LSTM 中文分词大作业报告

1500012615 周哲

一、实验方法：

1. 实验目的：

通过 LSTM 长短时记忆递归神经网络来实现中文分词，并且利用提供的评测脚本评估分词算法的实际性能。

2. LSTM 进行中文分词的原理：

2.1 中文分词目标

所谓中文分词，其实可以看成是一种序列标注问题，将一段连续的中文句子（包括标点符号）通过一定的算法进行标注，得到一个按以字符为单位的标注序列。根据标注可以将句子切分成一连串独立的词。常见的两种标注模式为

a) SBME 四元标注法：S 代表单字词，B 代表词的开始，M 代表词中间的字，E 代表词结尾的字，例如：

'实/B 现/E 祖/B 国/E 的/S 完/B 全/E 统/B 一/E ，/S 是/S 海/B 内/M 外/E 全/B 体/E 中/B 国/E 人/S 的/S 共/B 同/E 心/B 愿/E 。

b) 二元标注法，只用分隔符标注是否是词的结尾，例如：

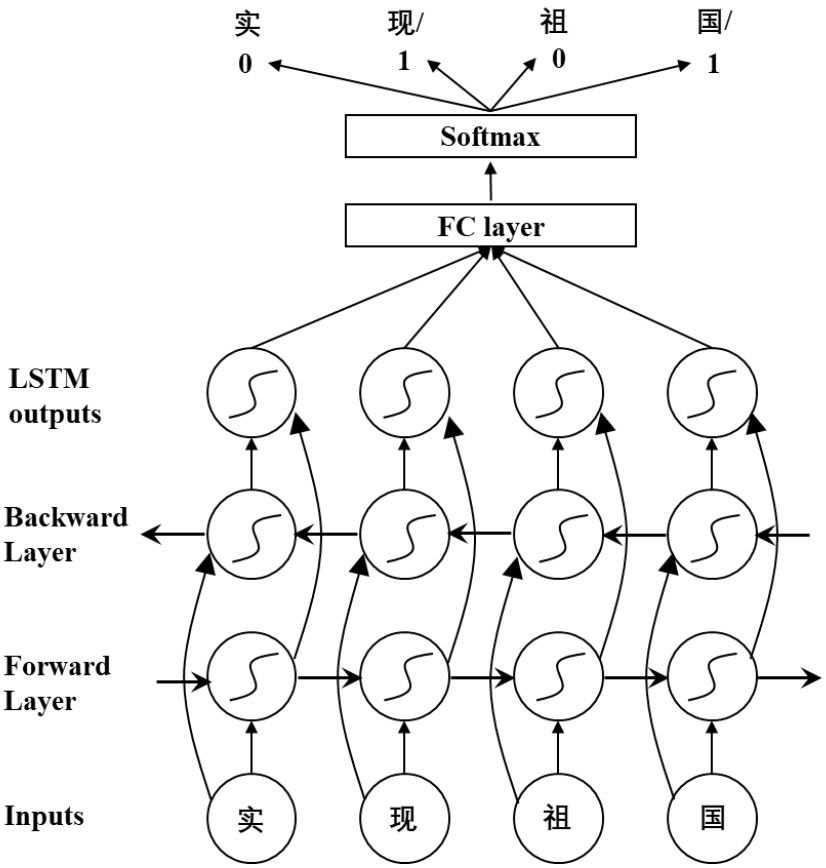
'实现/ 祖国/ 的/ 完全/ 统一/，/ 是/ 海内外/ 全体/中国/人/的/共同/心愿/。

要实现中文分词，就是对每一个字都识别出一个标注，这可以当作一个回归分类的问题来看，当然只用局部的信息进行分类往往是不够的，需要结合上下文信息，LSTM 正好可以用来从文字序列里面提取分类相关的上下文信息。

2.2 使用 LSTM 进行中文分词：

LSTM（Long Short-Term Memory）是长短期记忆网络，属于递归神经网络的一

种，以序列作为输入，能够学习到与时序相关的信息。LSTM 与传统 RNN 的区别在于 LSTM 引入了一些控制门，可以选择遗忘一些信息，对长时依赖的建模更有效。在这里我们使用双向 LSTM 来提取序列信息，双向 LSTM 分为前向和后向两个部分，前向和普通 LSTM 相同，后向则是将序列反转，然后输入到 LSTM 中。最后每个 time step 的输出结果被拼接起来作为最终的特征。为了实现标签分类，bi-LSTM 的输出之后接了一个全连接层，并且通过 softmax 函数计算每个 time step(对应了一个字符)应该被打上分割符标签的概率。示意图如下：



输入为“实现祖国”四个字的词向量，假设词向量的长度为 m，经过 bi-LSTM 后输出的是 4 个 2*hiddenSize 的 feature 向量，hiddenSize 指 LSTM 中每个 cell 中向量的长度，由于是双向 LSTM，所以要*2. 经过 FC layer 之后输出二维的向量，经过 Softmax 层输出是否是词尾（词尾对应分类标签 1）的概率。

二、实验设置

1. 实验环境

1.1 硬件环境：

处理器	GPU	RAM	ROM
Intel i9-7960x	GTX1080TI *2	32GB	2TB

1.2 软件环境：

- Ubuntu 16.04
- Python 3.5
- Pytorch 0.4
- Numpy
- Bcolz
- Matplotlib (画 loss 曲线图)

2. 数据集和预训练模型

2.1：训练数据：

仅使用课程提供的 train.txt 中的数据进行训练。包含 86924 个训练句子。

2.2：预训练的 Word Vectors （作为对照）

由于词向量对 LSTM 模型的训练有着很大的影响，而词向量既可以在训练过程中自动学习，也可以使用在更大的预料库上预训练得到的词向量。为了对比二者的差别，本实验中使用了以下预训练词向量：

https://github.com/SophonPlus/ChineseWordVectors

三、步骤：

1. 训练数据的处理：

由于训练数据是中文的字，而我们想要的训练数据应该是 x y 形式，x 为词向量序

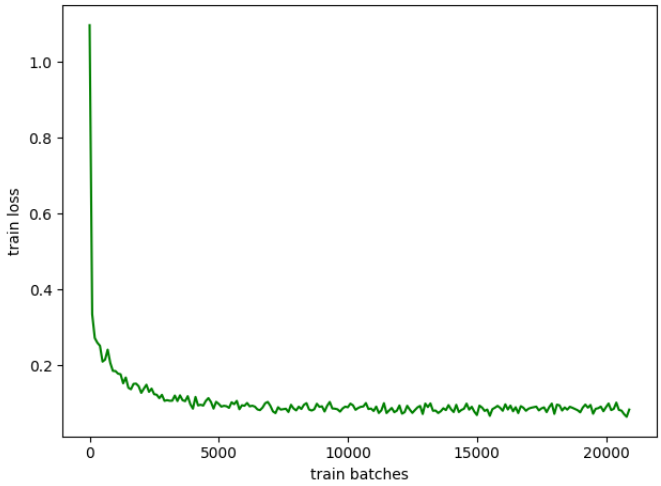
列，y 为对应的标签序列。因此需要将原始的训练数据进行处理。训练数据按行分，每行都是一个独立的句子。每个句子按空格将不同的词分隔开了，因此在处理训练数据时首先将每一行的句子按空格切分，得到独立的词，然后对每一个词中的每个字先转换为 embedding 中的字典索引值，再打上标签，如果这个字不是词的最后一个字，就打上 1，否则打上 2. 由于句子长短不一，为了训练方便，设定一个固定的 x 长度（32）对于不够长度的句子要补<pad>,对于超过长度的句子舍弃多余的部分。补<PAD> 对应形成标签为 0.

处理完数据后将其保存，以便训练时使用。

2. 使用训练数据训练模型

根据预处理的数据，训练分词模型。训练参数设置如下：

参数	**值**	**描述**
time_step	32	训练输入的序列长度
embedSize	64	词向量维度
hiddenSize	256	LSTM 隐层的维度
hiddenNum	2	LSTM 隐层的层数
epochs	10	总共训练的 epoch 数
lr decay steps	3,5,7	在哪些 epoch 的时候学习率衰减
Lr decay rate	0.1	每次学习率衰减的时候，乘以的系数
dropout	0.5	LSTM dropout 比率
batchSize	128	训练的 batch 大小
lr	1e-4	初始学习率
optimizer	adam	优化器



从训练的 loss 曲线可以看出来收敛速度还是很快的。

3. 在测试集上进行测试

3.1:测试方式：

训练完成的模型在测试集上进行测试，测试集的处理和训练集类似，但是要注意的是测试集以句子为单位进行测试，并没有长度的限制（训练集上是 32）。对于未登录词，以<PAD>为标签,如果使用的是预训练的词向量，则以<UNK> 为标签。

3.2：测试标准：

根据标准答案，测试 Precision Recall F-score

准确率（P） = 切分结果中正确分词数/切分结果中所有分词数 *100%

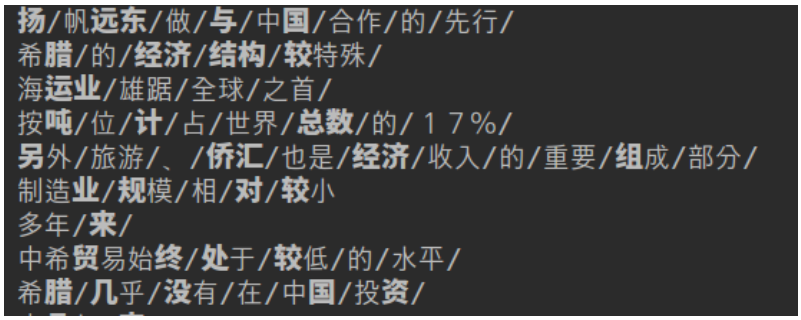
召回率（R） = 切分结果中正确分词数/标准答案中所有分词数 *100%

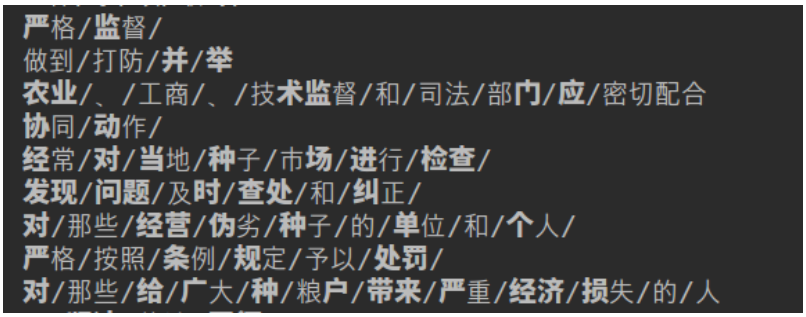
F-指标 = 2PR/(P+R)

四、实验结果

1. 结果可视化：

根据分词模型对测试集进行分词预测，其中一些结果如下：





可以明显地看到模型的确学会了如何去分词，只是在某些情况下分词并不正确，如“扬帆“，”吨位“，“并举”等。

2. 指标测试与对比

根据测试结果计算准确率，召回率和 F-指标。

在实验中共测试了未使用预训练的词向量、使用预训练词向量和 PkuSeg 三组实验。结果如下。

	Ground truth	predicted	correct	Precision	Recall	F-score
Mine	95260	91942	85562	**93.06%**	**89.81%**	**91.40%**
Mine(#)			86334	**93.90%**	**90.63%**	**92.24%**
PkuSeg			91739	**99.78%**	**96.30%**	**98.01%**

#代表使用了预训练的词向量

可以看出，完全使用给定的数据情况下，本实验中使用的双向 LSTM+FC 模型可以达到 91.40%的 F-score，而如果使用预训练的词向量，有 0.84%的性能提升，不是很明显。本模型和 PkuSeg 比有着可观的差距，约为 6%左右。

五、总结

通过本次的课程作业，我学会了如何使用 LSTM 网络构建简单的中文分词模型。并且取得了较好的效果。本次实验没有用到复杂的序列标注算法，所以准确率与 PKUSeg 有着较大的差距，还有很大的提升空间。

附录：工程代码说明

data_util.py： 用于读取和转换训练数据
network.py: 定义模型结构
plot_loss.py：用来画 loss 曲线
train.py：训练脚本，可以在 parser 里面改参数
test.py：测试脚本，可以在 parser 里面改参数，通过改变 model 参数可以测试不同的模型。改变 test_mode 参数可以改变测试的模式。

model 文件夹：存放训练产生的 model,默认里面放了两个训好的模型，一个是使用了预训练词向量，一个未使用
data 文件夹：存放了训练和测试数据，预训练词向量等。