人工神经网络实验报告

RNN

张钰晖

2015011372, yuhui-zh15@mails.tsinghua.edu.cn, 185-3888-2881

目录

1 问题描述

语句层面的情感分类是自然语言处理 (NLP) 领域中经典的研究问题, 斯坦福数据集 (Stanford Sentiment Treebank, SST) 是该研究领域经典的数据集，提供了 11855 句样例，并被分为大小 8544/1101/2210 的训练集、验证集、测试集。下面展示了其中的几句样本。

```
3 The Rock is destined to be the 21st Century 's new `` Conan '' and that he 's going to make a splash even greater than Arnold Schwarzenegger , Jean-Claud Van Damme or Steven Segal .
4 The gorgeously elaborate continuation of `` The Lord of the Rings '' trilogy is so huge that a column of words can not adequately describe co-writer\/director Peter Jackson 's expanded vision of J.R.R. Tolkien 's Middle-earth .
3 Singer\/composer Bryan Adams contributes a slew of songs -- a few potential hits , a few more simply intrusive to the story -- but the whole package certainly captures the intended , er , spirit of the piece .
2 You 'd think by now America would have had enough of plucky British eccentrics with hearts of gold .
3 Yet the act is still charming here .
4 Whether or not you 're enlightened by any of Derrida 's lectures on `` the other '' and `` the self , '' Derrida is an undeniably fascinating and playful fellow .
4 Just the labour involved in creating the layered richness of the imagery in this chiaroscuro of madness and light is astonishing .
3 Part of the charm of Satin Rouge is that it avoids the obvious with humour and lightness .
4 a screenplay more ingeniously constructed than `` Memento ''
```

图 1: SST 数据样本

在本次作业中，笔者将采用目前机器学习领域最为流行的框架 Tensorflow，实现循环神经网络 RNN、GRU 与 LSTM，来完成这一分类任务。

2 具体实现

2.1 词向量导入的实现

数据提供了经过训练的 17530 个词向量，每个词被 embed 为 300 维向量，下面展示了其中一个单词 the 的词向量。

```
the 0.27204 -0.06203 -0.1884 0.023225 -0.018158 0.0067192 -0.13877 0.17708 0.17709 2.5882 -0.35179 -0.17312 0.43285 -0.10708
0.15006 -0.19982 -0.19093 1.1871 -0.16207 -0.23538 0.003664 -0.19156 -0.085662 0.039199 -0.066449 -0.04209 -0.19122 0.011679
-0.37138 0.21886 0.0011423 0.4319 -0.14205 0.38059 0.30654 0.020167 -0.18316 -0.0065186 -0.0080549 -0.12063 0.027507 0.29839
-0.22896 -0.22882 0.14671 -0.076301 -0.1268 -0.0066651 -0.052795 0.14258 0.1561 0.05551 -0.16149 0.09629 -0.076533 -0.049971
-0.010195 -0.047641 -0.16679 -0.2394 0.0050141 -0.049175 0.013338 0.41923 -0.10104 0.015111 -0.077706 -0.13471 0.119 0.10802
0.21061 -0.051904 0.18527 0.17856 0.041293 -0.014385 -0.082567 -0.035483 -0.076173 -0.045367 0.089281 0.33672 -0.22099 -0.0067275
0.23983 -0.23147 -0.88592 0.091297 -0.012123 0.013233 -0.25799 -0.02972 0.016754 0.01369 0.32377 0.039546 0.042114 -0.088243
0.30318 0.087747 0.16346 -0.40485 -0.043845 -0.040697 0.20936 -0.77795 0.2997 0.2334 0.14891 -0.39037 -0.053086 0.062922 0.065663
-0.13906 0.094193 0.10344 -0.2797 0.28905 -0.32161 0.020687 0.063254 -0.23257 -0.4352 -0.017049 -0.32744 -0.047064 -0.075149
-0.18788 -0.015017 0.029342 -0.3527 -0.044278 -0.13507 -0.11644 -0.1043 0.1392 0.0039199 0.37603 0.067217 -0.37992 -1.1241
-0.057357 -0.16826 0.03941 0.2604 -0.023866 0.17963 0.13553 0.2139 0.052633 -0.25033 -0.11307 0.22234 0.066597 -0.11161 0.062438
-0.27972 0.19878 -0.36262 -1.0006e-05 -0.17262 0.29166 -0.15723 0.054295 0.06101 -0.39165 0.2766 0.057816 0.39709 0.025229 0.24672
-0.08905 0.15683 -0.2096 -0.22196 0.052394 -0.01136 0.050417 -0.14023 -0.042825 -0.031931 -0.21336 -0.20402 -0.23272 0.07449
0.088202 -0.11063 -0.33526 -0.014028 -0.29429 -0.086911 -0.1321 -0.43616 0.20513 0.0079362 0.48505 0.064237 0.14261 -0.43711
0.12783 -0.13111 0.24673 -0.27496 0.15896 0.43314 0.090286 0.24662 0.066463 -0.20099 0.1101 0.03644 0.17359 -0.15689 -0.086328
-0.17316 0.36975 -0.40317 -0.064814 -0.034166 -0.013773 0.062854 -0.17183 -0.12366 -0.034663 -0.22793 -0.23172 0.239 0.27473
0.15332 0.10661 -0.060982 -0.024805 -0.13478 0.17932 -0.37374 -0.02893 -0.11142 -0.08389 -0.055932 0.068039 -0.10783 0.1465
0.094617 -0.084554 0.067429 -0.3291 0.034082 -0.16747 -0.25997 -0.22917 0.020159 -0.02758 0.16136 -0.18538 0.037665 0.57603 0.20684
0.27941 0.16477 -0.018769 0.12062 0.069648 0.059022 -0.23154 0.24095 -0.3471 0.04854 -0.056502 0.41566 -0.43194 0.4823 -0.051759
-0.27285 -0.25893 0.16555 -0.1831 -0.06734 0.42457 0.010346 0.14237 0.25939 0.17123 -0.13821 -0.066846 0.015981 -0.30193 0.043579
-0.043102 0.35025 -0.19681 -0.4281 0.16899 0.22511 -0.28557 -0.1028 -0.018168 0.11407 0.13015 -0.18317 0.1323
```

图 2: 词向量

通过导入词向量初值，可以加速神经网络的收敛速度。

实际有 18430 个词需要 embed，对于数据中没有的 900 个词，采用全部置 0 的方式实现 embed。

```
embed = []
embed_dict = {}
with open('%s/vector.txt' % (path)) as f:
   for line in f:
      splitline = line.strip().split(' ')
      word = splitline[0]
      vector = []
      for i in range(FLAGS.embed_units):
         vector.append(float(splitline[i + 1]))
      embed_dict[word] = vector
zero_vector = []
for i in range(FLAGS.embed_units):
   zero_vector.append(float(0))
for word in vocab_list:
   if word in embed_dict:
      embed.append(embed_dict[word])
   else:
      embed.append(zero_vector)
```

2.2 模型细节的实现

首先实现 tensorflow 的 placeholder，由于训练时 batch size 大小可变，可以直接采用置 None 的方式让系统自动判断大小。

```
self.texts = tf.placeholder(tf.string, shape=[None, None]) # shape: batch*len
self.texts_length = tf.placeholder(tf.int64, shape=[None]) # shape: batch
self.labels = tf.placeholder(tf.int64, shape=[None]) # shape: batch
```

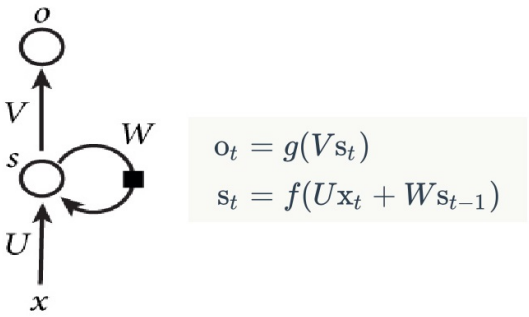
其次是 logits 的实现，在 RNN 输出层与最终结果之间应该接入一个全连接层，tensorflow 提供了全连接层的实现 tf.layers.dense。

```
logits = tf.layers.dense(states, num_labels)
```

2.3 RNNCell 的实现

2.3.1 BasicRNNCell 的实现

BasicRNN 是最基础的 RNN，其原理如下图所示。



$$o_t = g(Vs_t)$$
$$s_t = f(Ux_t + Ws_{t-1})$$

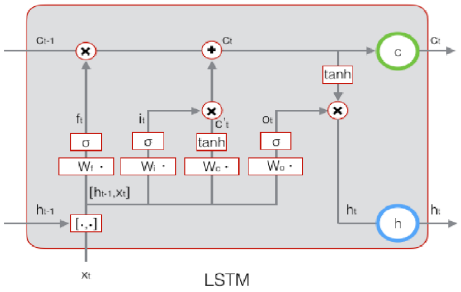
图 3: BasicRNN

在实现过程中，对图中 U 矩阵和 W 矩阵使用 tf.concat 进行了拼接，这样可以简化为一个矩阵相乘，代码实现如下。

```
def __call__(self, inputs, state, scope=None):
   with tf.variable_scope(scope or "basic_rnn_cell", reuse=self._reuse):
      W = tf.get_variable('W', [FLAGS.embed_units + self._num_units, self._num_units])
      b = tf.get_variable('b', [self._num_units], initializer=tf.constant_initializer(0.0))
      new_state = self._activation(tf.matmul(tf.concat([inputs, state], axis=1), W) + b)
   return new_state, new_state
```

2.3.2 BasicLSTMCell 的实现

LSTM 是 BasicRNN 的变种，本质也属于 RNN。由于 BasicRNN 存在梯度消失和梯度爆炸的问题，无法实现多层传递，而 LSTM 很好的解决了这个问题，其原理如下图所示。



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \circ \tanh(c_t)$$

图 4: BasicLSTM

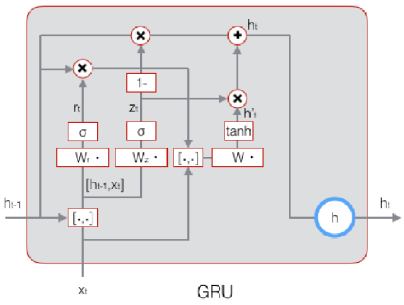
在实现过程中，变量命名和图中公式保持一致，同时对图中矩阵使用 tf.concat 进行了拼接，这样可以简化为一个矩阵相乘，再使用 tf.split 拆分为所需要的部分，大大简化代码实现，

代码如下。

```
def __call__(self, inputs, state, scope=None):
    with tf.variable_scope(scope or "basic_lstm_cell", reuse=self._reuse):
        c, h = state
        W = tf.get_variable('W', [FLAGS.embed_units + self._num_units, self._num_units * 4])
        b = tf.get_variable('b', [self._num_units * 4], initializer=tf.constant_initializer
            (0.0))
        f, i, o, c_tilde = tf.split(value=tf.matmul(tf.concat([inputs, h], axis=1), W) + b,
            num_or_size_splits=4, axis=1)
        new_c = tf.sigmoid(f + self._forget_bias) * c + tf.sigmoid(i) * self._activation(
            c_tilde)
        new_h = tf.sigmoid(o) * self._activation(new_c)
    return new_h, (new_c, new_h)
```

2.3.3 GRUCell 的实现

GRU 是 LSTM 的变种，本质也属于 RNN。由于 LSTM 实现复杂，GRU 对 LSTM 做了很多简化，同时却保持着和 LSTM 相同的效果，GRU 最近变得越来越流行，其原理如下图所示。



$$
\begin{aligned}
\mathbf{z}_t &= \sigma(W_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \\
\mathbf{r}_t &= \sigma(W_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \\
\tilde{\mathbf{h}}_t &= \tanh(W \cdot [\mathbf{r}_t \circ \mathbf{h}_{t-1}, \mathbf{x}_t]) \\
\mathbf{h} &= (1 - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \tilde{\mathbf{h}}_t
\end{aligned}
$$

图 5: GRU

在实现过程中，变量命名和图中公式保持一致，同时对图中矩阵使用 tf.concat 进行了拼接，这样可以简化为一个矩阵相乘，再使用 tf.split 拆分为所需要的部分，大大简化代码实现，代码如下。

```
def __call__(self, inputs, state, scope=None):
    with tf.variable_scope(scope or "gru_cell", reuse=self._reuse):
        W_zr = tf.get_variable('W_zr', [FLAGS.embed_units + self._num_units, self._num_units *
             2])
        b_zr = tf.get_variable('b_zr', [self._num_units * 2], initializer=tf.
            constant_initializer(1.0))
```

```
        z, r = tf.split(value=tf.sigmoid(tf.matmul(tf.concat([inputs, state], axis=1), W_zr) +
             b_zr), num_or_size_splits=2, axis=1)
        W = tf.get_variable('W', [FLAGS.embed_units + self._num_units, self._num_units])
        b = tf.get_variable('b', [self._num_units], initializer=tf.constant_initializer(0.0))
        h = self._activation(tf.matmul(tf.concat([inputs, r * state], axis=1), W) + b)
        new_h = z * state + (1 - z) * h
    return new_h, new_h
```

2.4 Tensorboard 的实现

由于框架已经使用 tensorflow 实现了保存训练集数据，增加两个新的 tf.Summary 简单扩充后即可实现保存验证集和测试集的数据，代码如下。

```
        # 保存验证集
        summary_dev = tf.Summary()
        summary_dev.value.add(tag='loss/dev', simple_value=loss)
        summary_dev.value.add(tag='accuracy/dev', simple_value=accuracy)
        summary_writer.add_summary(summary_dev, epoch)
        # 保存测试集
        summary_test = tf.Summary()
        summary_test.value.add(tag='loss/test', simple_value=loss)
        summary_test.value.add(tag='accuracy/test', simple_value=accuracy)
        summary_writer.add_summary(summary_test, epoch)
```

2.5 双向 LSTM 的实现

双向 LSTM 相比单向 LSTM 而言，考虑了文本正向反向双重关系，因此可以利用更充足的文本信息。

由于已经实现了 BasicLSTMCell，借助 tensorflow 的框架可以简单修改实现双向 LSTM。

```
cell_fw = BasicLSTMCell(num_units)
cell_bw = BasicLSTMCell(num_units)
outputs, states = tf.nn.bidirectional_dynamic_rnn(cell_fw, cell_bw, self.embed_input, self.
    texts_length, dtype=tf.float32, scope="rnn")
(cell_fw, hidden_fw), (cell_bw, hidden_bw) = states
logits = tf.layers.dense(tf.concat([hidden_fw, hidden_bw], axis=1), 256)
logits = tf.layers.dropout(logits)
logits = tf.layers.dropout(tf.layers.dense(logits, num_labels))
```

3 单层 RNN 训练曲线

训练曲线刻画了准确率 Accuracy 与损失 Loss 随训练次数变化的曲线。

上一节实现了 BasicRNN、BasicLSTM、GRU 三种循环神经网络，本节均采用规模为 512 的单层 RNN 网络测试情感分类问题，利用 tensorboard 绘出训练曲线。

网络参数如下：

表 1: 网络参数

损失函数	SoftmaxCrossEntropyLoss
词向量维度	300
RNN 单元数	512
学习率	0.001
批大小	16

表 2: RNN 网络结构

层	**规模**
RNN	输入: $N * 300$
	输出: $N * 512$
Linear	输入: $N * 512$
	输出: $N * 5$

3.1 BasicRNN

下图展示了 BasicRNN 的 Loss-Epoch 曲线，平滑率 0.3：

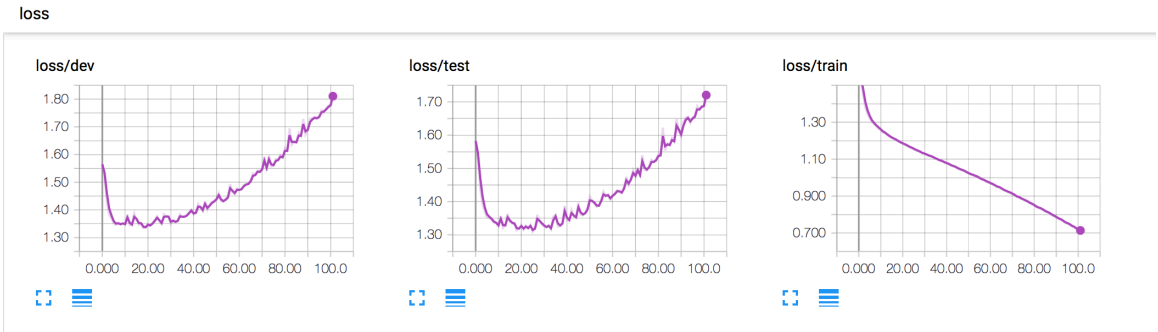


图 6: Loss-Epoch 曲线

下图展示了 BasicRNN 的 Accuracy-Epoch 曲线，平滑率 0.3：

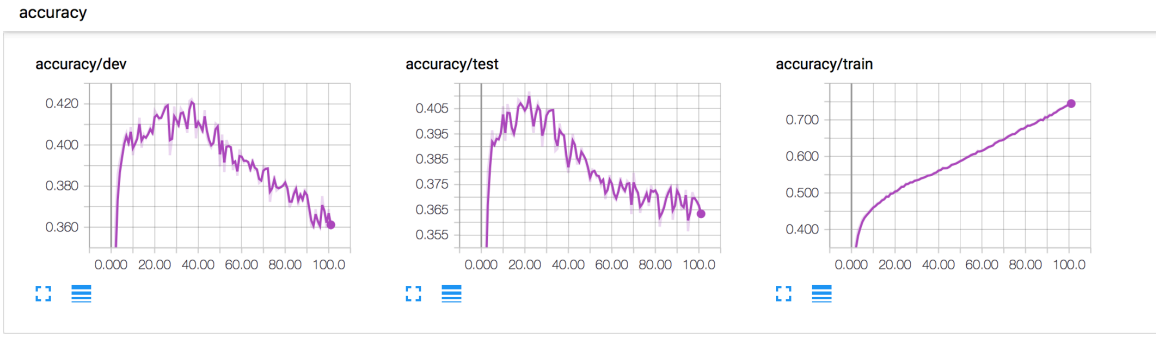


图 7: Accuracy-Epoch 曲线

从训练曲线可以看出，BasicRNN 存在严重的过拟合问题，即模型对训练集拟合加深，对测试集拟合减弱，训练一段时间后测试集准确率不升反降，最终模型在训练集上准确率远远高于测试集上准确率。

BasicRNN 在测试集准确率的峰值为 41.18%。

3.2 BasicLSTM

下图展示了 BasicLSTM 的 Loss-Epoch 曲线，平滑率 0.3：

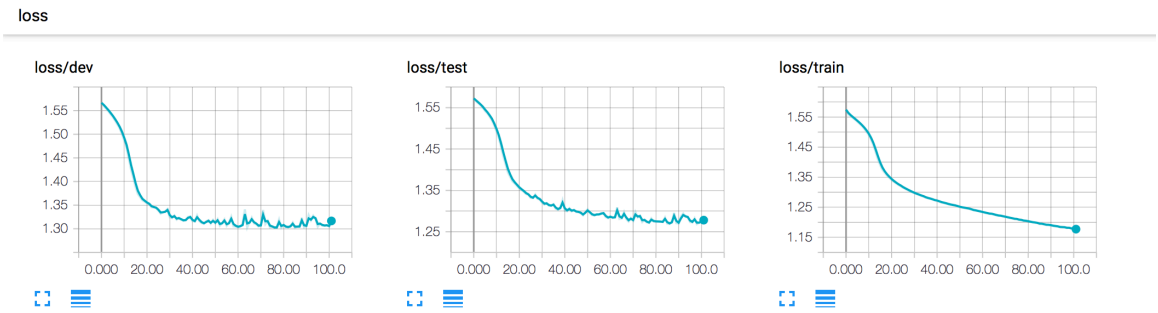


图 8: Loss-Epoch 曲线

下图展示了 BasicLSTM 的 Accuracy-Epoch 曲线，平滑率 0.3：

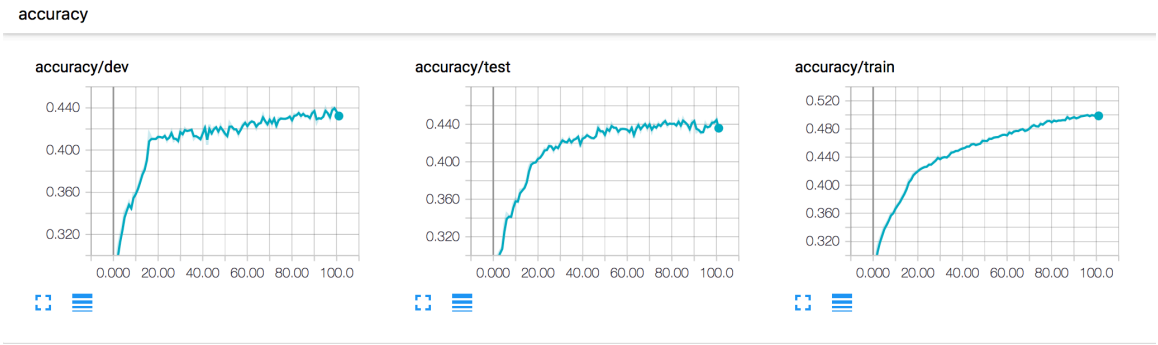


图 9: Accuracy-Epoch 曲线

从训练曲线可以看出，BasicLSTM 存在轻微的过拟合问题，模型收敛性相对较好。BasicLSTM 在测试集准确率的峰值为 44.61%。

3.3 GRU

下图展示了 GRU 的 Loss-Epoch 曲线，平滑率 0.3：

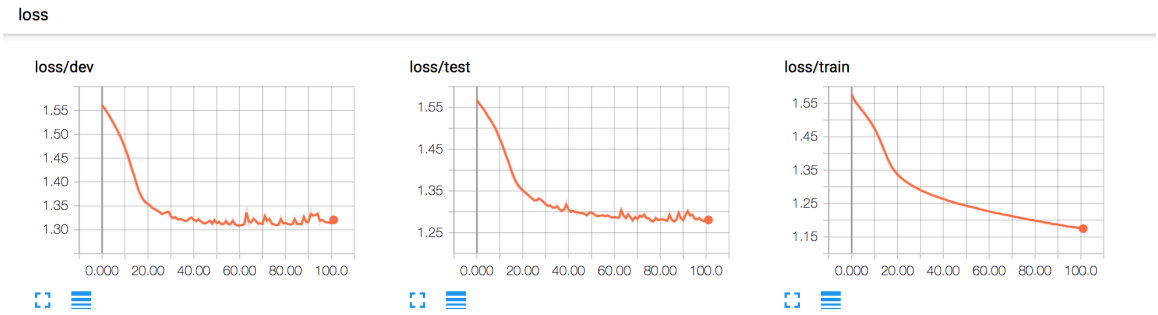


图 10: Loss-Epoch 曲线

下图展示了 GRU 的 Accuracy-Epoch 曲线，平滑率 0.3：

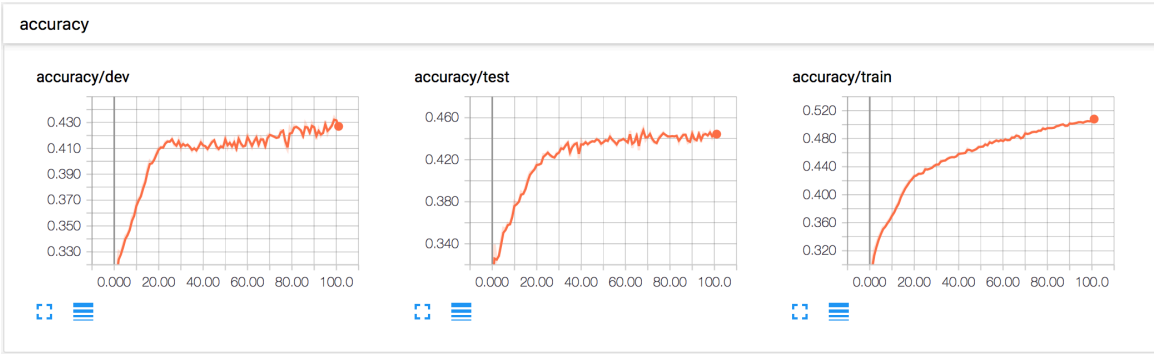


图 11: Accuracy-Epoch 曲线

从训练曲线可以看出，GRU 存在轻微的过拟合问题，模型收敛性相对较好。GRU 在测试集准确率的峰值为 45.02%。

4 单层 RNN 对比分析

4.1 训练曲线

下图将 BasicRNN、BasicLSTM、GRU 的 Loss-Epoch 绘于一张图中便于对比分析，平滑率 0.6：

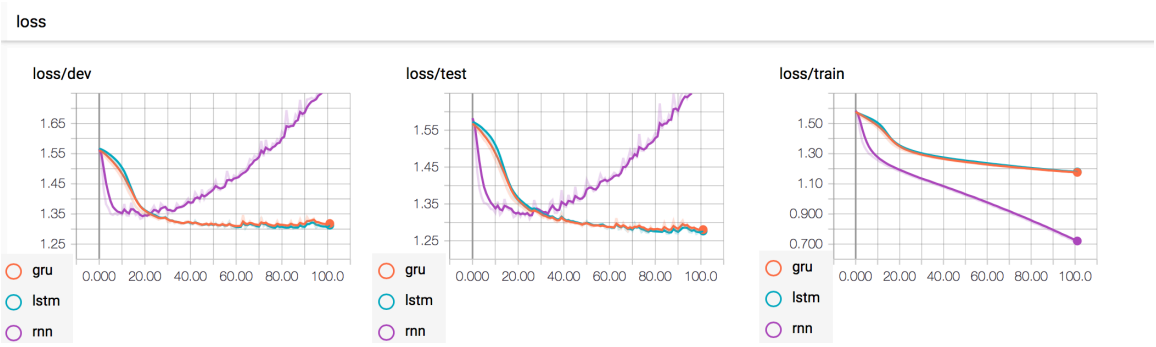


图 12: Loss-Epoch 曲线

下图将 BasicRNN、BasicLSTM、GRU 的 Accuracy-Epoch 绘于一张图中便于对比分析，平滑率 0.6：

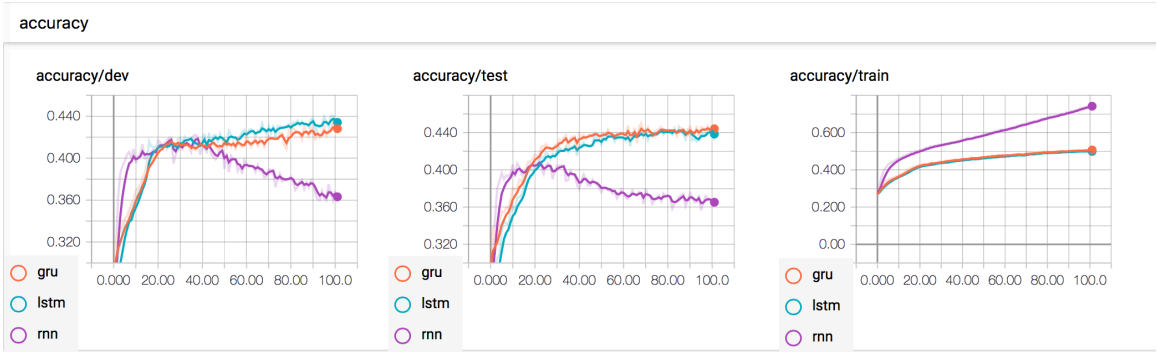


图 13: Accuracy-Epoch 曲线

4.2 训练结果

下面这张表格从准确率、收敛速度、训练速度等方面对比了 BasicRNN、BasicLSTM 和 GRU，所有网络均训练 100 个 Epoch：

表 3: 对比

	BasicRNN	**BasicLSTM**	**GRU**
最高准确率 (训练集)	**74.54%**	50.02%	50.94%
最终准确率 (训练集)	**74.54%**	49.81%	50.94%
最高准确率 (验证集)	42.23%	**44.05%**	43.42%
最终准确率 (验证集)	35.88%	**43.05%**	42.51%
最高准确率 (测试集)	41.18%	44.61%	**45.02%**
最终准确率 (测试集)	36.20%	43.21%	**44.43%**
收敛性	差	**较好**	**较好**
过拟合	严重	**轻微**	**轻微**
收敛速度	不收敛	**一般**	**一般**
训练速度 (GPU 服务器)	**较快 (约 13s/epoch)**	较慢 (约 38s/epoch)	一般 (约 28s/epoch)

4.3 结果分析

以上选取了 BasicRNN、BasicLSTM 和 GRU 进行对比，通过不同方面的对比，我们可以清楚地看到三者之间的异同。

- 从测试集准确率可以看出，GRU 和 BasicLSTM 在测试集上准确率接近，GRU 准确率略

高于 BasicLSTM，但都显著高于 BasicRNN。从结果来看，单层 RNN 网络中，GRU 和 BasicLSTM 优于 RNN。

- 从训练集准确率可以看出，三种网络的训练集准确率都明显高于测试集准确率，因此都存在一定的过拟合。但是 BasicRNN 的训练集准确率远远高于其测试集准确率，过拟合非常严重，而 BasicLSTM 和 GRU 则相对过拟合比较轻微。
- 从收敛性和收敛速度可以看出 (此处收敛速度是指训练稳定时所需迭代次数)，BasicRNN 不收敛，在训练一段时间后出现严重过拟合，BasicLSTM 和 GRU 收敛性相对较好，但是收敛速度都比较一般，需要训练 20 个 Epoch 以上才渐渐收敛。
- 从训练速度可以看出 (此处训练速度是指训练一个 1Epoch 所需时间)，BasicRNN 因为模型相对较为简单从而速度最快，而 BasicLSTM 计算相对非常复杂，因此速度最慢，GRU 在 BasicLSTM 的基础上进行了适度的简化，速度得到的些许的提升。

整体上来看，GRU 在 BasicRNN 和 BasicLSTM 间兼顾了性能与效果，效果最好。

5 网络构建

本小节中，笔者将融合人工神经网络所学知识，尝试自己设计与搭建模型提高情感分类的准确率。

最终，笔者选择了双向 LSTM 模型作为最终模型，其最终准确率达到了 45.52%。

5.1 模型选择

最终采用的模型如下，基本框架是一个双向 LSTM，后面接两层全连接层，同时使用 Dropout 层防止过拟合。

表 4: 网络结构

层	规模
Bi-LSTM	输入: $N * 300$
	输出: $N * 512$
Linear	输入: $N * 512$
	输出: $N * 256$
Dropout	输入: $N * 256$
	输出: $N * 256$
	保留率: 0.5
Linear	输入: $N * 256$
	输出: $N * 5$

5.2 训练曲线

训练曲线刻画了准确率 Accuracy 与损失 Loss 随训练次数变化的曲线，本节利用 tensorboard 绘出 BiLSTM 的训练曲线和 BasicLSTM 的训练曲线进行对比，平滑率 0.6。

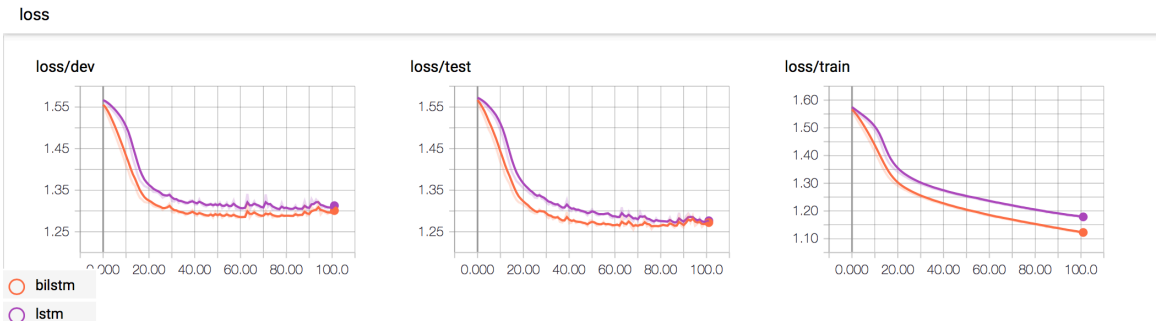


图 14: Loss-Epoch 曲线

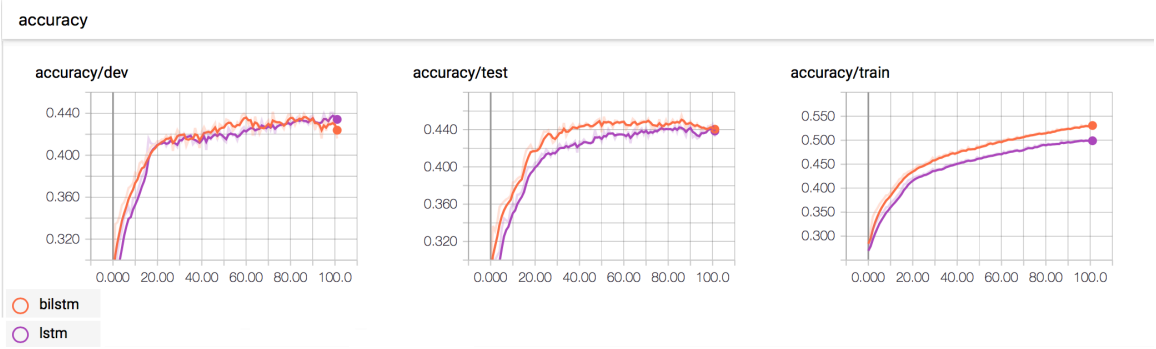


图 15: Accuracy-Epoch 曲线

5.3 最终结果

表 5: 对比

	单向 LSTM	**双向 LSTM**
最高准确率 (训练集)	50.02%	**52.91%**
最终准确率 (训练集)	49.81%	**52.91%**
最高准确率 (验证集)	44.05%	**44.14%**
最终准确率 (验证集)	43.05%	**43.69%**
最高准确率 (测试集)	44.61%	**45.52%**
最终准确率 (测试集)	43.21%	**44.16%**
收敛性	**较好**	**较好**
过拟合	**轻微**	**轻微**
收敛速度	**一般**	**一般**
训练速度 (GPU 服务器)	**较慢 (约 38s/epoch)**	更慢 (约 58s/epoch)

- 从测试集准确率可以看出，双向 LSTM 准确率比单向 LSTM 更高。
- 从训练集准确率可以看出，无论是双向 LSTM 还是单向 LSTM 训练集准确率都高于测试集准确率，因此都存在一定的过拟合，但都比较轻微。
- 从收敛性和收敛速度可以看出 (此处收敛速度是指训练稳定时所需迭代次数),单向 LSTM 和双向 LSTM 收敛性相对较好，但是收敛速度都比较一般，需要训练 20 个 Epoch 以上才渐渐收敛。

- 从训练速度可以看出 (此处训练速度是指训练一个 1Epoch 所需时间)，双向 LSTM 计算量基本为单向 LSTM 两倍，因此速度更慢。

综合来看，双向 LSTM 效果相比单向 LSTM 较好。

5.4 模型分析

单向 LSTM，只根据文本前面信息推出后面信息，但有时候只看前面的词是不够的。

例如，我今天不舒服，我打算 (?) 一天。

只根据' 不舒服'，可能推出我打算' 去医院'，' 睡觉'，' 请假' 等等，但如果加上后面的' 一天'，能选择的范围就变小了，' 去医院' 这种就不能选了，而' 请假" 休息' 之类的被选择概率就会更大。

双向 LSTM，可以充分利用前后信息，正向计算得到值 A，反向计算得到值 A'，最终的输出值 y 取决于 A 和 A'，其结构如下图所示。

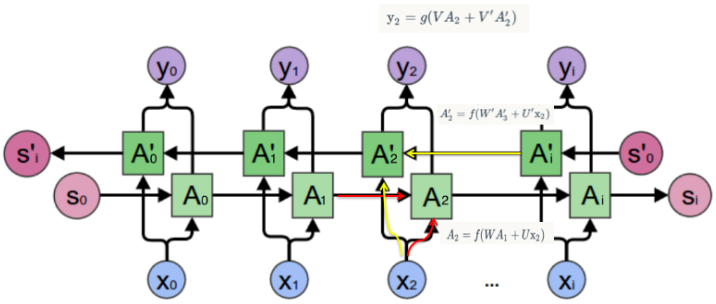


图 16: BiLSTM

因此，双向 LSTM 理应比单向 LSTM 更高，从实验结果也印证了这一事实。

在实现过程中，构建两个 BasicLSTMCell，利用 tf.nn.bidirectional_dynamic_rnn 实现。

6 参数选择

为了达到更高的准确率，笔者常数进行调参，由于调参是一个非常费力费时间的工作，笔者尝试改变大量参数进行测试，最终选出结果相对较好的一组参数，如下表所示。

表 6: 参数选择

超参数	值	解释
学习率	1e-3%	控制梯度下降速度
梯度下降器	GradientDescentOptimizer	采用 Adam 上升速率更快，但效果很差
批大小	16	控制梯度下降速率与稳定性
RNN 隐层节点数	512	更高或更低准确率都下降
MLP 隐层层数	1	更高或更低准确率都下降
MLP 隐层节点数	256	更高或更低准确率都下降

7 实验心得

这次实验让我第一次亲自实现了 RNN、LSTM、GRU, 通过本次实验, 我对 RNN、LSTM、GRU 的理解进一步得到加深，也步入了 NLP 领域的大门。使用 Tensorflow 搭建框架大大简化了作业难度，我也更加深入的理解了 Tensorflow。

我也尝试搭建了不同的模型并改变不同的参数，最终实现了准确率为 45.52% 的双向 LSTM，也更加理解了科研的过程和调参的艰辛。

不过对我而言，这依旧只是一个入门，还需要进一步的练习与实践。

参考文献

[1] 人工神经网络, 黄民烈, 2017.

[2] Recurrent Neural Network Regularization, *Wojciech Zaremba, Ilya Sutskever, Oriol Vinyals*, 2015.

[3] Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, *Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio*, 2014.

[4] 零基础入门深度学习, *Han Bingtao*, 2017.

[5] Tensorflow 源代码, *Google*, 2017.