Lab2 循环神经网络

和泳毅 PB19010450

一、实验目标

1. 编写RNN的语言模型，并基于训练好的词向量，编写RNN模型用于文本分类。
   本次实验内容；
2. 实现SimpleRNN, LSTM, GRU(基于训练好的词向量Glove)， 并对结果进行对比；
3. 对比单层SimpleRNN、两层RNN、BRNN、DBRNN；
4. 对比单层LSTM、两层LSTM、BLSTM、DBLSTM；
5. 对比单层GRU、两层GRU、BGRU、DBGRU；
6. 对比隐藏神经元数量对结果的影响；

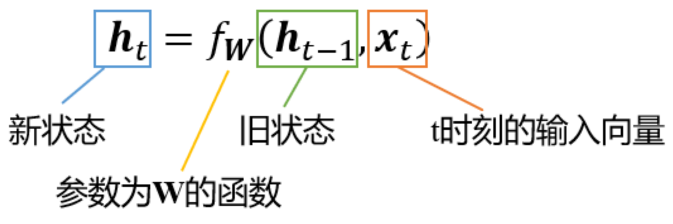
二、数据集介绍

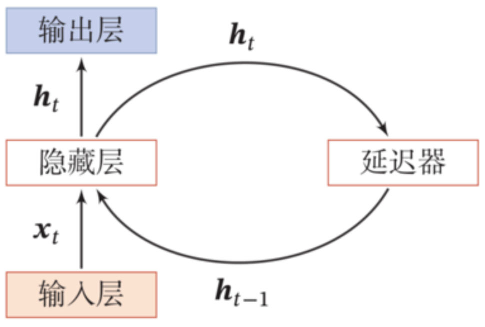
Tiny Imagenet是斯坦福大学提供的图像分类数据集，其中包含200个类别，每个类别包含500张训练图像，50张验证图像及50张测试图像。

三、实验原理

1.循环神经网络（Recurrent Neural Network）

循环神经网络通过使用带自反馈的神经元，能够处理任意长度的时序数据

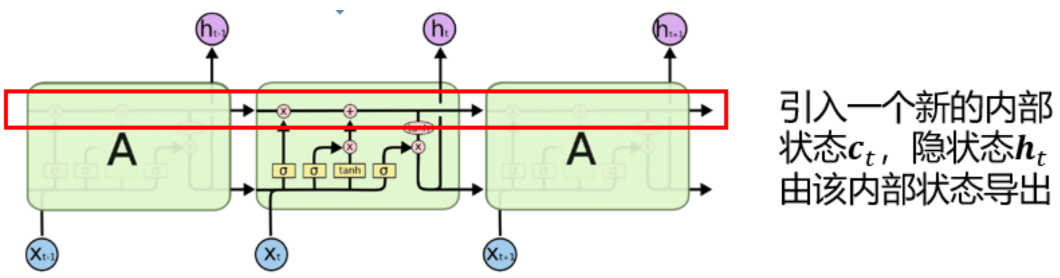


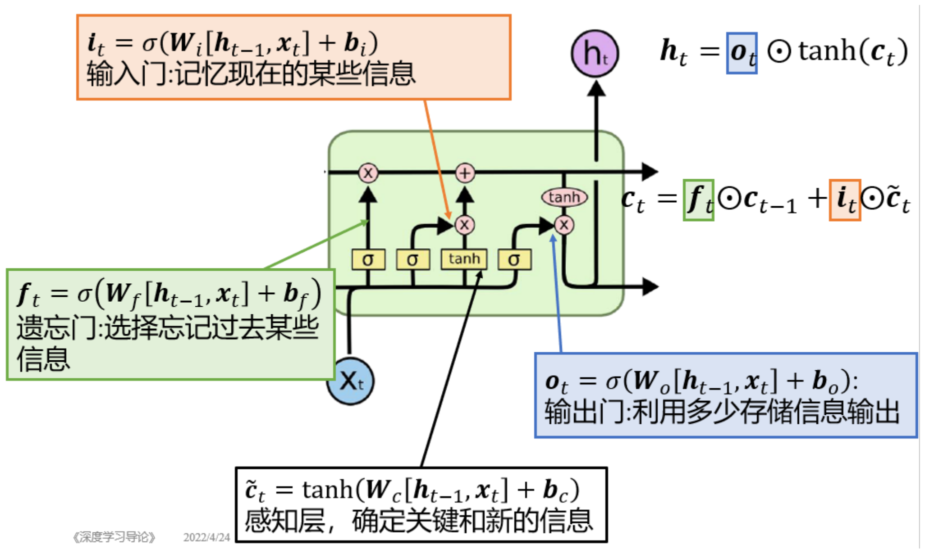


状态 h_t“存储”直到t时刻的历史数据$\{x_1, \cdots, x_t\}$，

$$\boldsymbol{h}_t = f_W(f_W(f_W(\boldsymbol{h}_{t-3}, \boldsymbol{x}_{t-2}), \boldsymbol{x}_{t-1}), \boldsymbol{x}_t)$$

2、长短期记忆神经网络（Long Short-Term Memory, LSTM ）





$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$
输入门:记忆现在的某些信息
$h_t = o_t \odot \tanh(c_t)$
$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$
遗忘门:选择忘记过去某些信息
$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$:
输出门:利用多少存储信息输出
$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$
感知层，确定关键和新的信息
《深度学习导论》 2022/4/24

$$\begin{bmatrix} \tilde{c}_t \\ o_t \\ i_t \\ f_t \end{bmatrix} = \begin{bmatrix} \tanh \\ \sigma \\ \sigma \\ \sigma \end{bmatrix} \left(W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b \right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot \tanh(c_t)$$

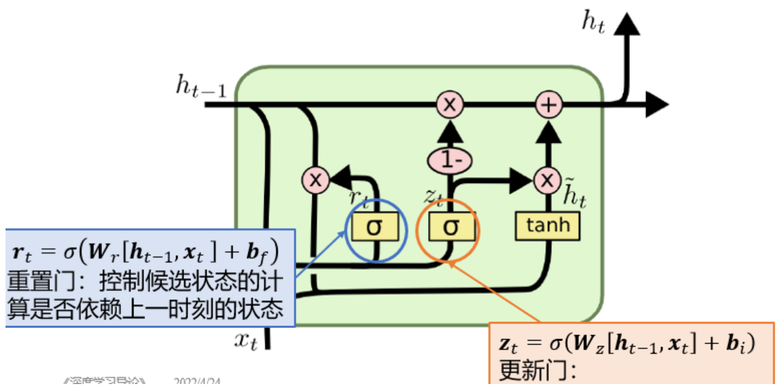
LSTM网络中，记忆单元c可以在某个时刻捕捉到关键信息，并有能力保存一定的时间间隔，因此生命周期要长期短期记忆，因此称为长短期记忆。

3. 门控循环单元（GRU）

统一输出单元(h_t)和记忆单元(c_t)，引入更新门控制当前状态需要从历史状态中保留多少信息以及需要从候选状态中接收多少新信息。

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \widetilde{h}_t$$

$$\widetilde{h}_t = \tanh(W_h [r_t \odot h_{t-1}, x_t] + b_c)$$


h_t
h_{t-1}
r_t
z_t
$\tilde{h}_t$
1-
σ
σ
tanh
x_t
$r_t = \sigma(W_r[h_{t-1}, x_t] + b_f)$
重置门：控制候选状态的计算是否依赖上一时刻的状态
$z_t = \sigma(W_z[h_{t-1}, x_t] + b_i)$
更新门：

四、核心代码

1. 文件读入

```python
imdb_dir = r"C:/Users/Administrator/Desktop/Lab2/lab2/data/imdb/aclImdb"
train_dir = os.path.join(imdb_dir,"train")

labels = []
texts = []

for label_type in ["neg","pos"]:
    dir_name = os.path.join(train_dir,label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == ".txt":
            with open(os.path.join(dir_name,fname),encoding='UTF-8') as f:
                texts.append(f.read())
            if label_type == "neg":
                labels.append(0)
            else:
                labels.append(1)
```

设定最大词数与句长。

```python
maxlen = 300
training_samples = 20000
validation_samples = 5000
max_words = 5000

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index

```

```python
data = sequence.pad_sequences(sequences,maxlen=maxlen)
labels = np.asarray(labels)
indices = np.arange(0,data.shape[0])
np.random.shuffle(indices)

data = data[indices ]
labels = labels[indices]
x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples:training_samples+validation_samples]
y_val = labels[training_samples:training_samples+validation_samples]
```

2.GloVe

```python
glove_dir = r"C:/Users/Administrator/Desktop/Lab2/lab2/vector_cache"
embeddings_index = {}
f = open(os.path.join(glove_dir,"glove.6B.100d.txt"),encoding="utf-8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:],dtype="float32")
    embeddings_index[word] = coefs
f.close()
print(len(embeddings_index))

embedding_dim = 100

embedding_matrix = np.zeros((max_words,embedding_dim))
for word,i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

五、实验结果

统一设置优化器与目标函数：

```
model.compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = ["acc"]
)
```

早停机制与学习率衰减:

```
ES=EarlyStopping(monitor='val_acc',patience=5)
filepath='best.hdf5'
checkpoint = ModelCheckpoint(filepath, monitor='val_acc',
verbose=0,save_best_only=True,mode='max')
callbacks_list = [checkpoint]
reduce_lr = ReduceLROnPlateau(monitor='val_acc', patience=3, mode='auto')
```

1. SimpleRNN

1.1 单层RNN

参数如下:

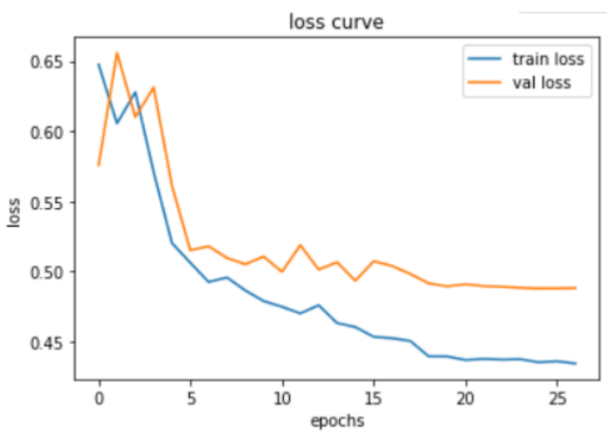
```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(SimpleRNN(128))
model.add(Dropout(0.1))
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集:

```
loss: 0.4373 - acc: 0.8025 - val_loss: 0.4891 - val_acc: 0.7802
```

测试集:

```
loss: 0.4831887483596802
acc: 0.7745599746704102
```



1.2 两层RNN

参数如下：

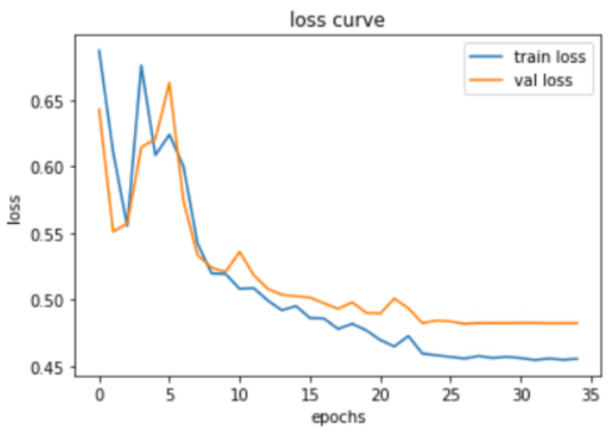
```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(SimpleRNN(128,return_sequences=True))
model.add(Dropout(0.1))
model.add(SimpleRNN(32))
model.add(Dropout(0.1))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集：

```
loss: 0.4571 - acc: 0.7952 - val_loss: 0.4824 - val_acc: 0.7822
```

测试集：

```
loss: 0.48209798336029O5
acc: 0.7752000093460083
```



1.3 BRNN

参数如下：

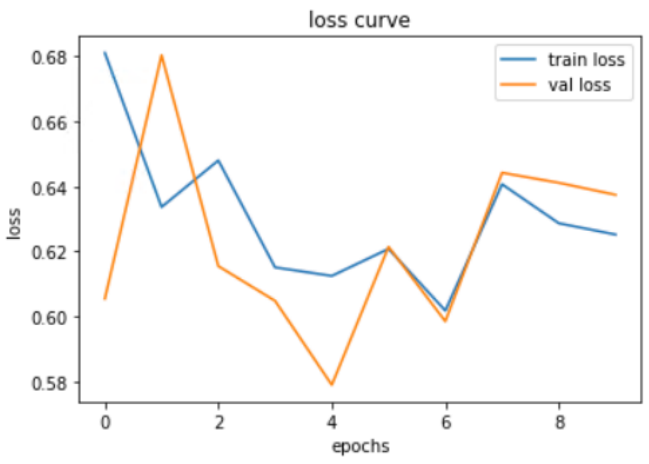
```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(Bidirectional(SimpleRNN(64)))
model.add(Dropout(0.1))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集：

```
loss: 0.6125 - acc: 0.6822 - val_loss: 0.5789 - val_acc: 0.7144
```

测试集：

```
loss: 0.5796838402748108
acc: 0.7117199897766113
```



1.4 DBRNN

参数如下：

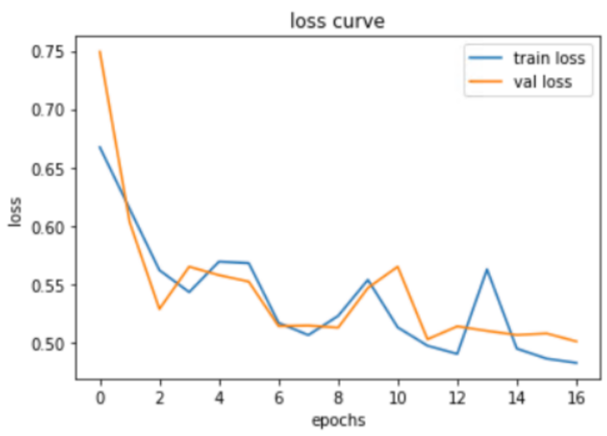
```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(Bidirectional(SimpleRNN(64,return_sequences=True)))
model.add(Dropout(0.1))
model.add(SimpleRNN(64))
model.add(Dropout(0.1))
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集：

```
loss: 0.4972 - acc: 0.7660 - val_loss: 0.5027 - val_acc: 0.7686
```

测试集：

```
loss: 0.500249981880188
acc: 0.7675600051879883
```



2.LSTM

2.1 单层LSTM

参数如下：

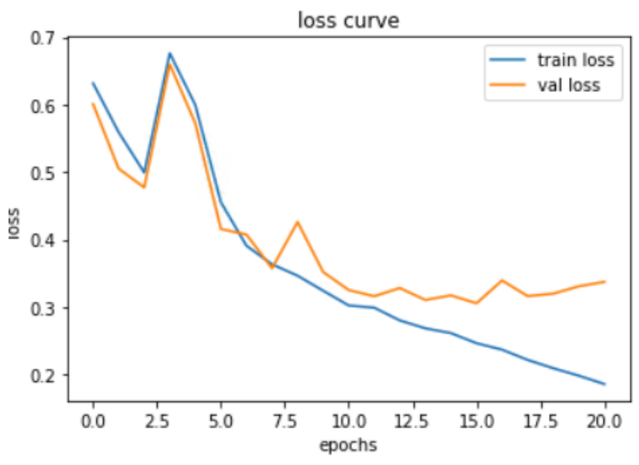
```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(128,activation="relu"))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集：

```
loss: 0.1860 - acc: 0.9272 - val_loss: 0.3374 - val_acc: 0.8742
```

测试集：

```
loss: 0.3022507131099701
acc: 0.8752400279045105
```



2.2 两层LSTM

参数如下：

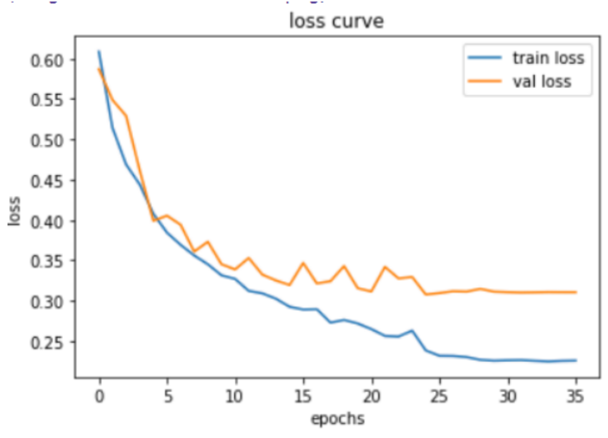
```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(LSTM(32,return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(64))
model.add(Dropout(0.2))
model.add(Dense(128,activation="relu"))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集：

```
loss: 0.2263 - acc: 0.9093 - val_loss: 0.3107 - val_acc: 0.8716
```

测试集：

```
loss: 0.30184149742126465
acc: 0.8762000203132629
```



2.3 BLSTM

参数如下：

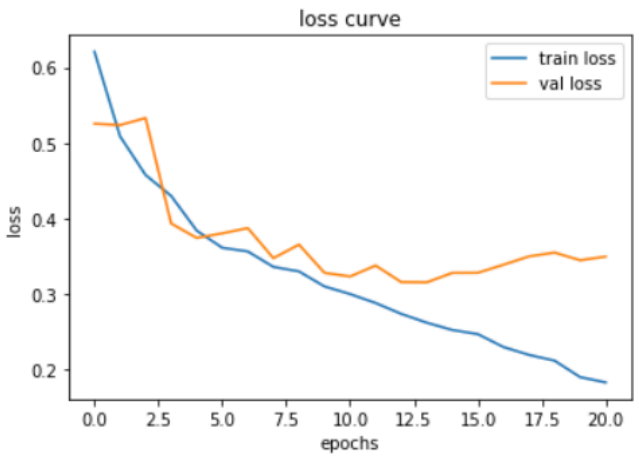
```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.1))
model.add(Dense(128,activation="relu"))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集：

```
loss: 0.2469 - acc: 0.8989 - val_loss: 0.3281 - val_acc: 0.8662
```

测试集：

```
loss: 0.343205988407135
acc: 0.8693600296974182
```



2.4 DBLSTM

参数如下：

```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(Bidirectional(LSTM(32,return_sequences=True)))
model.add(Dropout(0.1))
model.add(LSTM(64))
model.add(Dropout(0.1))
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集:

```
loss: 0.3206 - acc: 0.8652 - val_loss: 0.3275 - val_acc: 0.8592
```

测试集:

```
loss: 0.3261337876319885
acc: 0.8693600296974182
```

3.GRU

3.1 单层GRU

参数如下:

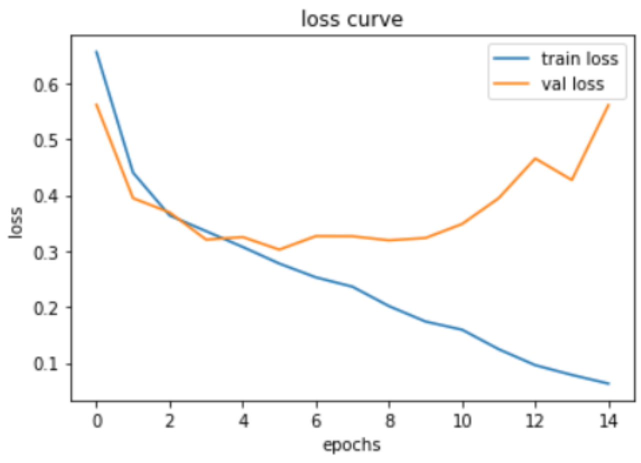
```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(GRU(256))
model.add(Dropout(0.1))
model.add(Dense(64,activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(8,activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集:

```
loss: 0.1740 - acc: 0.9352 - val_loss: 0.3233 - val_acc: 0.8814
```

测试集:

```
loss: 0.3155556321144104
acc: 0.8787199854850769
```



3.2 两层GRU

参数如下：

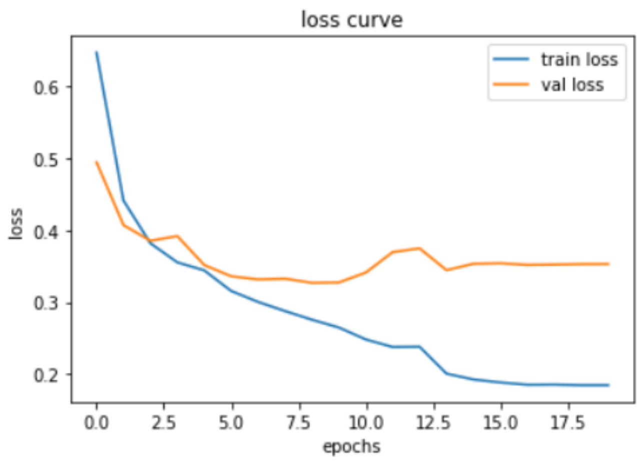
```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(GRU(32,return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(64))
model.add(Dropout(0.2))
model.add(Dense(128,activation="relu"))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集：

```
loss: 0.2848 - acc: 0.8814 - val_loss: 0.3196 - val_acc: 0.8666
```

测试集：

```
loss: 0.30018819849014282
acc: 0.8707200288772583
```



3.3 BGRU

参数如下：

```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(Bidirectional(GRU(64)))
model.add(Dropout(0.1))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集：

```
loss: 0.2646 - acc: 0.8911 - val_loss: 0.3272 - val_acc: 0.8646
```

测试集：

```
loss: 0.32852908968925476
acc: 0.8669999837875366
```

3.4 DBGRU

参数如下：

```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(Bidirectional(GRU(32,return_sequences=True)))
model.add(Dropout(0.1))
model.add(GRU(64))
model.add(Dropout(0.1))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

训练集与验证集：

```
loss: 0.2764 - acc: 0.8856 - val_loss: 0.3157 - val_acc: 0.8686
```

测试集：

```
loss: 0.29279622435569763
acc: 0.8777999877929688
```

4.神经元数量对比

以SimpleRNN为例，对比在验证集上的正确率，基本结构如下：

```
model = Sequential()
model.add(Embedding(max_words,embedding_dim,input_length=maxlen))
model.add(SimpleRNN(x))
model.add(Dropout(0.1))
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

其中x为神经元数量，分别取32、64、128、256来对比：

层数	Loss	正确率
32	0.5759	0.7016
64	0.4911	0.7754
128	0.4891	0.7802
256	0.5452	0.7480

可以看到神经元并不是越多越好，在该模型下最佳个数是128。

六、实验总结

可以看到，LSTM和GRU在测试集上的正确率可以达到接近88%，而SimpleRNN却不能达到80%，说明LSTM和GRU的效果确实要好很多。但是在该数据集上，增加层数、增加双向都不会有明显的改善，反而会增加训练时长，说明训练效果和层数或双向没有一定的关系，需要视具体情况而定。