

## 8.11 - 8.17

2020年8月17日 10:02

### 一周小结

本周主要针对agnews数据集对新闻分类问题进行了了解，基于keras框架，利用TextRNN得到约88%的准确率，用预训练好的词向量作为嵌入层权重准确率达到90%，用Text CNN得到了超过92.5%的准确率。

(在华为云ModelArts的notebook中运行的，GPU 1\*p100，源码见附件，主要文件是agnews\_train.ipynb，以及model/agnews\_model.py)

### 主要过程：

#### 1. agnews数据读取与预处理

从csv文件读取数据并存入列表，利用keras的分词函数进行分词并转成numpy数组，得到的数组结构如下：





```
Loading data complete.
shape of x_train tensor: (120000, 150)
shape of y_train tensor: (120000, 4)
shape of x_test tensor: (7600, 150)
shape of y_test tensor: (7600, 4)
Tokenizing complete.
```

训练集包括了120000条数据，测试集包括了7600条数据

这里，由于数据集的平均长度为38（部分论文称平均长度为45，但我测试后为38），数据集最大长度为180，因此取了最大序列长度150

#### 2. 读取预训练好的词向量（嵌入矩阵）

主要利用的词向量来源：glove\_vector: glove.6B.50d.txt等

名称	修改日期	类型	大小
 glove.6B.50d.txt	2014/8/5 4:15	文本文档	167,335 KB
 glove.6B.100d.txt	2014/8/5 4:14	文本文档	338,982 KB
 glove.6B.200d.txt	2014/8/5 4:14	文本文档	677,181 KB
 glove.6B.300d.txt	2014/8/28 3:19	文本文档	1,013,636...

主要过程应该是：从4000000个词中选出现次数最多的前

20000(max\_num\_words)个词作为嵌入向量，在嵌入层中将词向量作为权重使用

(这里我不知道理解的是否正确)

```
embedding_layer = Embedding(num_words + 1, embedding_dim,
weights=[embedding_matrix],
input_length=max_sequence_length)
```

#### 3. 经过了预处理后，得到了数据集，测试集，以及一个训练好的嵌入矩阵

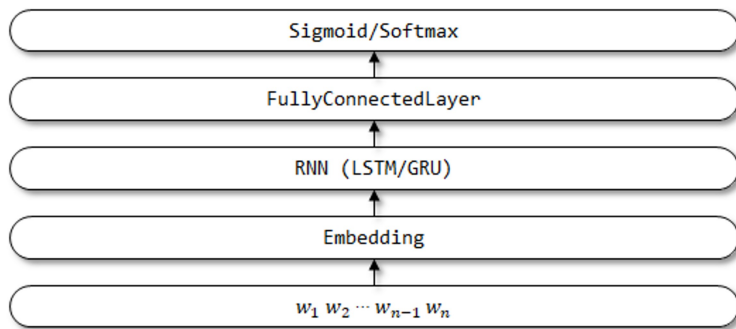
```
Loading data complete.
shape of x_train tensor: (120000, 150)
shape of y_train tensor: (120000, 4)
shape of x_test tensor: (7600, 150)
shape of y_test tensor: (7600, 4)
Tokenizing complete.
embedding_matrix shape: (20001, 100)
```

我利用上述这些数据进行训练与测试

#### 4. TextRNN

这里我先没有使用训练好的词向量

网络简单结构：嵌入层（2维训练数据转为3维）--> LSTM(128) --> Dense(4, softmax)



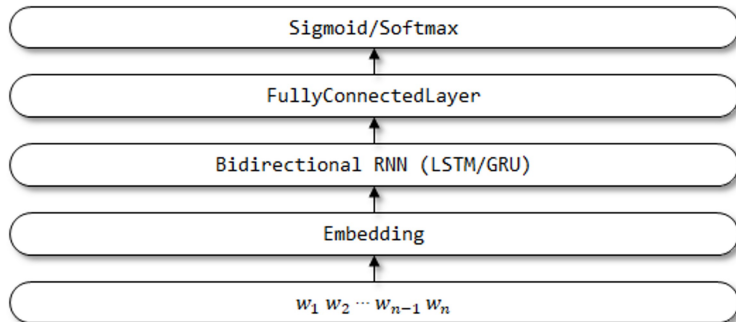
得到的结果：

```
Build model...
Train...
Train on 120000 samples, validate on 7600 samples
Epoch 1/3
120000/120000 [=====] - 59s 495us/step - loss: 0.4720 - acc: 0.8383 - val_loss: 0.3905 - val_acc: 0.8766
Epoch 2/3
/home/ma-user/anaconda3/envs/TensorFlow-1.8/lib/python3.6/site-packages/keras/callbacks.py:569: RuntimeWarning: Early stopping conditioned on metric `val_accuracy` which is not available. Available metrics are: val_loss, val_acc, loss, acc
(self.monitor, ', '.join(list(logs.keys()))), RuntimeWarning
120000/120000 [=====] - 53s 438us/step - loss: 0.3515 - acc: 0.8859 - val_loss: 0.3942 - val_acc: 0.8770
Epoch 3/3
120000/120000 [=====] - 53s 438us/step - loss: 0.3459 - acc: 0.8870 - val_loss: 0.4068 - val_acc: 0.8657
Test...
result label: [1 0 3 ... 2 3 0]
y_label: [3 0 0 ... 2 3 3]
```

## 5. TextBiRNN

没有使用训练好的词向量

网络结构：Embedding --> Bi-LSTM --> Dense



得到的结果：

```
Build model...
Train...
Train on 120000 samples, validate on 7600 samples
Epoch 1/3
120000/120000 [=====] - 104s 865us/step - loss: 0.7128 - acc: 0.7295 - val_loss: 0.4263 - val_acc: 0.8554
Epoch 2/3
120000/120000 [=====] - 100s 834us/step - loss: 0.3758 - acc: 0.8775 - val_loss: 0.4108 - val_acc: 0.8655
Epoch 3/3
120000/120000 [=====] - 100s 837us/step - loss: 0.3602 - acc: 0.8822 - val_loss: 0.4147 - val_acc: 0.8629
Test...
test set accuracy: 0.8628947368421053
```

结果反而没有单向的LSTM效果好，可能参数还需要优化，这里我没有进一步调整了

## 6. LSTM（实际上这是我第一个构建出来并进行测试的网络）

这里我利用了预训练的词向量作为嵌入层权重进行训练

因为这是第一个使用的网络，所以做了比较多的调整，网络相对复杂了一些

网络结构：

embedding\_layer --> LSTM(100, dropout=0.2) --> BN --> Dense(20) -->

dropout(0.2) --> relu() --> Dense(4)

训练结果：

```

Build model...
Training...
Train on 120000 samples, validate on 7600 samples
Epoch 1/5
120000/120000 [=====] - 59s 491us/step - loss: 0.5701 - acc: 0.7990 - val_loss: 0.3778 - val_acc: 0.8861
Epoch 2/5
120000/120000 [=====] - 56s 468us/step - loss: 0.2977 - acc: 0.9068 - val_loss: 0.3633 - val_acc: 0.8967
Epoch 3/5
120000/120000 [=====] - 56s 465us/step - loss: 0.2231 - acc: 0.9287 - val_loss: 0.3582 - val_acc: 0.8941
Epoch 4/5
120000/120000 [=====] - 56s 467us/step - loss: 0.1826 - acc: 0.9408 - val_loss: 0.3835 - val_acc: 0.8967
Epoch 5/5
120000/120000 [=====] - 56s 468us/step - loss: 0.1589 - acc: 0.9483 - val_loss: 0.3920 - val_acc: 0.8951
7600/7600 [=====] - 1s 172us/step
Test score: 0.3919618063067135
Test accuracy: 0.8951315809237329

```

这里使用的是100维的词向量，准确率可以达到89.5%以上，我也尝试了用300维的向量，准确率可以达到90%

这里实际上有一个尚未理解的问题，这个网络是参考了网上的源码，同时我发现很多代码中，在LSTM层后面，都有一层全连接层，且这个全连接层的神经元数量为1，这是为什么？

```

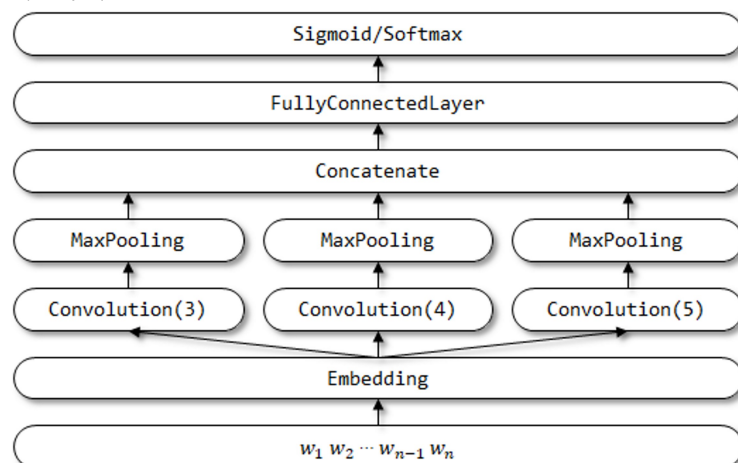
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid')) # 这是为什么？明明更多的神经元可以有更好的效果

```

我一开始使用这个网络结构，准确率只能达到82%，而一旦将神经元数量调整为20，准确率马上上升到89%左右。

## 7. TextCNN网络

网络结构：



训练结果：

```

Build model...
Train...
Train on 120000 samples, validate on 7600 samples
Epoch 1/5
120000/120000 [=====] - 10s 80us/step - loss: 0.2045 - acc: 0.9195 - val_loss: 0.1230 - val_acc: 0.9581
Epoch 2/5
120000/120000 [=====] - 9s 73us/step - loss: 0.0851 - acc: 0.9711 - val_loss: 0.1113 - val_acc: 0.9609
Epoch 3/5
120000/120000 [=====] - 9s 73us/step - loss: 0.0490 - acc: 0.9847 - val_loss: 0.1119 - val_acc: 0.9621
Epoch 4/5
120000/120000 [=====] - 9s 74us/step - loss: 0.0266 - acc: 0.9929 - val_loss: 0.1151 - val_acc: 0.9625
Epoch 5/5
120000/120000 [=====] - 9s 73us/step - loss: 0.0144 - acc: 0.9967 - val_loss: 0.1221 - val_acc: 0.9627
Test...
result label: [1 3 1 1 1 2 3 1 0 2 2 1 0 1 2 1 2 1 0 3]
y label:      [1 1 1 1 1 2 3 1 0 2 2 1 0 1 2 1 2 1 0 3]
-----
result label: [2 1 2 2 1 2 1 2 1 1 1 2 3 2 2 2 1 2 1 1]
y label:      [2 1 2 2 1 2 1 2 1 1 1 2 3 2 2 2 1 2 1 1]
test set accuracy: 0.9248684210526316

```

最后的验证集准确率达到了96%，但我最后又测了一次却只有92.5%左右，这里有点没明白，但卷积网络效果很好。

总结：

实际上在一开始我打算参照XLNet论文中的几个准确率比较高的算法进行复现，但可能受限于个人能力，无论是哪种网络都复现不出来。。。似乎每种模型都对

算力有要求？这里没暂时还没找到有效的方法来复现，我打算下一周再尝试一下  
~

Model	IMDB	Yelp-2	Yelp-5	DBpedia	AG	Amazon-2	Amazon-5
CNN [15]	-	2.90	32.39	0.84	6.57	3.79	36.24
DPCNN [15]	-	2.64	30.58	0.88	6.87	3.32	34.81
Mixed VAT [31, 23]	4.32	-	-	0.70	4.95	-	-
ULMFIT [14]	4.6	2.16	29.98	0.80	5.01	-	-
BERT [35]	4.51	1.89	29.32	0.64	-	2.63	34.17
XLNet	<b>3.20</b>	<b>1.37</b>	<b>27.05</b>	<b>0.60</b>	<b>4.45</b>	<b>2.11</b>	<b>31.67</b>

Table 4: Comparison with state-of-the-art error rates on the test sets of several text classification datasets. All BERT and XLNet results are obtained with a 24-layer architecture with similar model sizes (aka BERT-Large).

(引自XLNet论文)