1. 背景介绍

本文为命名实体识别项目《NER Toy》的实验报告。

命名实体识别（英语：Named Entity Recognition），简称 NER，是指识别文本中具有特定意义的实体，主要包括人名、地名、机构名、专有名词等，以及时间、数量、货币、比例数值等文字。

例如，在句子“玄幻的也很好看，比如天蚕土豆的斗破苍穹，武动乾坤”中，我们可以提取出 Book 实体：“斗破苍穹”和“武动乾坤”。

本项目的目的是构建一个能够自动地从文本中提取命名实体的模型。其中，输入为句子，输出为句子中可能存在的实体，并进行标注。标注的方式采用“BIO”的形式：B 表示实体的开始，I 表示实体的内部或者实体的末尾，O 表示非实体。

2. 数据集介绍

数据集为 NER_toy.zip，包含 train.txt，valid.txt 和 test.txt。train.txt 包含 7809 个句子，valid.txt 包含 975 个句子，test.txt 包含 978 个句子。

每个文件中包含了两列，第一列是句子中的字，第二列是该字对应的标注。例如：

```
玄 O
幻 O
的 O
也 O
很 O
好 O
看 O
， O
比 O
如 O
天 O
蚕 O
土 O
豆 O
的 O
斗 B-book
破 I-book
苍 I-book
穹 I-book
， O
武 B-book
动 I-book
乾 I-book
坤 I-book
。 O
```

不同的句子之间用换行符加以区分。

3. 数据预处理

在数据预处理部分，为每个字提取了词性、词边界、偏旁部首和拼音等特征。

然后为字、词性、词边界、偏旁部首、拼音以及标签（也就是 BIO 标注）分别建立词汇表，将具体的值映射到 index。这样一来，每个字就具有多个特征值。那么，一个句子中有多个字，我们就能得到句子在某个特征上的 sequence。

另外，由于训练集中句子只有 7000 多个，我在这里做了数据增强，对句子长度进行排序，然后将相邻的两个句子进行拼接、将相邻的三个句子进行拼接。

在返回 batch 数据之前，根据整个 batch 中最长句子的长度 seq_len，对整个 batch 中的数据进行 padding。返回的一个 bacth 的 shape 为：$[5, batch_size, seq_len]$，这里的 5 表示的是总共有 5 个特征：

```
[
    [ [填充后的句子1的word向量], [填充后的句子2的word特征], ...],
    [ [填充后的句子1的flag向量], [填充后的句子2的flag向量], ...],
    ...,
    [ [填充后的句子1的pinyin向量], [填充后的句子2的pinyin向量], ...]
]
```

4. 模型

4.1 Embedding

在得到了原始的特征向量以后，需要做 Embedding。

所谓的 Embedding 可以理解成参数矩阵，如下图，假设上面的矩阵是目标空间中的词向量，下面的矩阵是两个句子的 word 或者其他特征的 index 序列。那么，对于第一个句子的第一个元素 0，需要到目标空间中的词向量中进行查找，得到下标为 0 的元素为：[1, 1, 1, 1, 1]；同理，对于第一个句子的第二个元素 2，到目标空间中的词向量中进行查找，得到下标为 2 的元素为：[3, 3, 3, 3, 3] 。

以此类推，最终就可以得到 Embedding 后的矩阵。

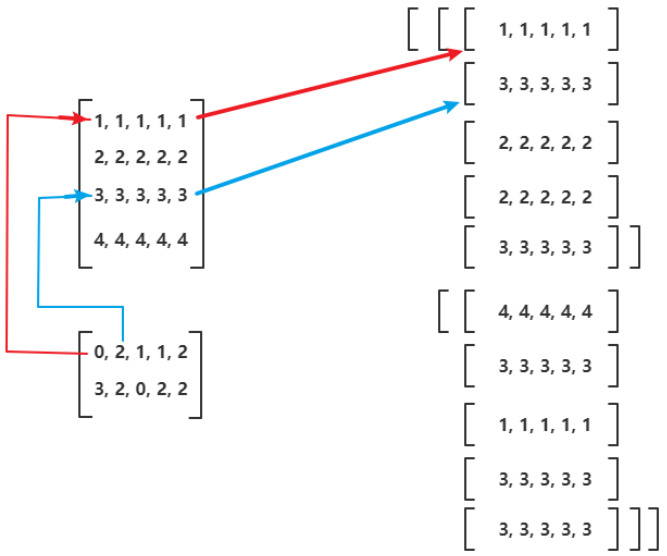

图 1: embedding-demo

通过模型训练更新矩阵，得到不同特征不同值的嵌入向量。这样的嵌入向量相比于前面预处理中简单的将特征值转换为 index 具有更丰富的信息。

对于 pytorch 来说，Embedding 的 API 为：nn.Embedding(vocab_size, embed_dim)。其中，vocab_size 为词表大小，以 word 为例，vocab_size 就是 word 中所有词的数目。embed_dim 是你指定的要将该特征嵌入到多少维的空间中。

因为在数据预处理部分，除了 word 还额外创建了四个特征，因此需要为不同的特征创建不同的 nn.Embeddig 对象。

```
1
2 word: nn.Embedding(word_vocab_size, 100)
3 flag(词性): nn.Embedding(word_vocab_size, 50)
4 bound(词位 or 词边界): nn.Embedding(word_vocab_size, 50)
5 radical(偏旁部首): nn.Embedding(word_vocab_size, 50)
6 pinyin(拼音): nn.Embedding(word_vocab_size, 80)
```

如下图所示，对于 word 来说，假设 batch_size 为 10，当前 batch 中句子最大长度也为 10。那么经过 Embedding 之后，变为 [10, 10, 100] ，也就是 [batch_size, max_len, embed_dim]

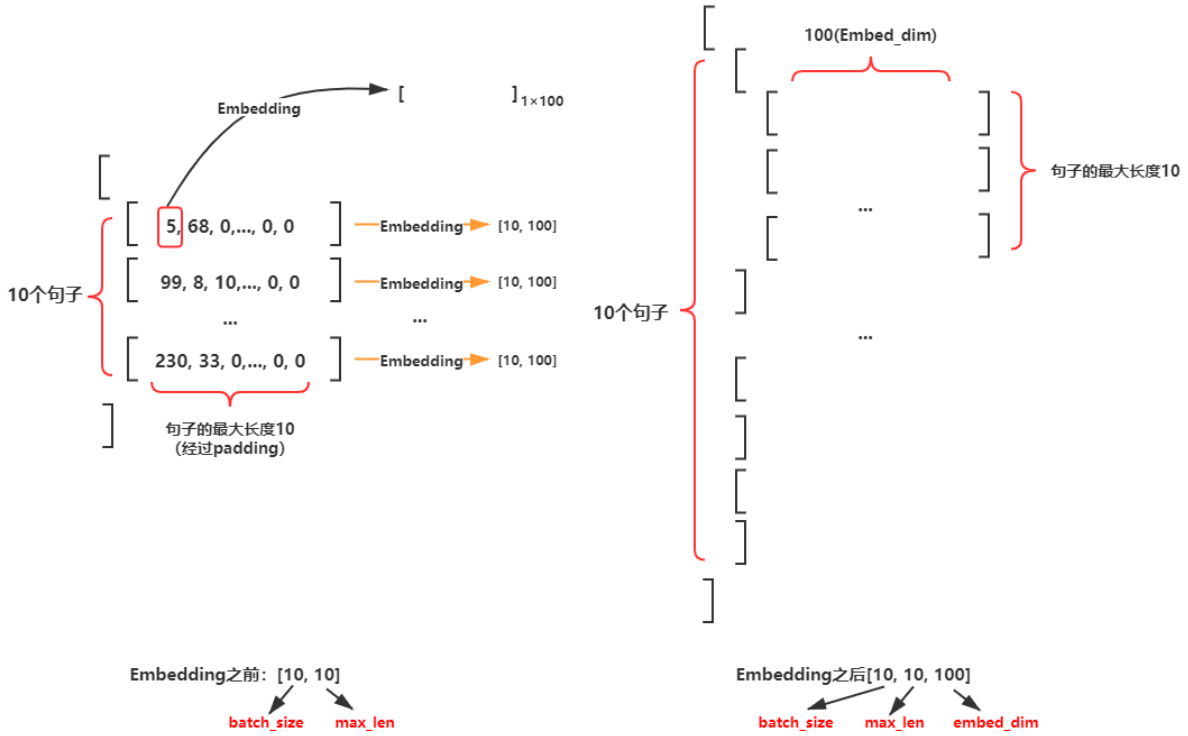

图 2: 具体的 embedding-demo

既然一个 word 有这么多个特征，并且 Embedding 之后又各过各的，这样是没法训练的。因此，训练之前我们肯定要通过某种方式将属于同一个 word 的 Embedding 之后的向量揉在一起。

最简单的，便是利用 pytorch 跟 concat 相关的 API，将属于一个 word 的向量都拼接在一起。最后，拼接完的维度就是：[10, 10, 100+50+50+50+80] = [10, 10, 330]，即 [batch_size, max_len, word_embed_dim + flag_embed_dim + bound_embed_dim + radical_embed_dim + pinyin_embed_dim]

4.2 模型结构

在做完 Embedding 之后，就可以将数据喂给模型。

模型的整体架构如下，将输入数据喂给 BiLSTM，输出每个 word 在不同 label 上的概率，将这样的概率矩阵（发射得分矩阵）传递给 CRF。在 CRF 中，初始化一个权重矩阵，作为转移得分矩阵。通过 CRF，可以学习到标签在转移时候的约束条件。比如 B-xx 后面跟的是 I-xx。

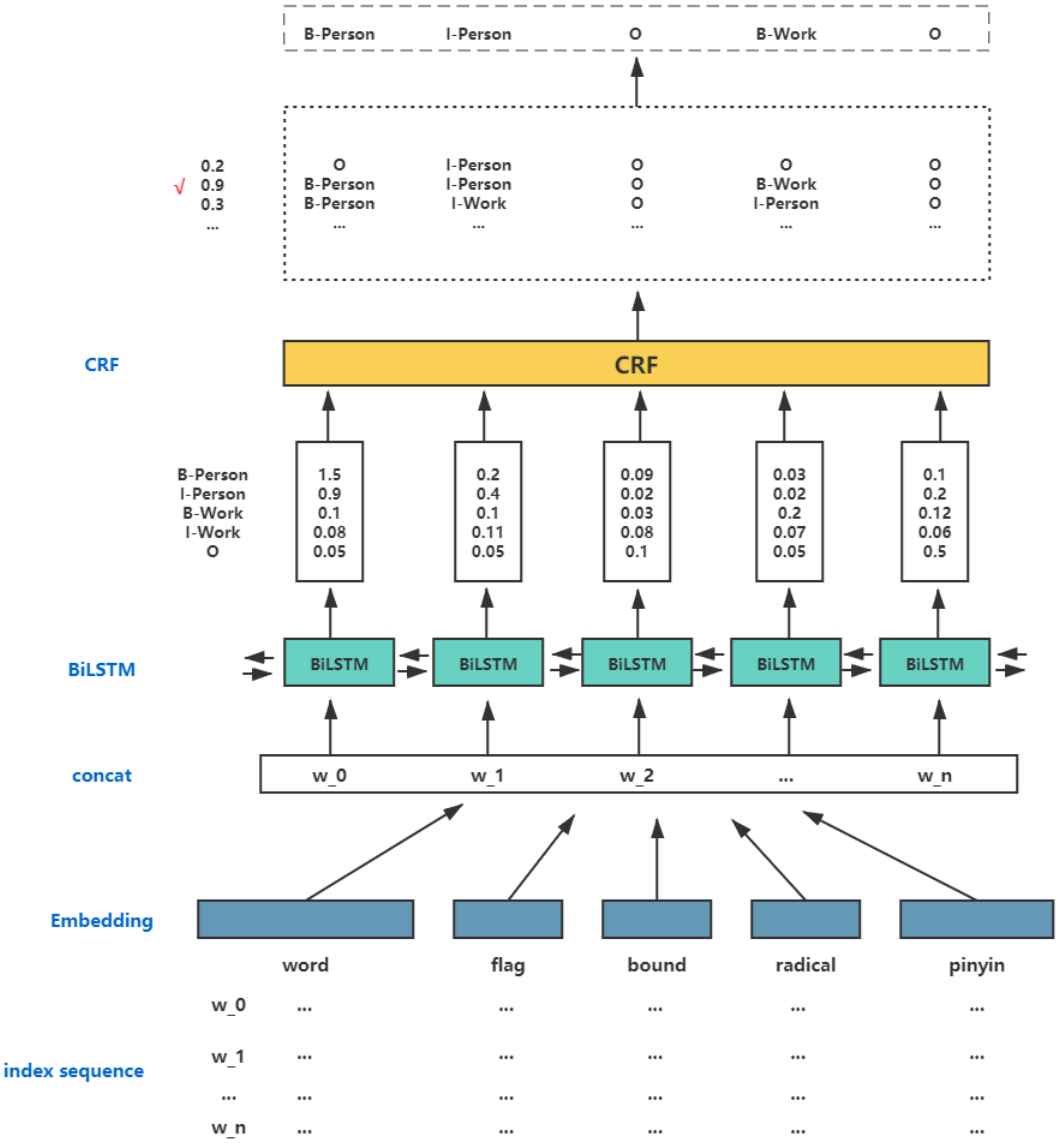

图 3: 模型架构

4.3 Loss

给定一个句子，句子中字的 label 序列存在多条路径。那么，loss 的计算与真实路径得分和所有路径得分相关：

$$LossFunction = \frac{P_{RealPath}}{P_1 + P_2 + ... + P_N} \tag{1}$$

训练的目标是使得真实路径的得分站的比重越来越大。

4.4 模型评估

模型的评估指标采用精确率（Precision）、召回率（Recall）和 F1-Score。

举例来说，假设给定一个句子的 label sequence：y_true，以及模型预测的 y_pred。那么需要计算以下三种指标：

$$\begin{aligned} TP &= y\_pred\text{中标注的实体在}y\_true\text{中也出现了} \\ FP &= y\_pred\text{中标注的实体在}y\_true\text{中并未出现} \\ FN &= y\_true\text{中的实体在}y\_pred\text{中并未出现} \end{aligned} \tag{2}$$

则：

$$\begin{aligned} precision &= \frac{TP}{TP+FP} \\ recall &= \frac{TP}{TP+FN} \\ f1 &= \frac{2 \cdot precision \cdot recall}{precision + recall} \end{aligned} \tag{3}$$

5. 实验

5.1 实验设置

1. 数据集：NER_toy.zip
2. 超参数：
   - epoch: 20
   - batch_size: 8
   - hidden_dim: 128
   - word_embed_dim: 100
   - flag_embed_dim: 50
   - bound_embed_dim: 50
   - radical_embed_dim: 50
   - pinyin_embed_dim: 80
3. gpu or cpu: 由于从 cpu 改到 gpu 的 batch 运算还有点问题... 所以最后只在 cpu 上跑了
4. 训练与评估：每两个 epoch 就在验证集上验证，将验证集上 f1-score 最好的模型保存到本地。最后再加载模型，在测试集上进行验证。

5.2 实验结果

5.2.1 数据增强

经过一段老牛破车般漫长的等待以后，得到了下面的实验结果。

经过 20 个 epoch 的训练，模型在训练集上的 loss 逐步下降，f1-score 逐步上升；在验证集上的 loss 先下降后上升，f1-score 先上升后下降，但是变化幅度不大，有种出道即巅峰的感觉。

做完数据增强，扩充了训练集以后，训练集中的句子数量达到了两万多，而训练集和测试集中的句子较少。我认为可能是模型过拟合了，有可能是数据增强那部分对句子进行拼接的操作导致的。

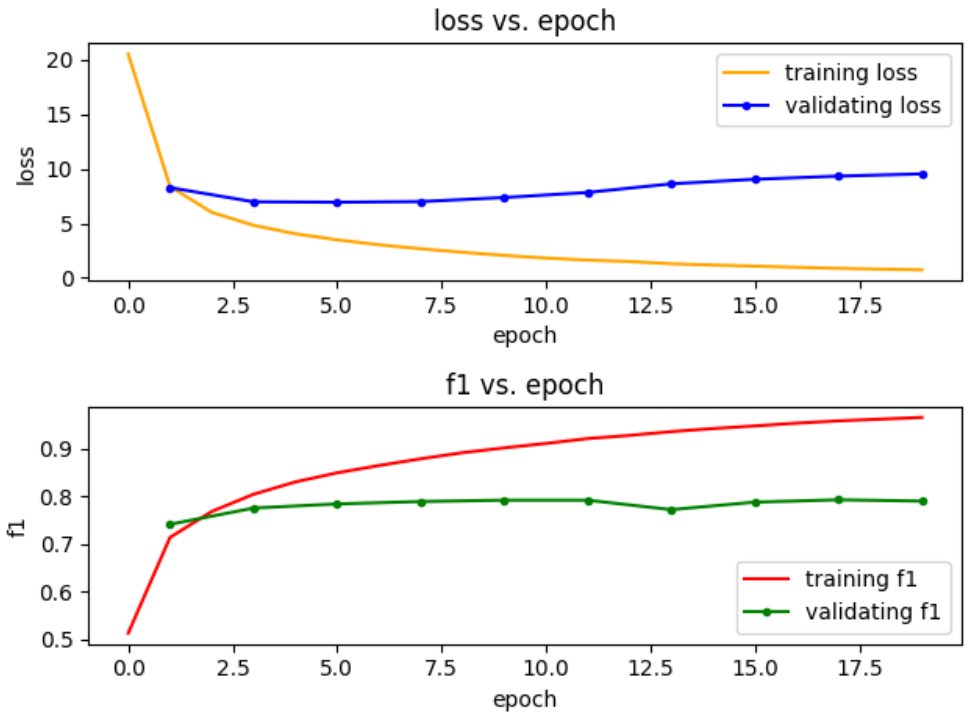

图 4: train-process

最终在训练集上的 f1-score 为 0.9653，在验证集上的 f1-score 为 0.7928，在测试集上的 f1-score 为 0.7841

dataset	precision	recall	f1-score
训练集	0.9731	0.9576	0.9653
验证集	0.8033	0.7826	0.7928
测试集	0.8043	0.7650	0.7841

表 1: 实验结果

5.2.2 未做数据增强

将数据增强的部分去掉以后，训练集变为了 7809 个句子。训练速度比之前的快很多，但是模型收敛得比较慢。从图中可以看出，验证集上的 loss 是先下降后上升的。

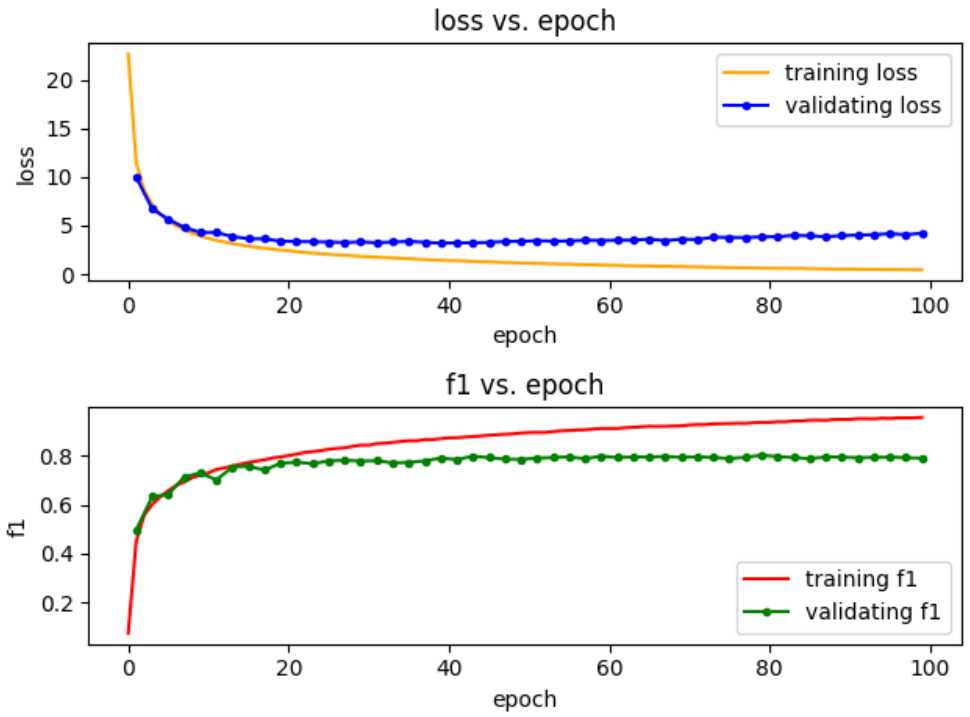

图 5: train-process2

在训练了 20 个 epoch 以后，模型在验证集上：precision_score=0.8282, recall_score=0.7158, f1_score=0.7679。

训练了 100 个 epoch 以后，结果如下表。在训练集上的 f1-score 为 0.9549，在验证集上的 f1-score 为 0.8006，在测试集上的 f1-score 为 **0.7879**。

dataset	precision	recall	f1-score
训练集	0.9641	0.9459	0.9549
验证集	0.8183	0.7836	0.8006
测试集	0.8114	0.7657	**0.7879**

表 2: 实验结果

5.3 模型预测

从测试集中随机选取一个 bacth，用在未做数据增强的数据集上训练得到的模型进行预测。

下面是从中选取的一个句子的预测效果：

其中，第一列为 word，第二列为模型预测的该 word 的 label，第三列为真实的 label，第四列为判断模型预测和真实值是否相等。

从一些测试的结果来看，预测的结果还是挺准确的。

```
word            pred            true            correct?
-------------------------------------------------------
```

```
加          O           O           True
班          O           O           True
,           O           O           True
就          O           O           True
应          O           O           True
该          O           O           True
听          O           O           True
一          O           O           True
些          O           O           True
比          O           O           True
较          O           O           True
刺          O           O           True
激          O           O           True
的          O           O           True
歌          O           O           True
,           O           O           True
,           O           O           True
狂          B-music     B-music     True
浪          I-music     I-music     True
,           O           O           True
,           O           O           True
沙          B-music     B-music     True
漠          I-music     I-music     True
UNK         I-music     I-music     True
UNK         I-music     I-music     True
,           O           O           True
,           O           O           True
num         B-music     B-music     True
num         I-music     I-music     True
度          I-music     I-music     True
num         I-music     I-music     True
,           O           O           True
我          O           O           True
比          O           O           True
较          O           O           True
喜          O           O           True
欢          O           O           True
听          O           O           True
这          O           O           True
种          O           O           True
类          O           O           True
型          O           O           True
的          O           O           True
歌          O           O           True
,           O           O           True
你          O           O           True
听          O           O           True
```

```
51 听          O          O          True
52 看          O          O          True
```

6. 总结

本文用 BiLSTM+CRF 实现了命名实体识别任务，最终，在测试集上的 f1-score 可以达到 0.7879。

经过本次实验，学到了数据预处理、模型搭建、评估指标，还有训练和评估的知识。现在的代码还有一些地方可以改进和实验的，比如：

- 做数据增强，应该也可以对验证集和测试集做相同的处理。或者做完数据增强以后，再将三个数据集融合，然后重新划分数据集。
- 词向量不是随机初始化，而是使用预训练的权重向量
- 模型中将一些循环改成适用于 GPU 的矩阵运算。（这边一直不知道咋改... 目前只是把最里层的计算改成了矩阵运算，但是结果好像和非矩阵运算版本的有点出入）