Deep Learning

For POS

前导知识

- 参考http://neuralnetworksanddeeplearning.com/ 前三章
- 如果使用pytorch或其他框架，无需关注如何计算梯度
- Pytorch教程及安装方法见官网:https://pytorch.org/（安装命令需要翻墙才能看到）

目录

- Word embedding
- Feedforword neural network
- Feedforword neural network + CRF
- LSTM

词嵌入(word embedding)

- 如何表示一个图片？
  - 在手写数字识别的例子中，图片本身由0和1的像素组成。
  - 例如28*28的图片就可以用长度784的向量表示，输给神经网络计算。
- 词是一个字符串，那么如何表示一个词？
  - 假定我们有一个词典D，包含了N个词，每个词对应一个index
- One-hot vector(独热向量)
  - 长度为N，对应index的位置为1，例如：

$$\text{母亲}\ [1\ 0\ 0\ 0\ \dots\ 0\ 0\ 0]$$
$$\text{妈妈}\ [0\ 0\ 1\ 0\ \dots\ 0\ 0\ 0]$$

  - 缺点：高维、稀疏，词与词之间没有关联

词嵌入(word embedding)

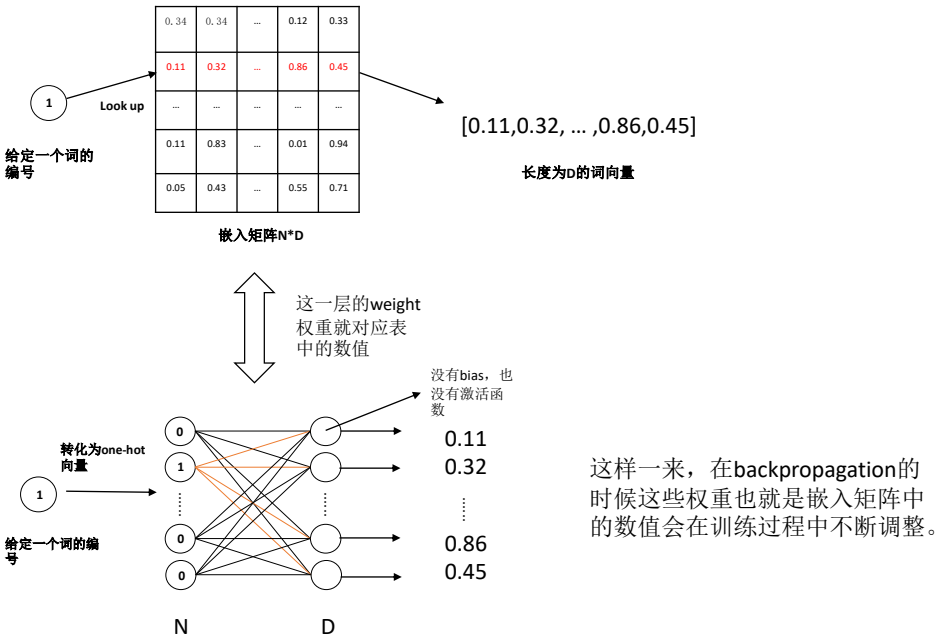
- 词嵌入
  - 这种表示词的向量通常是低维、稠密的，所有的词向量的维度都一样，每一维都可以认为在表达某种语义。例如:

$$\begin{matrix} \text{母亲} \\ \text{妈妈} \end{matrix} \begin{bmatrix} 0.13 & -0.52 & \dots & 0.93 & 0.44 \\ 0.11 & -0.49 & \dots & 0.80 & 0.46 \end{bmatrix}$$

  - 可以看出，“母亲”和“妈妈”这种含义相近的词，其向量在数值上非常接近。使用这种表示方式，便建立了一个语义向量空间，每个词都被嵌在语义空间中的一个点上，这也是其被称为词嵌入的原因。含义相近的词在语义空间中的位置也会非常接近。

词嵌入与神经网络

- 所有的词向量可以用一张二维表来存储，每一行对应一个词的词向量
- 有了一个词的index，就可以在表中查到该词的向量（look up）
- 该二维表会在训练过程中不断调整。为什么？
  - 事实上，表中的词向量就可以看成是一层全连接网络的权重。查找表的操作可以看成是通过神经网络的一层，是神经网络的一部分。
  - 示意图

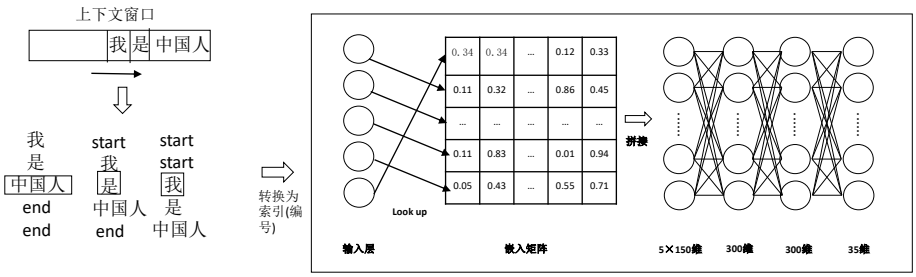


词嵌入与神经网络

- 在backpropagation的时候，每次更新的只有嵌入矩阵中的一行，即与输入为1的神经元相连的权重，也就是该词对应的词向量。
  - 为什么？(直观来讲，因为其他神经元的输入都是0所以与其相连的weight的梯度都是0)
- 如何训练出一个embedding？
  - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
  - 只需要了解，不需要自己训练。实际做的时候可以先随机初始化。
  - 两个预训练词向量下载地址 https://pan.baidu.com/s/1tU3mZjLWCJCrFyknCxtLBQ

Feedforword neural network

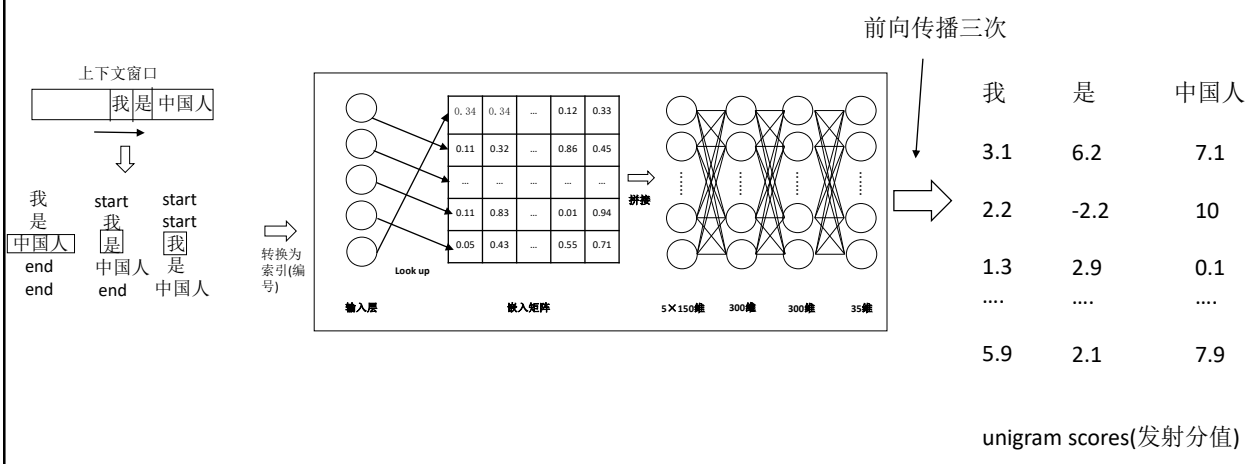
- 最简单的前向全连接的神经网络,那么如何实现POS呢？
- 看做一个简单的分类问题：
  - 手写数字图片识别：给定一张图片，预测该图是0-9（类别）中的哪个
  - POS：给定一个词，预测该词属于哪个词性（类别）
- 但是词的词性是离不开句子的，单单只看一个词（其词向量）根本无法预测。因此我们要把该词放在句子中，同时看它前后的词。
- 所以输入为一个词及其前后两个词，转换为词向量后拼接起来（作为该词的特征），输入到隐藏层，最后输出的是对应每个词性的概率（分值）。
- 示意图



- 假设每个词用150维向量表示，共有35个可能的词性
- 隐藏层此处示例为2层，各300，实际用一层可能效果好
- 输出层可以用softmax函数得到对应标签的概率。直接取最大值作为预测的结果。
- 实际上还是以一个一个词为单位来预测的。

Feedforword neural network + CRF

- 以句子为单位
  - 给定一个词序列，输出一个词性序列
- 需要一个bigram score矩阵（转移分值矩阵）
  - 随机初始化，作为模型的一部分，会随着训练调整
- 结果用viterbi算法预测
- 如何计算loss？



	我	是	中国人
词性1	3.1	6.2	7.1
词性2	2.2	-2.2	10
词性3	1.3	2.9	0.1
	….	….	….
词性T	5.9	2.1	7.9

unigram scores(发射分值)

	词性1	词性2	词性3
词性1	3.1	6.2	7.1
词性2	2.2	-2.2	10
词性3	1.3	2.9	0.1
	….	….	….
词性T	5.9	2.1	7.9

bigram scores(转移分值)

Forword算法

$$Z = \sum_{Y \in T^n} e^{score(S,Y)}$$

根据正确的词性序列

$$e^{score(S,Y^j)}$$

$$loss = -\log(e^{score(S,Y^j)}/Z)$$

Feedforword neural network + CRF

- 训练完后，根据bigram score和unigram score用viterbi算法预测最优词性序列。
- Bigram score 矩阵会随着训练而调整。

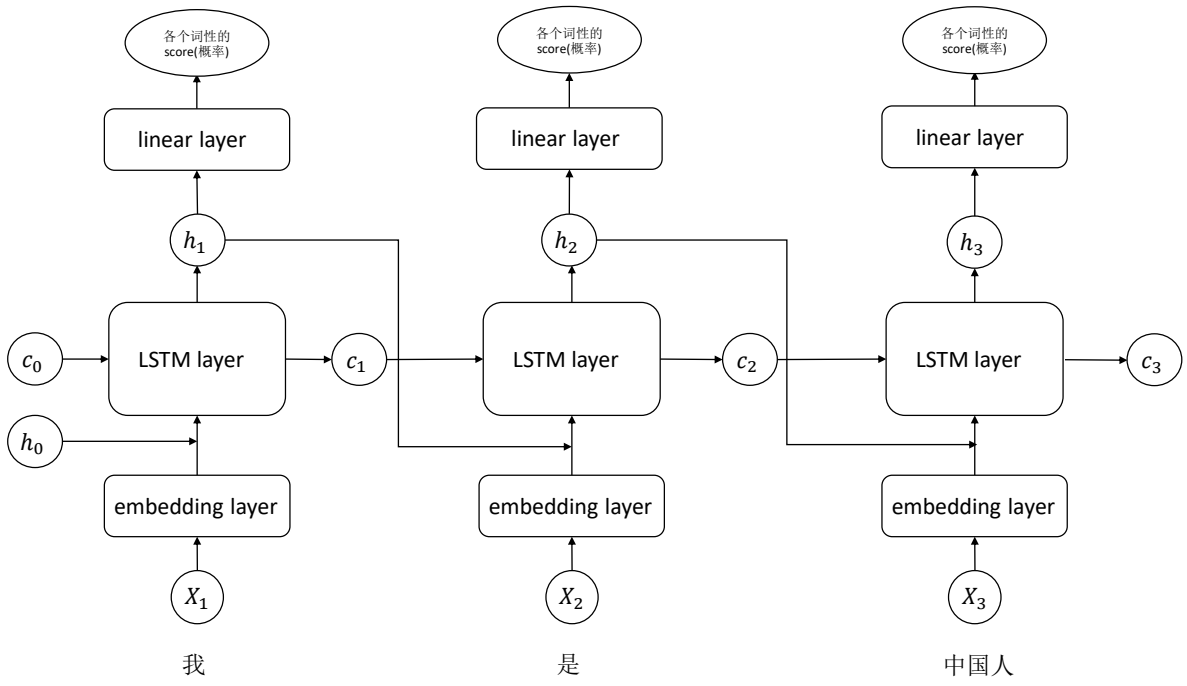
结果

数据集为conll格式，来自CTB7。
训练集：train.conll,共46,572个句子,共1,057,943个词。
开发集：dev.conll,共2,097个句子,共59,955个词。
测试集：test.conll,共2,796个句子,共81,578个词。
预训练词向量：base_embeddings.txt

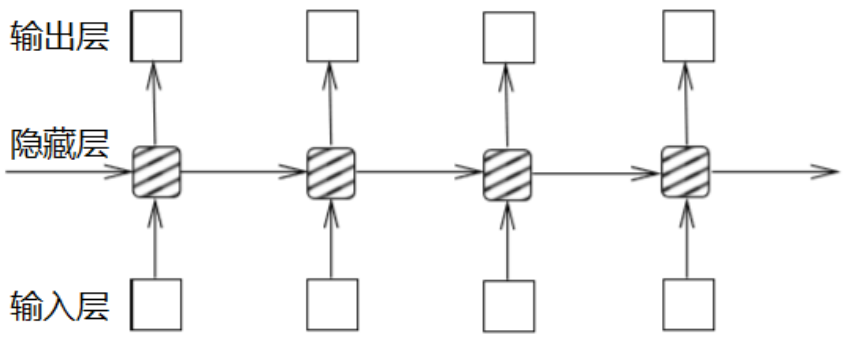
model	dev准确率	test准确率	迭代	时间/迭代
FFNN	92.87%	92.61%	24/34	~3min
FFNN+CRF	93.14%	92.77%	25/35	~9min

LSTM

- 参考资料
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2018/Lecture/Seq%20(v2).pdf
- (NAACL2016)Neural Architectures for Named Entity Recognition


各个词性的score(概率)
各个词性的score(概率)
各个词性的score(概率)
linear layer
linear layer
linear layer
h_1
h_2
h_3
c_0
LSTM layer
c_1
LSTM layer
c_2
LSTM layer
c_3
h_0
embedding layer
embedding layer
embedding layer
X_1
X_2
X_3
我
是
中国人

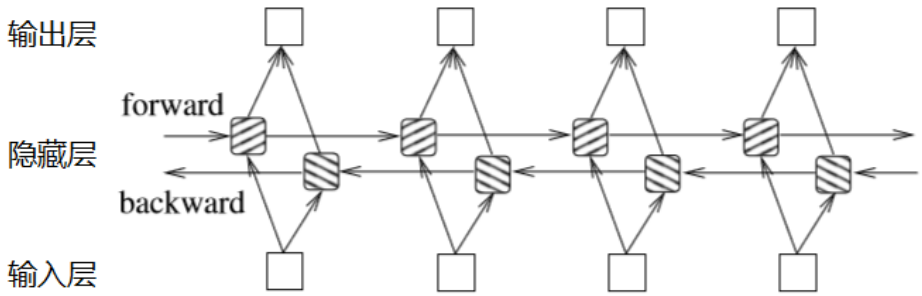
LSTM简化图


输出层
隐藏层
输入层

Bi-LSTM

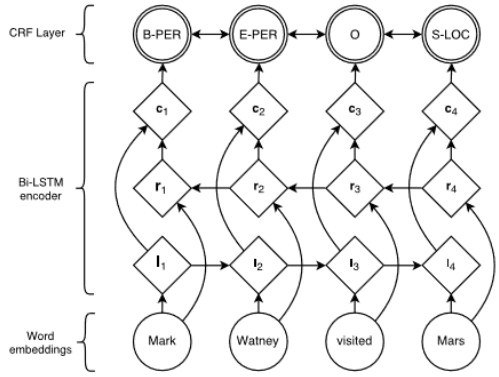
- 双向LSTM网络由一个正向LSTM网络和一个反向LSTM网络组成
- 它们的结构一致，只是一个正向传播信息，另一个反向传播信息。

Bi-LSTM简易图



Bi-LSTM+CRF

Main architecture of the network. Word embeddings are given to a bidirectional LSTM. li represents the word i and its left context, ri represents the word i and its right context. Concatenating these two vectors yields a representation of the word i in its context, ci.



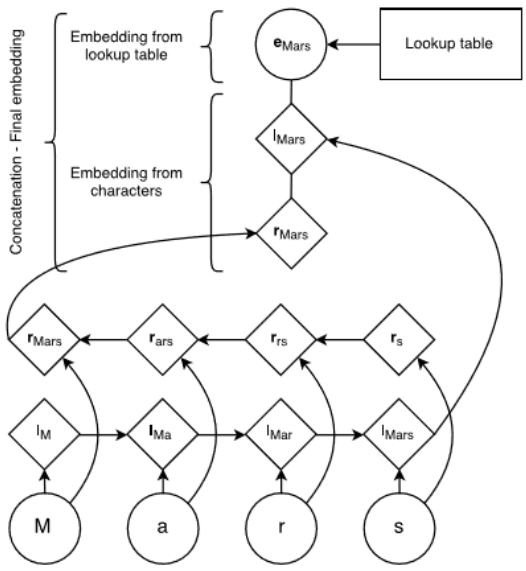
优化

- Dropout
- 加入字符级特征
  - 1.词向量拼上该词第一个字或者最后一个字的字向量
  - 2.aver-pooling:词向量拼上该词所有字的字向量之和的平均
  - 3.利用一层char lstm获取字符级特征，再拼上词向量输入到下一层lstm
  - ……

Char Level LSTM

- 一个词首先拆成字符通过一层BiLSTM
- 把最后一个时间点的output拿出来拼在词的word embedding上
- 把拼接起来的特征输入到下一层BiLSTM

The character embeddings of the word “Mars” are given to a bidirectional LSTMs. We concatenate their last outputs to an embedding from a lookup table to obtain a representation for this word.



LSTM in Pytorch

- 需要对句子做padding才能实现minibatch方法
  - 最佳方法是只对一个batch里的句子padding。
  - 对整个训练集做padding也没问题(此处问题不大)虽然省事但是会浪费内存。
- 实现char lstm的时候也要对词做padding
  - char embedding矩阵随机初始化
  - 此处如果padding成训练集里最长的词内存会浪费极多。因为词的数量多，其中长的词非常少。
  - 直接把长的词后面部分截掉
- 双向LSTM只需改一下参数和维度
- 在词向量与字符特征拼接的地方加上dropout

结果

数据集为conll格式，来自CTB5。
训练集：train.conll,共16,091个句子,共437,991个词。
开发集：dev.conll,共803个句子,共20,454个词。
测试集：test.conll,共1,910个句子,共50,319个词。
预训练词向量：giga.100.txt

model	dev准确率	test准确率	迭代	时间/迭代
Bilstm(2层)+crf	94.82%	94.30%	21/31	9min
first char embedding+bilstm(2层)+crf	95.05%	94.57%	20/30	9min
aver-pooling+bilstm(2层)+crf	95.35%	94.93%	28/38	10min
charlstm+wordlstm+crf	95.69%	95.29%	20/30	6min(优化的CRF)

结果

数据集为conll格式，来自CTB7。
训练集：train.conll,共46,572个句子,共1,057,943个词。
开发集：dev.conll,共2,097个句子,共59,955个词。
测试集：test.conll,共2,796个句子,共81,578个词。
预训练词向量：giga.100.txt

model	dev准确率	test准确率	迭代	时间/迭代
BiLstm(2层)+crf	94.95%	94.75%	17/27	23min
first char embedding+BiLstm(2层)+crf	95.28%	95.06%	32/42	23min
aver-pooling+BiLstm(2层)+crf	95.57%	95.25%	16/26	24min
charlstm+wordlstm+crf	95.66%	95.39%	18/28	19min(优化过了CRF)