	85.32 / 77.63 / 81.30	460 min
Cascade + BiLSTM +CRF	79.77 / 80.21 / 79.99	38 min
Cascade + BiLSTM + CRF + WLF	84.96 / 79.31 / 82.04	40 min
Cascade + BiLSTM + CRF + WLF + WOL	82.60 / 82.14 / 82.37	40 min
Cascade + BERT + CRF	87.83 / 79.28 / 83.33	130 min
Cascade + BERT + CRF + WOL	83.51 / 84.22 / 83.86	130 min

- 代码：上述所有模型的代码都在这里：https://github.com/wavewangyue/ner，带 BERT 的可以自己去下载BERT_CHINESE预训练的 ckpt 模型，然后解压到 bert_model 目录下
- 环境：Python3，Tensorflow1.12
- 数据：一个电商场景下商品标题中的实体识别，因为是工作中的数据，并且通过远程监督弱标注的质量也一般，完整数据就不放了。但是我 sample 了一些数据留在 git 里了，为了直接 git clone 完，代码原地就能跑，方便你我他

ok 下面正经开工

1. BI-LSTM+CRF

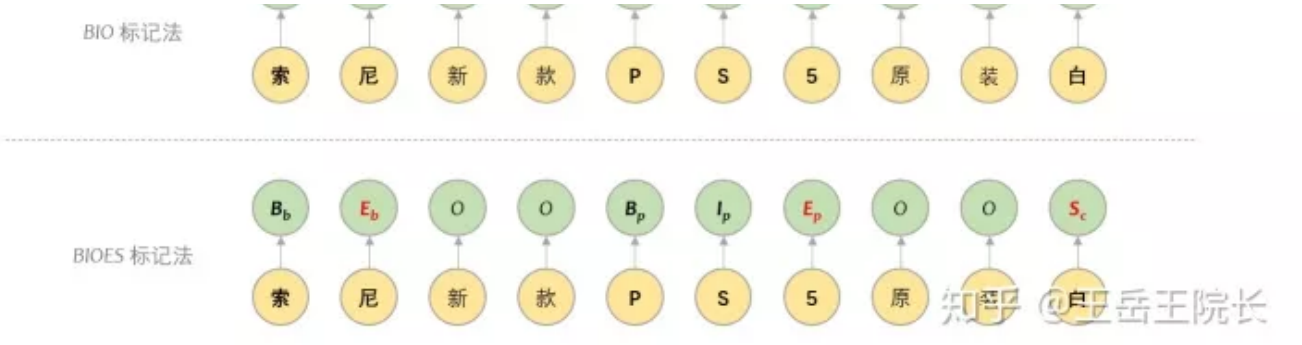
用纯 HMM 或者 CRF 做 NER 的话就不讲了，比较古老了。从 LSTM+CRF 开始讲起，应该是2015年被提出的模型[1]，模型架构在今天来看非常简单，直接上图



BI-LSTM 即 Bi-directional LSTM，也就是有两个 LSTM cell，一个从左往右跑得到第一层表征向量 l，一个从右往左跑得到第二层向量 r，然后两层向量加一起得到第三层向量 c

如果不使用CRF的话，这里就可以直接接一层全连接与softmax，输出结果了；如果用CRF的话，需要把 c 输入到 CRF 层中，经过 CRF 一通专业缜密的计算，它来决定最终的结果

这里说一下用于表示序列标注结果的 BIO 标记法。序列标注里标记法有很多，最主要的还是 BIO 与 BIOES 这两种。B 就是标记某个实体词的开始，I 表示某个实体词的中间，E 表示某个实体词的结束，S 表示这个实体词仅包含当前这一个字。区别很简单，看图就懂。一般实验效果上差别不大，有些时候用 BIOES 可能会有一内内的优势



另外，如果在某些场景下不考虑实体类别（比如问答系统），那就直接完事了，但是很多场景下需要同时考虑实体类别（比如事件抽取中需要抽取主体客体地点机构等等），那么就需要扩展 BIO 的 tag 列表，给每个“实体类型”都分配一个 B 与 I 的标签，例如用“B-brand”来代表“实体词的开始，且实体类型为品牌”。当实体类别过多时，BIOES 的标签列表规模可能就爆炸了

「基于 Tensorflow 来实现 LSTM+CRF 代码也很简单，直接上」

```
self.inputs_seq = tf.placeholder(tf.int32, [None, None], name="inputs_seq")
self.inputs_seq_len = tf.placeholder(tf.int32, [None], name="inputs_seq_len"
self.outputs_seq = tf.placeholder(tf.int32, [None, None], name='outputs_seq'

with tf.variable_scope('embedding_layer'):
    embedding_matrix = tf.get_variable("embedding_matrix", [vocab_size_char,
    embedded = tf.nn.embedding_lookup(embedding_matrix, self.inputs_seq) # E

with tf.variable_scope('encoder'):
    cell_fw = tf.nn.rnn_cell.LSTMCell(hidden_dim)
    cell_bw = tf.nn.rnn_cell.LSTMCell(hidden_dim)
    ((rnn_fw_outputs, rnn_bw_outputs), (rnn_fw_final_state, rnn_bw_final_sta
        cell_fw=cell_fw,
        cell_bw=cell_bw,
        inputs=embedded,
        sequence_length=self.inputs_seq_len,
        dtype=tf.float32
    )
    rnn_outputs = tf.add(rnn_fw_outputs, rnn_bw_outputs) # B * S * D

with tf.variable_scope('projection'):
    logits_seq = tf.layers.dense(rnn_outputs, vocab_size_bio) # B * S * V
    probs_seq = tf.nn.softmax(logits_seq) # B * S * V
    if not use_crf:
        preds_seq = tf.argmax(probs_seq, axis=-1, name="preds_seq") # B * S
```

```
        else:
            log_likelihood, transition_matrix = tf.contrib.crf.crf_log_likelihoc
            preds_seq, crf_scores = tf.contrib.crf.crf_decode(logits_seq, transi

    with tf.variable_scope('loss'):
        if not use_crf:
            loss = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=logits_
            masks = tf.sequence_mask(self.inputs_seq_len, dtype=tf.float32) # B
            loss = tf.reduce_sum(loss * masks, axis=-1) / tf.cast(self.inputs_se
        else:
            loss = -log_likelihood / tf.cast(self.inputs_seq_len, tf.float32) #
```

Tensorflow 里调用 CRF 非常方便，主要就 crf_log_likelihood 和 crf_decode 这两个函数，结果和 loss 就都给你算出来了。它要学习的参数也很简单，就是这个 transition_matrix，形状为 V*V，V 是输出端 BIO 的词表大小。但是有一个小小的缺点，就是官方实现的 crf_log_likelihood 里某个未知的角落有个 stack 操作，会悄悄地吃掉很多的内存。如果 V 较大，内存占用量会极高，训练时间极长。比如我的实验里有 500 个实体类别，也就是 V=500*2+1=1001，训练 1epoch 的时间从 30min 暴增到 400min

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/gradients_impl.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
```

不过好消息是，Tensorflow2.0 里，这个问题不再有了

坏消息是，Tensorflow2.0 直接把 tf.contrib.crf 移除了，目前还没有官方实现的 CRF 接口

再说一下为什么要加 CRF。从开头的 Leaderboard 里可以看到，BiLSTM 的 F1 Score 在72%，而 BiLSTM+CRF 达到 80%，提升明显

Model	Precision / Recall / F1	Training Time - 1 epoch, GPU: Tesla P40
BiLSTM	72.01 / 72.49 / 72.25	30 min
BiLSTM + CRF	82.59 / 78.36 / 80.42	400 min

那么为什么提升这么大呢？CRF 的原理，网上随便搜就一大把，就不讲了（因为的确很难，我也没太懂），但是从实验的角度可以简单说说，就是 LSTM 只能通过输入判断输出，但是 CRF 可以通过学习转移矩阵，看前后的输出来判断当前的输出。这样就能学到一些规律（比如“O 后面不能直接接 I”“B-brand 后面不可能接 I-color”），这些规律在有时会起到至关重要的作用

例如下面的例子，A 是没加 CRF 的输出结果，B 是加了 CRF 的输出结果，一看就懂不细说了

2. BERT+CRF & BERT+LSTM+CRF

用 BERT 来做，结构上跟上面是一样的，只是把 LSTM 换成 BERT 就 ok 了，直接上代码

首先把 BERT 这部分模型搭好，直接用 BERT 的官方代码。这里我把序列长度都标成了“S+2”是为了提醒自己每条数据前后都加了“[CLS]”和“[SEP]”，出结果时需要处理掉

```
from bert import modeling as bert_modeling

self.inputs_seq = tf.placeholder(shape=[None, None], dtype=tf.int32, name="i
```

```
self.inputs_mask = tf.placeholder(shape=[None, None], dtype=tf.int32, name="
self.inputs_segment = tf.placeholder(shape=[None, None], dtype=tf.int32, nam
self.outputs_seq = tf.placeholder(shape=[None, None], dtype=tf.int32, name='

bert_config = bert_modeling.BertConfig.from_json_file("./bert_model/bert_cor

bert_model = bert_modeling.BertModel(
    config=bert_config,
    is_training=True,
    input_ids=self.inputs_seq,
    input_mask=self.inputs_mask,
    token_type_ids=self.inputs_segment,
    use_one_hot_embeddings=False
)

bert_outputs = bert_model.get_sequence_output() # B * (S+2) * D
```

然后在后面接东西就可以了，可以接 LSTM，可以接 CRF

```
if not use_lstm:
    hiddens = bert_outputs
else:
    with tf.variable_scope('bilstm'):
        cell_fw = tf.nn.rnn_cell.LSTMCell(300)
        cell_bw = tf.nn.rnn_cell.LSTMCell(300)
        ((rnn_fw_outputs, rnn_bw_outputs), (rnn_fw_final_state, rnn_bw_final
            cell_fw=cell_fw,
            cell_bw=cell_bw,
            inputs=bert_outputs,
            sequence_length=inputs_seq_len,
            dtype=tf.float32
        )
        rnn_outputs = tf.add(rnn_fw_outputs, rnn_bw_outputs) # B * (S+2) * D
    hiddens = rnn_outputs

with tf.variable_scope('projection'):
    logits_seq = tf.layers.dense(hiddens, vocab_size_bio) # B * (S+2) * V
    probs_seq = tf.nn.softmax(logits_seq)

    if not use_crf:
        preds_seq = tf.argmax(probs_seq, axis=-1, name="preds_seq") # B * (S
    else:
        log_likelihood, transition_matrix = tf.contrib.crf.crf_log_likelihoo
        preds_seq, crf_scores = tf.contrib.crf.crf_decode(logits_seq, transi
```

其实我原来不太相信 BERT 在中文上的效果，加上我比较排斥这种不讲道理的庞然大物

真正实验了发现，BERT确实强啊

把我显存都给吃光了，但确实强啊

训练一轮要那么久，但确实强啊

讲不出任何道理，但确实强啊

相比较单纯使用 BERT，增加了 CRF 后效果有所提高但区别不大，再增加 BiLSTM 后区别很小，甚至降低了那么一内内

Model	Precision / Recall / F1	Training Time - 1 epoch, GPU: Tesla P40
BERT	86.35 / 76.14 / 80.92	140 min
BERT + CRF	86.23 / 77.43 / 81.60	410 min
BERT + BiLSTM + CRF	85.32 / 77.63 / 81.30	460 min

另外，BERT 还有一个至关重要的训练技巧，就是调整学习率。BERT内的参数在 fine-tuning 时，学习率一定要调小，特别时后面还接了别的东西时，一定要按两个学习率走，甚至需要尝试多次反复调，要不然 BERT 很容易就步子迈大了掉沟里爬不上来，个人经验

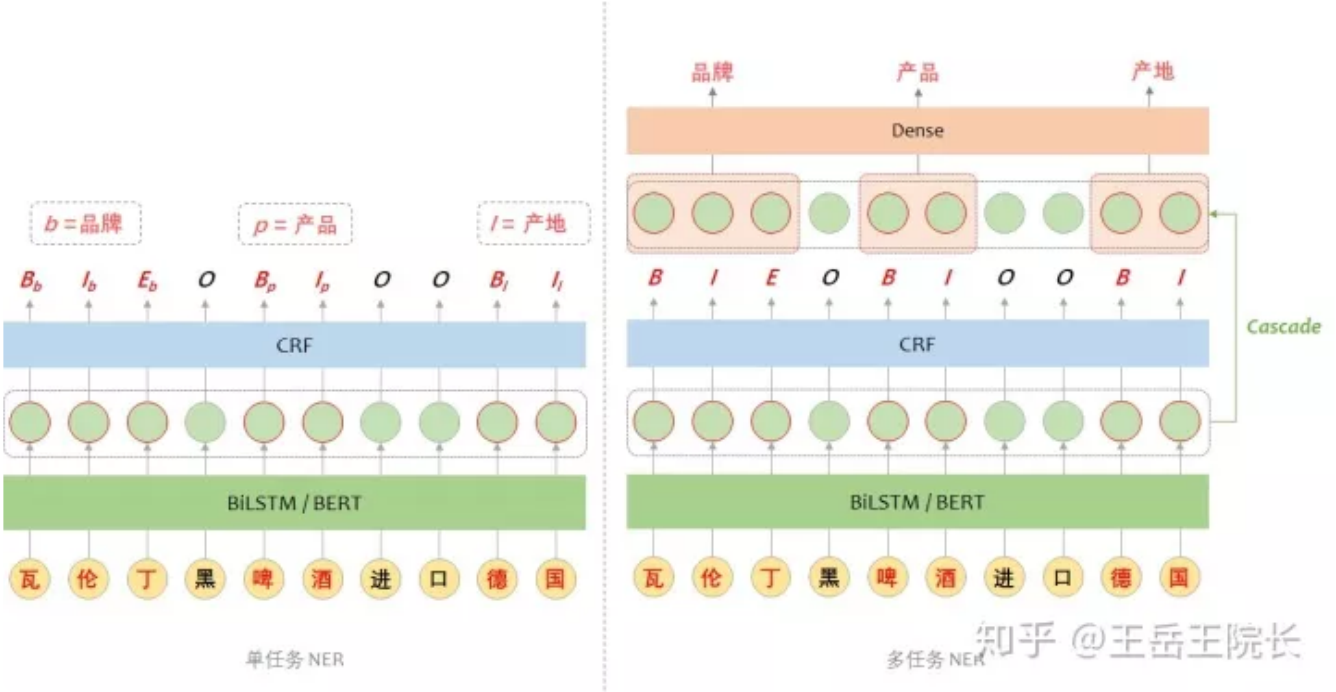
参数优化时分两个学习率，实现起来就是这样

```
    with tf.variable_scope('opt'):
        params_of_bert = []
        params_of_other = []
        for var in tf.trainable_variables():
            vname = var.name
            if vname.startswith("bert"):
                params_of_bert.append(var)
            else:
                params_of_other.append(var)
        opt1 = tf.train.AdamOptimizer(1e-4)
        opt2 = tf.train.AdamOptimizer(1e-3)
        gradients_bert = tf.gradients(loss, params_of_bert)
        gradients_other = tf.gradients(loss, params_of_other)
        gradients_bert_clipped, norm_bert = tf.clip_by_global_norm(gradients_ber
        gradients_other_clipped, norm_other = tf.clip_by_global_norm(gradients_c
        train_op_bert = opt1.apply_gradients(zip(gradients_bert_clipped, params_
        train_op_other = opt2.apply_gradients(zip(gradients_other_clipped, param
```

3. Cascade

上面提到过，如果需要考虑实体类别，那么就需要扩展 BIO 的 tag 列表，给每个“实体类型”都分配一个 B 与 I 的标签，但是当类别数较多时，标签词表规模很大，相当于在每个字上都要做一次类别数巨多的分类任务，不科学，也会影响效果

从这个点出发，就尝试把 NER 改成一个多任务学习的框架，两个任务，一个任务用来单纯抽取实体，一个任务用来判断实体类型，直接上图看区别



这个是参考 ACL2020 的一篇论文[2]的思路改的，“Cascade”这个词是这个论文里提出来的。翻译过来就是“级联”，直观来讲就是“锁定对应关系”。结合模型来说，在第一步得到实体识别的结果之后，返回去到 LSTM 输出那一层，找各个实体词的表征向量，然后再把实体的表征向量输入一层全连接做分类，判断实体类型

关于如何得到实体整体的表征向量，论文里是把各个实体词的向量做平均，但是我搞了好久也没明白这个操作是怎么通过代码实现的，后来看了他的源码，好像只把每个实体最开头和最末尾的两个词做了平均。然后我就更省事，只取了每个实体最末尾的一个词

具体实现上这样写：在训练时，每个词，无论是不是实体词，都过一遍全连接，做实体类型分类计算 loss，然后把非实体词对应的 loss 给 mask 掉；在预测时，就取实体最后一个词对应的分类结果，作为实体类型。上图解释

代码不贴了，感兴趣的可以在 git 里看

说一下效果。将单任务 NER 改成多任务 NER 之后，基于 LSTM 的模型效果降低了 0.4%，基于 BERT 的模型提高了 2.7%，整体还是提高更明显。另外，由于 BIO 词表得到了缩减，CRF 运行时间以及消耗内存迅速减少，训练速度得到提高

Model	Precision / Recall / F1	Training Time - 1 epoch, GPU: Tesla P40
BiLSTM + CRF	82.59 / 78.36 / 80.42	400 min
BERT + CRF	86.23 / 77.43 / 81.60	410 min
Cascade + BiLSTM +CRF	79.77 / 80.21 / 79.99	38 min
Cascade + BERT + CRF	87.83 / 79.28 / 83.33	130 min

P.S. 另外，既然提到了 NER 中的实体类型标签较多的问题，就提一下之前看过的一篇文章[3]。这篇论文主要就是为了解决实体类型标签过多的问题（成千上万的数量级）。文中的方法是：把标签作为输入，也就是把所有可能的实体类型标签都一个个试一遍，根据输入的标签不同，模型会有不同的实体抽取结果。文章没给代码，我复现了一下，效果并不好，具体表现就是无论输入什么标签，模型都倾向于把所有的实体都抽出来，不管这个实体是不是对应这个实体类型标签。也可能是我复现的有问题，不细讲了，就是顺便提一句，看有没有人遇到了和我一样的情况

> Scaling Up Open Tagging from Tens to Thousands: Comprehension Empowered Attribute Value Extraction from Product Title. ACL 2019

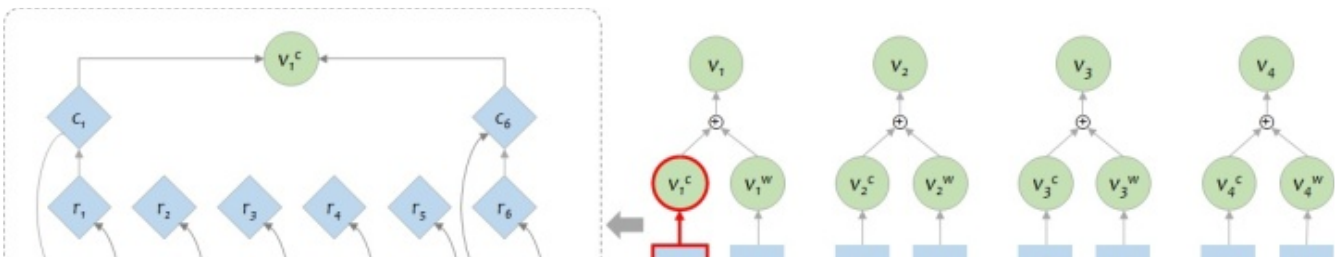
4. Word-Level Feature

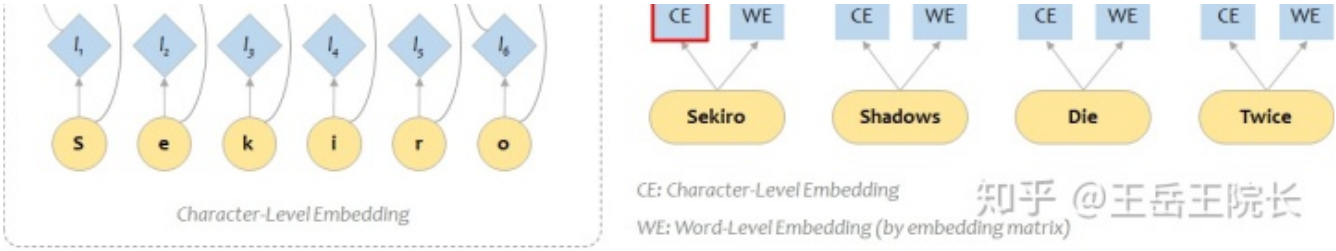
中文 NER 和英文 NER 有个比较明显的区别，就是英文 NER 是从单词级别（word level）来做，而中文 NER 一般是字级别（character level）来做。不仅是 NER，很多 NLP 任务也是这样，BERT 也是这样

因为中文没法天然分词，只能靠分词工具，分出来的不一定对，比如“黑啤酒精酿”，如果被错误分词为“黑啤、酒精、酿”，那么“啤酒”这个实体就抽取不到了。类似情况有很多

但是无论字级别、词级别，都是非常贴近文本原始内容的特征，蕴含了很重要的信息。比如对于英文来说，给个单词“Geilivable”你基本看不懂啥意思，但是看到它以“-able”结尾，就知道可能不是名词；对于中文来说，给个句子“小龙女说我也想过过过儿过过的生活”就一时很难找到实体在哪，但是如果分好词给你，一眼就能找到了。就这个理解力来说，模型跟人是一样的

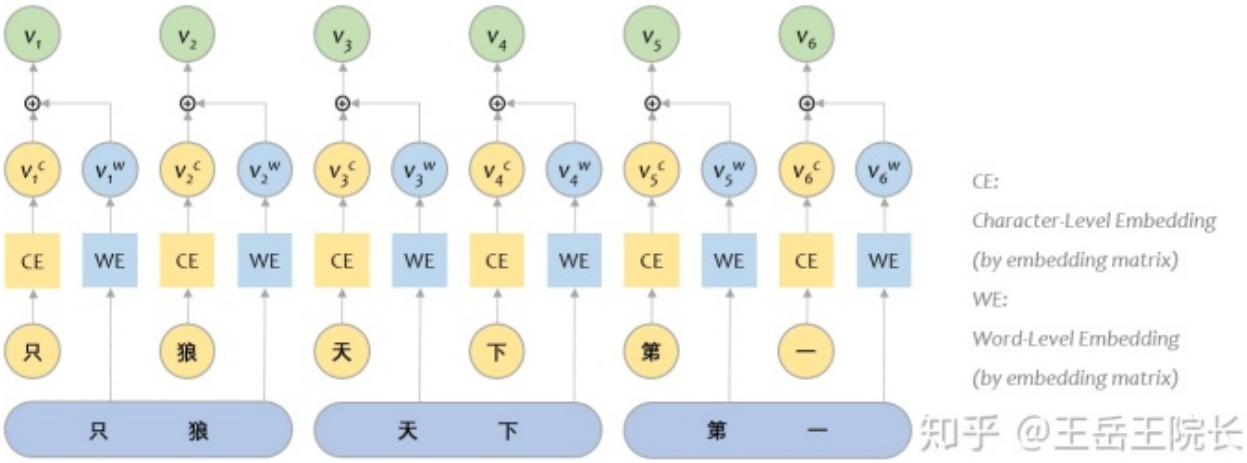
在英文 NLP 任务中，想要把字级别特征加入到词级别特征上去，一般是这样：单独用一个 BiLSTM 作为 character-level 的编码器，把单词的各个字拆开，送进 LSTM 得到向量 vc；然后和原本 word-level 的（经过 embedding matrix 得到的）的向量 vw 加在一起，就能得到融合两种特征的表征向量。如图所示





但是对于中文 NER 任务，我的输入是字级别的，怎么把词级别的表征结果加入进来呢？

ACL2018 有个文章[4]是做这个的，提出了一种 Lattice-LSTM 的结构，但是涉及比较底层的改动，不好实现。后来在 ACL2020 论文里看到一篇文章[5]，简单明了。然后我就再简化一下，直接把字和词分别通过 embedding matrix 做表征，按照对应关系，拼在一起就完事了，看图就懂



具体代码就不放了，感兴趣可以上 git 看

从结果上看，增加了词级别特征后，提升很明显

Model	Precision / Recall / F1	Training Time - 1 epoch, GPU: Tesla P40
Cascade + BiLSTM +CRF	79.77 / 80.21 / 79.99	38 min
Cascade + BiLSTM + CRF + WLF	84.96 / 79.31 / 82.04	40 min

很可惜，我还没有找到把词级别特征结合到 BERT 中的方法。因为 BERT 是字级别预训练好的模型，如果单纯从 embedding 层这么拼接，那后面那些 Transformer 层的参数就都失效了

上面的论文里也提到了和 BERT 结合的问题，论文里还是用 LSTM 来做，只是把句子通过 BERT 得到的编码结果作为一个“额外特征”拼接过来。但是我觉得这不算“结合”，至少不应该。但是也非常容易理解为什么论文里要这么做，BERT 当道的年代，不讲道理，打不过就只能加入，方法不同也得强融，么得办法

5. Weight of Loss

本来打算到这就结束了，后来临时决定再加一点，因为感觉这点应该还挺有意思的

大多数 NLP task 的评价指标有这三个：Precision / Recall / F1Score，Precision 就是找出来的有多少是正确的，Recall 是正确的有多少被找出来了，F1Score是二者的一个均衡分。这里有三点常识

> 方法固定的条件下，一般来说，提高了 Precision 就会降低 Recall，提高了 Recall 就会降低 Precision，结合指标定义很好理解
>
> 通常来说，F1Score 是最重要的指标，为了让 F1Score 最大化，通常需要调整权衡 Precision 与 Recall 的大小，让两者达到近似，此时 F1Score 是最大的
>
> 但是 F1Score 大，不代表模型就好。因为结合工程实际来说，不同场景不同需求下，对 P/R 会有不同的要求。有些场景就是要求准，不允许出错，所以对 Precision 要求比较高，而有些则相反，不希望有漏网之鱼，所以对 Recall 要求高

对于一个分类任务，是很容易通过设置一个可调的“阈值”来达到控制 P/R 的目的的。举个例子，判断一张图是不是 H 图，做一个二分类模型，假设模型认为图片是 H 图的概率是 p，人为设定一个阈值 a，假如 p>a 则认为该图片是 H 图。默认情况 p=0.5，此时如果降低 p，就能达到提高 Recall 降低 Precision 的目的

但是 NER 任务怎么整呢，他的结果是一个完整的序列，你又不能给每个位置都卡一个阈值，没有意义

然后我想了一个办法，通过控制模型学习时的 Loss 来控制 P/R：如果模型没有识别到一个本应该识别到的实体，就增大对应的 Loss，加重对模型的惩罚；如果模型识别到了一个不应

该识别到的实体，就减小对应的 Loss，当然是选择原谅他

实现上也是通过 mask 来实现，看图就懂

实现也非常简单，放一下对应的代码

```
# logits_bio 是预测结果，形状为 B*S*V，softmax 之后就是每个字在BIO词表上的分布概率，不
# self.outputs_seq_bio 是期望输出，形状为 B*S
# 这是原本计算出来的 loss
loss_bio = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=logits_bio,
# 这是根据期望的输出，获得 mask 向量，向量里出现1的位置代表对应的字是一个实体词，而 O_tag_
masks_of_entity = tf.cast(tf.not_equal(self.outputs_seq_bio, O_tag_index), t
# 这是基于 mask 计算 weights
weights_of_loss = masks_of_entity + 0.5 # B  *S
# 这是加权后的 loss
loss_bio = loss_bio * weights_of_loss # B * S
```

从实验效果来看，原本 Precision 远大于 Recall，通过权衡，把两个分数拉到同个水平，可以提升最终的 F1Score

Model	Precision / Recall / F1	Training Time - 1 epoch, GPU: Tesla P40

Cascade + BiLSTM + CRF + WLF	84.96 / 79.31 / 82.04	40 min
Cascade + BiLSTM + CRF + WLF + WOL	82.60 / 82.14 / 82.37	40 min
Cascade + BERT + CRF	87.83 / 79.28 / 83.33	130 min
Cascade + BERT + CRF + WOL	83.51 / 84.22 / 83.86	130 min

除此之外，在所有深度学习任务上，都可以通过调整 Loss 来达到各种特殊的效果，还是挺有意思的，放飞想象，突破自我

总结

总结放在开头了，就这样

完结，撒花

「参考」

Bidirectional LSTM-CRF Models for Sequence Tagging

A Novel Cascade Binary Tagging Framework for Relational Triple Extraction

Scaling Up Open Tagging from Tens to Thousands: Comprehension Empowered Attribute Value Extraction from Product Title

Chinese NER Using Lattice LSTM

Simplify the Usage of Lexicon in Chinese NER

The End

【推荐阅读】

初学者|NLP相关任务简介

【科研】自然语言理解难在哪儿?

自然语言处理中注意力机制综述

新年送福气|您有一份NLP大礼包待领取

“达观杯”文本智能处理挑战赛，季军带你飞

【机器学习】一文读懂线性回归、岭回归和Lasso回归