命名实体识别报告

姓名：张荐科

目录

1 命名实体识别-NER 概述

命名实体识别——Name Entity Recognition (NER) 是在一段文本中，将预先定义好的实体类型识别出来；它是自然语言处理中一个非常重要且基础的问题。

解决 NER 问题的第一步是文字向数据的转化，实际上就是特征数据的提取。这一部分在计算机视觉中其实并没有这么复杂，因为图片本身就是一个大量矩阵数据组成的事物。而相比之下，文字以及它所蕴含的语义，它与周围文字间的联系都是非数值形式的，如何将它量化也是 NER 面临的第一个问题。目前，文本特征表示的方法一般都是在单个字符或词库中的词语的独热向量量化（One-hot Vector）的基础上，分析文本上下文或部分词语特征进行单个数据的串联合并，从而生成一种能够体现局部语义信息的特征数据。这其中包括 Word-Embedding、Char-Embedding 及其之上的 Word2Vec、GloVe 等等 NLP 特征提取和处理算法。

有了语言的特征提取后，NER 所做的识别实际上就是经典机器学习领域的分类问题。可以使用传统机器学习的有无监督、聚类等等和深度学习里的更深更庞大的学习网络的方法。这时我们能够用随心所欲的采用各种机器学习的算法进行处理。

最后就是模型的验证与评估。这一点在 NLP 领域和 CV 领域有着很大的不同。一般来说分类的结果只需要去考察标签预测的准确率，然而这在 NER 中是远远不够的。NER 中的实体是通过一个或者多个标签决定的，因此为这个特点设置了专门的针对实体判断准确性的指标，分别包括 precision 查准率，recall 查全率和 F1-measure 指标，分别体现了查找到的实体中正确的概率，能够查找到多少实体和综合性能，这三个指标以及常规的准确率 accuracy 间有着很大的差异，学习的结果和程度也大相径庭，这一点将在后面的参数测试和效果评估中详细阐述。

另外，作为理论依据，我自己推导了 softmax 的求导，这里简写如下：

$$L=\frac{1}{N}\sum_{i=1}^{N}L_i\ (+reg\cdot\sum_k W_k^2)=\frac{1}{N}\sum_{i=1}^{N}-log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

$softmax$的$\nabla_W L$计算方法

$\frac{\partial L_k}{\partial w_{ij}}=-\frac{\sum_j e^{s_j}}{e^{s_{y_k}}}\cdot\frac{\partial(\frac{e^{s_{y_k}}}{\sum_j e^{s_j}})}{\partial w_{ij}}$，　w_{ij}即第 i 个像素的第 j 个分类的权重值，k 表示第 k 张图片的$loss$

若$j=y_k$(即 j 是 i 张图片的正确分类)

$$\frac{\partial L_k}{\partial w_{ij}}=-\frac{\sum_j e^{s_j}}{e^{s_{y_k}}}\cdot\frac{\partial(1-\frac{(\sum_j e^{s_j})-e^{s_{y_k}}}{\sum_j e^{s_j}})}{\partial w_{ij}}$$

考虑$s=xW$可以知道分子$(\sum_j e^{s_j})-e^{s_{y_k}}$不含$w_{ij}$,故只需对分母求导

$$\frac{\partial L_k}{\partial w_{ij}}=-\frac{\sum_j e^{s_j}}{e^{s_{y_k}}}\cdot\frac{(\sum_j e^{s_j})-e^{s_{y_k}}}{(\sum_j e^{s_j})^2}\cdot\frac{\partial e_{s_{y_k}}}{\partial w_{ij}}=-\frac{(\sum_j e^{s_j})-e^{s_{y_k}}}{e^{s_{y_k}}\cdot\sum_j e^{s_j}}\cdot e_{s_{y_k}}\cdot x_{ki}$$

$$=(\frac{e_{s_{y_k}}}{\sum_j e^{s_j}}-1)\cdot x_{ki}=(\frac{e_{s_j}}{\sum_j e^{s_j}}-1)\cdot x_{ki}$$

若$j\neq y_k$(即 j 是 i 张图片的正确分类)

$$\frac{\partial L_k}{\partial w_{ij}}=-\frac{\sum_j e^{s_j}}{e^{s_{y_k}}}\cdot\frac{\partial(\frac{e^{s_{y_k}}}{\sum_j e^{s_j}})}{\partial w_{ij}}$$

考虑$s=xW$可以知道分子$e^{s_{y_k}}$不含w_{ij},故只需对分母求导

$$\frac{\partial L_k}{\partial w_{ij}}=-\frac{\sum_j e^{s_j}}{e^{s_{y_k}}}\cdot\frac{e^{s_{y_k}}}{(\sum_j e^{s_j})^2}\cdot\frac{\partial e_{s_{y_k}}}{\partial w_{ij}}=\frac{1}{\sum_j e^{s_j}}\cdot e_{s_j}\cdot x_{ki}=(\frac{e_{s_j}}{\sum_j e^{s_j}})\cdot x_{ki}$$

以上推到可以直接转换为程序，遍历每一张图片和每一个分类，找出 s 行向量，

然后每次可以直接更新 dW 的一列（因为一行 s 的生成只与 w 的一列相关）

根据这一操作思路可以类似转化为向量化操作

2 我的工作

为了统一数据处理得到更好的训练效果，我使用了 **7** 分类进行命名实体识别，七种分类分别是 B-PERS，I- PERS，B-LOC，I- LOC，B-ORG，I- ORG，O。

总体来说，命名实体识别的项目中有关机器学习本身的部分相对简单，就是经典的 soft-

max 进行多分类。然而，作为 NLP 问题，比较耗时耗力的工作主要是在数据预处理和结果调试，由于文本与纯数值数据不同，特征的编码方式会直接影响训练的结果，因此这两个工作贯穿始终，我花费了大量时间在反复调节这两个环节最终得出一个令人满意的结果。

下面以尽量简短的方式列出我在这个项目中所进行的必要的工作：

1. 将最原始的 nr、ns、nt 标签数据进行清洗去噪，更换为新的 BIO 七分类标签导入另一个 txt 文件
2. 清洗的同时按照人名、地点、机构和其他 4 种大类统计各类的高频词，按序存入 4 个字典中，方便后面词库的调整和编码生成
3. 将高频词按以一定方式混合成为词库，根据词库为所有词进行 one-hot 编码，存入另一个字典备用
4. 导入第 1 步种产生的 BIO 标签文件，根据每个词的标签，链接其前后两个词的 one-hot 向量生成数据集备用，同时记录验证集和测试集饿划分界限
5. 定义 softmax 类，将训练所用的权重矩阵，学习率，mini-batch 操作，迭代梯度下降（编写了 pytorch 自动和手动求导两种）封装，调试好对外饿训练接口，方便使用
6. 编写实体识别函数，从 7 分类标签向量中检测实体，并据此编写求去查全率、查准率、F1-measure 的函数
7. 主函数中生成 train、validation、test 数据集，尝试不同的 learn-rate 进行训练，选取验证集上效果最好的模型在测试集上测试，完成相应结果的可视化，调参，debug 等工作
8. 在 jupyterNotebook 中完成 main 函数结果的展示，并额外进行了各种数据的可视化

另一方面，我还在调试词库，训练参数等方面做了很多工作，得出了很多有效的结论，是的训练结果的各项指标均很理想，F1-measure 达到 ***69.6%***。下面列出了我在调试中所做的工作，并给出了经过大量比较后得出的具有比较好结果的结论：

1. 尝试不同的学习率，选择效果最好的参数作为输出模型，最终的 learn-rate 取为 ***2***
2. 更改不同的词库选择方式，最终采用三种大类各取 600 个，其他类中取 100 个组成大小为 1900 的词库，兼顾 other 和实体词汇
3. 更改训练集中巨量样本 O 标签的个数，防止准确度虚高而实体检测能力很低的现象，训练集中只保留总训练集个数约 42% 的标签为 O 的样本（原有数据集中 O 占据了 93% 的样本，导致学习实体效果极差，筛选后比较好的方式是从中只取约 60000 个 O 样本）
4. 考察了迭代过程中 loss 的变化趋势，进行可视化并进行分析讨论
5. 考察了迭代中 precision、recall、F1-measure 的变化，进行可视化并进行分析讨论

所有代码文件的具体结果在 ***NER.ipynb*** 文件中给出，可以通过 jupyternotebook 打开查看详细过程和代码结果。

3 流程细节

3.1 数据预处理

3.1.1 数据清洗和重贴标签

数据预处理的第一步是进行数据清洗和重贴标签，仔细观察数据集，可以发现主要清洗的几种情况如下：1. 日期等标记为 m 的无关标签；2. 两种不同类型实体相互嵌套导致同一词语有两个标签；3. 大量标点符号和括号等语义无关的符号；4. 拼音标注和未标注词语的统一合并；5. 与语义无关的中文数字。据此，数据清洗中我主要进行了一下步骤：

- 按照 “\" 字符切分数据，对照规范标准处理 nr、nt、ns 对应的 BI 标签，其他类型的标记为 O
- 针对特殊的字符和嵌套情况进行特判，提高数据的可靠性，减少噪声
- 将词语和其标签以键值对存入字典中备用
- 方便直观判断处理效果，使用 write 函数将清洗重贴标签后的数据重写入一个新文件
- 处理同时统计各个类别词汇的出现次数，按照 PERS、LOC、ORG、O 四种分别存入四个字典中，用于后续的词库生成和调整

  这里展示重贴标签的一小部分示例：

$$19980101-01-001-001:m \quad 迈向:O \quad 充满:O \quad 希望:O \quad 的:O \quad 新:O \quad 世纪:O \quad ——:O$$
$$一九九八年:O \quad 新年:O \quad 讲话:O \quad （:O \quad 附:O \quad 图片:O \quad 1:O \quad 张:O \quad ）:O$$
$$19980101-01-001-002:m \quad 中共中央:B-ORG \quad 总书记:O \quad 、:O \quad 国家:O$$
$$主席:O \quad 江:B-PERS \quad 泽民:I-PERS$$

（这一部分的代码在 ***Preprocess.py*** 中的 ***BIO_label_dataset*** 和 ***write_marked_list_to_txt*** 中）

3.1.2 调整总词库并生成 one-hot 编码

由于词库决定了一个词语 one-hot 编码的长度，进而决定了特征的长度，极大影响了输入数据的大小，因此词库的大小和使用的词语都需要进行仔细的思考和测验。具体的测试调整需要在模型搭建完才能进行，因此在这一部分中我提前留好了接口，可以轻松地调整词库大小和各个类型词语的比例。

确定好了词库，就需要为每一个词语准备对应的 one-hot 向量，这个向量的长度是词库长度加 1，用单个位置的 1 表示这个词，例如：

大小为 5 的词库中第 2 个词$one-hot$向量编码为$(0,1,0,0,0,0)$,不在词库中的词编码为$(0,0,0,0,0,1)$

由于后面的处理全部使用的是 numpy 的矩阵类型，因此所有数据全部用 numpy 的 array 类型存储。这一部分的代码相对简单，需要注意的是词库的大小的选取，太大会导致内存不够无法正常开取空间，太小会导致训练准确性极低。

（这一部分的处理代码在 ***Preprocess.py*** 中的 ***Generate_onehot_code*** 和主函数中）

3.1.3 softmax 模型搭建

之后便是传统机器学习的经典模型搭建了。这里我主要学习了 cs231n 等比较经典的框架写法，编写了一个 softmax 类，把 **sgd** 优化器，训练迭代，历史指标记录，各个参数调整封装在其中。这样，定义并训练好的一个类就可以直接作为一个整体的模型用于其他数据的预测中，极大的方便了调试和测试使用。

优化器部分我分别编写了手动求导和自动求导两个函数，他们的接口完全一样，但是由于 pytorch 自动求导在这部分的优化与手动编写的向量化计算几乎一样，因此我最终在训练的时候使用了自己写的手动求导（与自动求导结果完全相同）。

另外，为了加速训练过程，这里使用了 ***mini-batch*** 小批量梯度下降的操作，它的原理就是在计算梯度的时候只用数据集中随机选取的一小部分，用它们的梯度作为整体的梯度，从而极大地加速了训练速度，在稳定性上和使用整个数据集求取梯度相同，但是节省了大量计算和时间资源。

这里呈现我编写的模型类的框架：（具体代码在 ***Softmax.py*** 中）Class Softmax(object)

- def init(self): """ 初始化类参数"""
- def train( ): """ Softmax 线性分类模型-完成迭代和梯度下降参数优化"""
- def predict(self, X): """ 将训练好的模型用于预测"""
- def loss(self, X, y): """ 手动求导的实现 (最终迭代使用) """
- def loss-with-autograd(self, X, y): """ 自动求导的实现 (最终迭代并未使用, 但与手动求导的结果完全相同, 写在这里保证任务的完整) """

3.1.4 模型训练与测试接口

这一部分用于将模型用于训练和测试，并进行一定程度的数据可视化。

模型中最重要的参数是梯度下降使用的学习率（learn-rate），因此在主函数的训练中需要使用循环针对不同的学习率进行训练，并记录各个评价指标，用 best-model 记录在验证集表现最好的模型，准备用于最后的测试集进行测试。在这一部分中，可以充分使用模型编写好的参数接口，可以多次测试不同迭代次数，mini-batch 大小等等可调超参数，并在循环中设置一定的输出，即时展示训练的过程和中间效果。

最后，在训练结束将最好的模型应用于测试集进行测试，查看最终的模型效果。

（这一部分的处理代码在 ***main.py*** 中）

4 最优训练结果呈现（7分类结果）

这里放出调试后我所得到最优参数条件下的训练结果（运行结果截图见附件）。

最终训练参数表：

表 1: 最优结果参数表

参数	值
学习率（learn-rate）	2
词库大小	每类 600 个（人名、机构、地点）+100 个 other，共 1900
mini-batch 大小	200
迭代次数	30000
O 类样本比例	60000、42%

训练结果和迭代过程中 loss 下降效果及训练集各个指标的变化曲线、在测试集上的预测结果如下:

```
Training results of learning-rate  2
################ Model on train set ###############
Total accuracy             :  0.6619351931506288
Train precision ratio      :  0.819403499368276
Train recall ratio         :  0.6747695771476817
Train F1-measure           :  0.7400863547498906
############ Model on validation set ###############
Total accuracy             :  0.8045619819066624
Validation precision ratio :  0.7304867634500427
Validation recall ratio    :  0.7403370503898717
Validation F1-measure      :  0.735378922631872
```

图 1: 训练结果

```
-------------Best model on test set!----------------
Label accuracy             :  0.8066437414030261
Test precision ratio       :  0.6933530280649927
Recall ratio of test data  :  0.6993350679387106
F1-measure of test data    :  0.6963312006165852
```

图 2: 预测结果

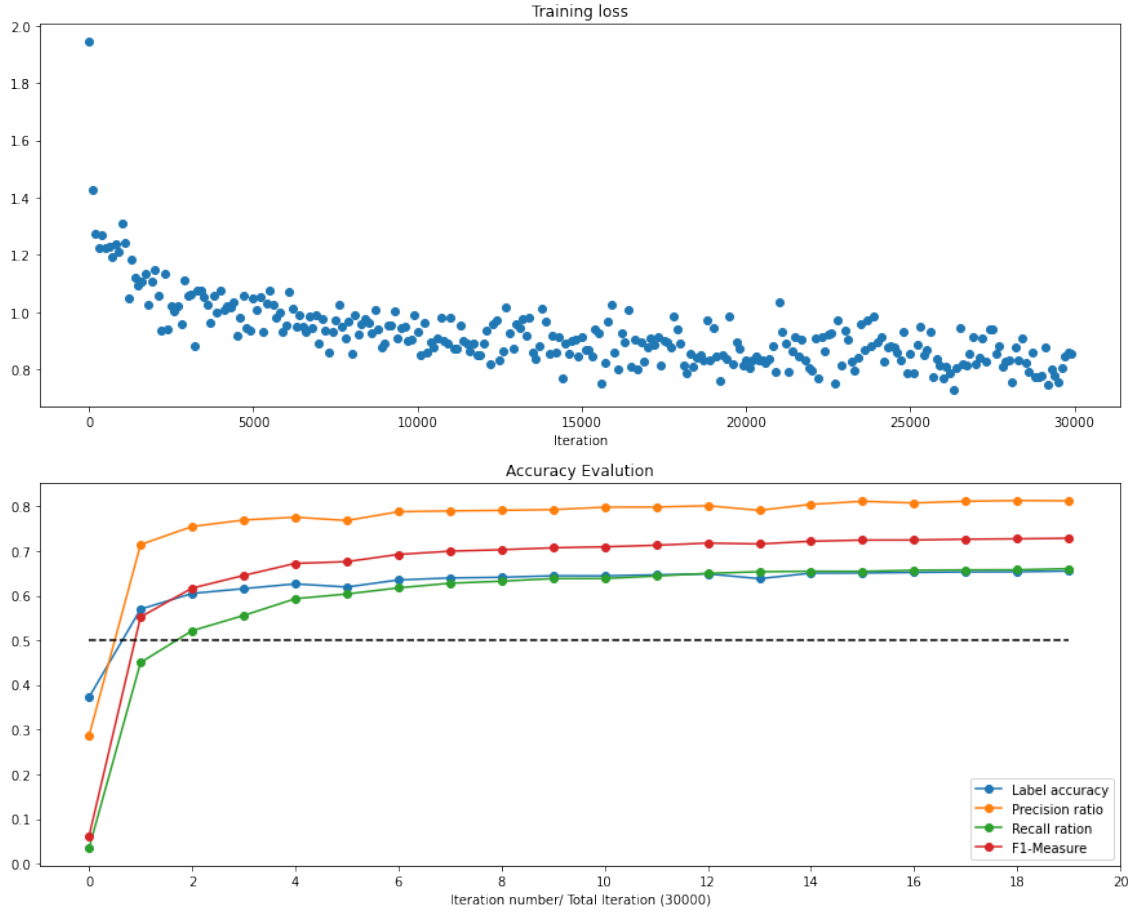

图 3: 指标变化曲线

最终：NER 模型在经过以上对训练集、词库和参数的调整后在测试集中的预测效果达到了 **F1** 指标的 **69.6%**。

5 训练测试分析及各指标调试

为了得到比较好的模型测试结果，我做了很多关于各个部分的调整优化，涉及很多数据结果的测试比较，同时也对这样的结果做出了分析讨论。在这里按照对模型结果影响程度由大到小的顺序讨论不同处理对结果的影响和原因。

5.1 迭代次数和学习率

在这个模型中，迭代次数和学习率对模型的影响很大，并且二者具有一定程度的相关性。具体而言，学习率越大，想要达到相同效果的迭代次数就越小，反之亦然。

这个模型的训练过程比较慢，而且比较独特。尽管 loss 的下降很快，在迭代到 5000 次左右就已经与最后的 loss 结果差不多了，但是 f1-measure 却一直在迭代中稳步提升，（这一结果可以看上面 **loss** 下降曲线）。因此，我先测试了迭代十万次的结果，然后像上面的图绘制出下

降曲线（与上图很像），截取了 f1-measure 非常平滑的迭代次数作为选择的最有迭代次数。最终选择了 ***30000*** 作为迭代次数。

同时，在这个模型中，想要达到比较好的效果，学习率需要取非常非常大的值，与一般使用的-2 或-3 数量级的完全不同，下表给出不同学习率在相同（30000 次）迭代下的在验证集上的结果：(注意这里的结果与上面的图表不一致，因为还有更多的处理优化)

表 2: 不同学习率下的训练结果

学习率	4×10^{-4}	3×10^{-3}	2×10^{-2}	0.1	1	2	5
准确率（正确标签比例）	84.65%	84.77%	84.96%	86.46%	89.07%	89.50%	89.65%
查准率	68.37%	72.17%	65.90%	78.90%	83.41%	82.71%	82.44%
查全率	2.18%	3.15%	5.63%	21.32	52.69%	61.23%	62.98%
F1-measure	4.23%	6.04%	10.37%	33.57%	64.58%	72.03%	71.41%

单看 F1-measure，可以得出学习率取 2 的效果最好，但是根据上面这个表，我们可以看到更多有用的信息：

准确率在学习率很低的情况下都能维持在 80% 以上并且评估模型贴标签 “正确能力” 的查准率大致与准确率成正比，说明 softmax 模型对于优化正确标签分类是很有效的。但是对于评估 **“能不能查出实体”** 能力的查全率却随着学习率的下降快速降低，这说明模型训练很大部分时间都在进行实体部分的拟合，但是整体的标签准确性已经很高了。这导致训练的大部分迭代都在内部悄悄改变参数，实际上迭代这么多次模型的改变和有效性对于我们都是一个 “黑盒”，很有可能我们一直在做无用功。

我认为背后的原因是 loss 的下降只能优化准确率和查准率，因为 softmax 的误差函数就是在评估每个标签的正确程度，这对于实体本身的识别是没用的（因为一个实体需要两个标签同时正确才算）。这也间接导致了只通过观察 loss 的大小是无法评判这个模型在命名实体识别的能力的。

对于这个问题的解决，我认为需要尝试改变误差函数本身，让误差函数能够验证一个实体的正确程度，而不是单个标签的正确程度，这样模型的训练就是直接 “奔着” 识别更多的实体而不是识别更多的标签，具有更快的训练速度和效果。(强烈建议：这块可以作为拓展研究，限于作业时间，我没有再做深入的尝试)

5.2 词库的选择

词库的调整是命名实体识别相较于其他机器学习参数优化有所不同的地方。尽管它不是训练中的超参数，但依旧对于实体识别的可靠性有着很大的影响作用。

词库的选择对于模型的作用有两个方面：

1. 决定了生成词向量特征的大小，太大会导致过大内存消耗训练过慢，太小导致特征不够精细，预测效果差

2. 影响了输入样本的编码 **“倾向”**，未被词库记录的词语统一用 Other 的向量标注，而如果很多要预测为实体的词没有在词库里，那么他们就被模型学习成了 O 类，导致结果很差

第一个因素很好理解，在调整时只需要根据自己的内存容量和训练需求选择一个尽量大的词库容量即可，根据我的测试 16G 的内存在样本数为十万左右时可以选择最大 2300 左右的词库容量，这样充分发挥词库大小可以让学习更加精细，计入的词汇更多，得到更好的结果。

第二个因素的影响很容易被忽略，但其实至关重要。首先考虑我们不应该按照全部出现词语的出现频数从高到低进行选择，因为这样的词库里有大量的词语全部是 O 类型的，其中不乏下属几类如标点符号；介词（“的”、“在”等）等等与实体识别毫无关系的词语。这会使得真正在词库中的属于 B、I 类型的有用词数量很少，大量的实体都被用 Other 的向量编码，那么训练效果就类似“盲猜”，因为模型甚至不知道输入的样本特征之间是否相同。所以首先需要分别对三类的每类实体词进行词频统计，并将它们作为主要的词库成员，保证特征的有效性。

另外，词库中也不能完全将 Other 类的词剔除，否则 O 的标签就变得无效了，同时 O 类词语对于连接实体见的语义也是有作用的，将这些高频出现的 O 词进行特殊编码有助于模型适应不同的样本，具有更好的鲁棒性和预测效果。

综合以上分析，我在同种参数条件下测试了以下几种不同的组合。（其中比如 $3\times300+100$ 意思是人名、机构、地点三类实体词中每种取最高频各 300 个，Other 类的取最高频的 100 个）

表 3: 不同词库配置下的训练结果

词库配置	$3\times400+100$	$3\times500+100$	$3\times500+300$	$3\times600+100$
准确率（正确标签比例）	54.65%	56.78%	58.42%	58.03%
查准率	65.37%	67.17%	69.85%	67.19%
查全率	46.62%	48.81%	46.30%	53.97%
F1-measure	50.07%	52.71%	51.95%	57.42%

可以看到，当 BI 类词汇在词库中比重上升时，查全率也显著上升，同时 O 的加入能够使得准确率上升，这一现象符合我们预期。在最终的结果里，我使用了 $3\times600+100$ 的词库配置。

5.3 O 类样本的去除

对于输入样本，统计后发现标签是 O 的样本在全部 100 万左右的样本中占比近 93%，也就是说模型“学到的”特征几乎全部是和 O 相关的，这也间接回答了上面的“为什么准确率超高，但是 F1 仍然很低”的原因。对于 NER，我们希望模型对标签为 B 和 I 的实体有更高的敏感性，而这样过量的 O 样本不利于模型的训练。

在此基础上，考虑到如果 O 样本全部被剔除，那么最终在验证集和测试集上准确率和查准率会很低，因为这两个数据集里面是不能剔除 O 的，需要对每类词语进行预测。因此训练集中也应该保留一部分的 O 样本，这样才能兼顾查全率和查准率，让 F1-measure 达到最大。然而，保留的 O 越多，总样本数也越多，词库能容纳的最大数量也减小，因此需要在这两者之间做出权衡。

下面给出相同参数下训练集取不同比例 O 样本后的验证集结果：

表 4: 不同词库配置下的训练结果

训练集 O 样本比例	20.9%	31.4%	41.9%	52.4%	62.8%	73.3%	93.7%(全部保留)
准确率（正确标签比例）	13.11%	17.95%	33.44%	55.66%	58.42%	63.62%	89.07%
查准率	63.89%	65.09%	66.33%	68.16%	67.17%	68.22%	73.41%
查全率	56.74%	56.32%	55.79%	55.20%	58.81%	55.65%	52.68%
F1-measure	60.10%	60.39%	60.60%	61.00%	62.71%	61.20%	61.34%

仔细观察上面的测试结果，可以看到随着 O 样本的变多，准确率有着显著的增长，但是 F1-measure 的结果却不尽然。这与上面的分析不谋而合。说明删除过量的 O 的重复样本有助于提高实体的识别能力（但是这会造成测试时部分 O 标签的错误判断，在准确率和查准率上有所折扣），整体上使得 F1-measure 的结果更好。另外，滤除 O 样本还有一个好处：减少输入样本数，节省内存空间，允许我们开去更大的词库并使得训练需要的迭代次数大幅减少。

综上，我们有理由认为去除过量的 O 输入样本是可靠并且十分有效的预处理操作。

6 个人收获总结

在命名实体识别这项任务中，我收获最多的是 NLP 领域里的数据预处理方法、调试经验和讨论分析的能力。

首先，从语言到能够被电脑训练的数值而言天生有着一条鸿沟，如何提取文字信息的特征是处理过程至关重要的一步。相较于计算机视觉里的图片，文字的特征更加复杂，更需要人为通过认知理解建立模型。同时，将如 one-hot 向量的在真正的数据集上实现也十分考验我的 python 代码能力。可以说这一部分简单但是繁琐，是我们进入 NLP 领域的第一步。

其次，这个任务极大地锻炼了我的调参找问题的能力。在起初的训练结果中，f1-measure 仅仅有千分之一，然后一步步查看实体识别数量，我意识到 softmax 并不是对于实体学习的，而是针对准确率的，因此需要提高实体标签的比例从而使得 softmax 分类起能够对大批量中很少的实体起到较好的查找作用。后来经过测试以及和同学交流又发现了学习率要比一般使用的高很多的现象等等，最终经过很多比较调整得到了较好的训练结果。

最后，我认为在这次任务中我也结合数据和理论进行了一些有效的判断分析。相较于一味地搭建网络进行复现，我第一次在机器学习领域有了探索和发现的体验，这是一种学以致用的过程，令我受益匪浅。

另外，在这里列出几个仍然没有想明白的问题，希望能够得到老师或者同学的解答或启发：

1. 为什么在命名实体识别中需要如此不同寻常大小的学习率? (样本间距? 或者什么本质因素)
2. 有没有针对实体本身的误差函数，而非针对标签的 softmax 误差作为 NER 的优化目标?
3. 词库大小极大地限制了 NER 的能力，有没有更高效的特征提取方式压缩输入数据的形状?