به نام خدا


درس: یادگیری عمیق

استاد: دکتر فاطمی‌زاده

گزارش تمرین عملی شماره ۶

سیّدمحمّدامین منصوری طهرانی

۹۴۱۰۵۱۷۴

# سوال ۱: Pattern

۱. در ابتدا به توضیح خلاصه قطعه‌های کد می‌پردازیم.

```
from __future__ import print_function
import tensorflow as tf
from tensorflow.contrib import rnn
import numpy as np


def string_gen(k):
    return [a] * k + [N] + [b] * k


def test_string_gen(k):
    return [a] * k + [N]


a = [1.0, 0.0, 0.0, 0.0]
N = [0.0, 1.0, 0.0, 0.0]
b = [0.0, 0.0, 1.0, 0.0]
e = [0.0, 0.0, 0.0, 1.0]
vocabulary = [a, N, b, e]
dictionary = ['a', 'N', 'b', 'e']

# Training Parameters
learning_rate = 0.001
training_steps = 10
display_step = 100
test = 15
training_iter = 200

vocab = sorted(set('aNbe'))

# Network Parameters
num_input = len(vocab)

neuron_num = 10  # Number of LSTM cell neurons.
# char2idx = {u: i for i, u in enumerate(vocab)}
# idx2char = np.array(vocab)
```

در این‌جا فقط دو تابع برای تولید داده‌های تست و آموزش نوشته شده است. کلمات موجود در متن به صورت one-hot در آمده‌اند و توابع خروجی one-hot را برای آن‌ها می‌دهند. در ادامه هم پارامترهای مدل تعریف می‌شود. عدد training step که برابر ۱۰ می‌باشد نشان دهنده این است که در هر بار از iteration ها که تعداد آن‌ها ۲۰۰ بار است، ۱۰ رشته گفته شده در سوال برای آموزش تولید شده و مدل را آموزش می‌دهند. لغت‌نامه (برای حروف) نیز به همراه تعداد نورون‌های خروجی سلول LSTM که باید احتمال کاراکتر بعدی را پیش‌بینی کند تعریف می‌شوند. در آخر نیز تعداد نورون‌های لایه مخفی تعریف شده‌اند.

```
# tf Graph input
X = tf.placeholder("float", [None, None, len(vocab)])
Y = tf.placeholder("float", [None, len(vocab)])

# Define weights
weights = {
    'out': tf.Variable(tf.random_normal([neuron_num, len(vocab)]))
}
biases = {
    'out': tf.Variable(tf.random_normal([len(vocab)]))
}


# noinspection PyPep8Naming,PyShadowingNames
def RNN(x, weights, biases):

    lstm_cell = tf.nn.rnn_cell.LSTMCell(neuron_num, use_peepholes=True, forget_bias=1.0)
    cell_states = lstm_cell.variables

    # Get lstm cell output
    outputs, states = tf.nn.dynamic_rnn(lstm_cell, x, dtype=tf.float32)

    # Linear activation, using rnn inner loop last output
    return tf.matmul(outputs[-1], weights['out']) + biases['out'], states, cell_states


logits, states, cell_state = RNN(X, weights, biases)
prediction = tf.nn.softmax(logits)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)
```

در این قسمت Placeholder ها و وزن و بایاس برای لایه مخفی تعریف شده‌اند. تابعی برای RNN نوشته شده که در آن سلول LSTM ساخته شده و خروجی آن بازگردانده می‌شود. در ادامه نیز خروجی شبکه که از softmax برای تعیین احتمال گذشته‌است، به عنوان پیش‌بینی تعیین می‌شود. تابع هزینه و بهینه‌ساز از نوع Adam و آموزش‌دهنده تعریف می‌شوند.

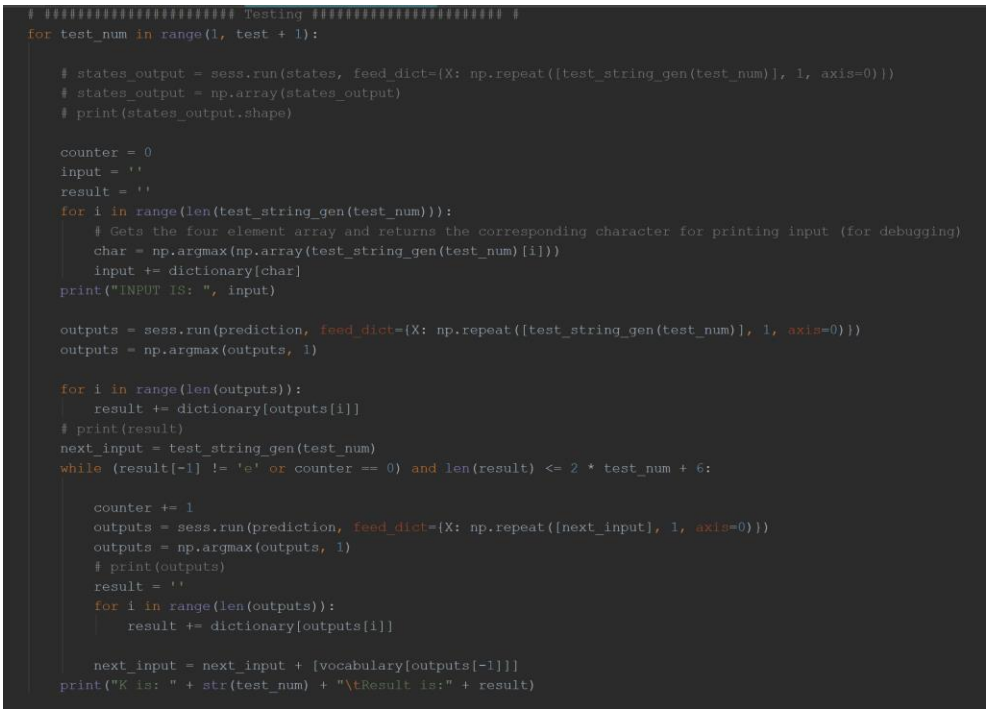
```
# Evaluate model (with test logits, for dropout to be disabled)
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

cell_1 = tf.summary.scalar('cell_states_neuron_1', states[0][0][0])
cell_2 = tf.summary.scalar('cell_states_neuron_2', states[0][0][1])
cell_3 = tf.summary.scalar('cell_states_neuron_3', states[0][0][2])
cell_4 = tf.summary.scalar('cell_states_neuron_4', states[0][0][3])
cell_5 = tf.summary.scalar('cell_states_neuron_5', states[0][0][4])
cell_6 = tf.summary.scalar('cell_states_neuron_6', states[0][0][5])
cell_7 = tf.summary.scalar('cell_states_neuron_7', states[0][0][6])
cell_8 = tf.summary.scalar('cell_states_neuron_8', states[0][0][7])
cell_9 = tf.summary.scalar('cell_states_neuron_9', states[0][0][8])
cell_10 = tf.summary.scalar('cell_states_neuron_10', states[0][0][9])


merge = tf.summary.merge_all()

with tf.Session() as sess:
    # Run the initializer
    sess.run(init)

    file_writer = tf.summary.FileWriter('cell_states_summary')
```

در این قسمت بزرگترین احتمال در بردار one-hot برداشته شده و با برچسب یا همان کاراکتر بعدی که تعیین کرده‌ایم مقایسه می‌شود و نسبت پیش‌بینی‌های صحیح آن به کل تعداد پیش‌بینی‌ها به عنوان دقت محاسبه می‌شود. Summary های تعریف شده برای نگه‌داشتن خروجی اسکالر حالت هر نورون لایه مخفی سلول

LSTM هستند. در ادامه Session شروع می‌شود و فایل‌های summary در فولدر cell_states_summary قرار می‌گیرند.

```
# ###################### Training ###################### #
for iteration in range(training_iter):

    for step in range(1, training_steps + 1):

        # batch_x = np.repeat([string_gen(step)], batch_size, axis=0)
        batch_x = np.repeat([string_gen(step)], 1, axis=0)
        batch_y = string_gen(step)[1:] + [e]
        # print("Batch X is: ", batch_x)
        # print("Batch Y is: ", batch_y)
        # print(type(batch_x))
        # print(type(batch_y))
        # print(len(batch_x))
        # print(len(batch_y))

        # Run optimization op (backprop)
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y})
        if iteration % display_step == 0:
            # Calculate batch loss and accuracy
            loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x,
                                                                 Y: batch_y})
            print("Iteration: " + str(iteration) + "\tStep " + str(step) + ", Minibatch Loss= " + "{:.4f}".format(
                loss) + ", Training Accuracy= " +
                  "{:.3f}".format(acc))
            outputs, Y_res = sess.run((prediction, Y), feed_dict={X: batch_x,
                                                                  Y: batch_y})

            outputs = np.argmax(outputs, 1)
            # print(outputs)
            result = ''
            Y_result = ''
            for i in range(len(outputs)):
                result += dictionary[outputs[i]]
                # Y_result += dictionary[int(Y_res[i])]
            # print(result)
            # print(Y_result)
print("Optimization Finished!")
```

این قسمت به آموزش مربوط است و به تعداد ۲۰۰ بار، هر بار به ترتیب رشته‌های مربوط به k های مختلف به شبکه به همراه خروجی مورد انتظار آن که یک واحد به راست شیفت یافته و e در آخر آن گذاشته شده شده‌است داده می‌شوند و شبکه آموزش می‌بینند. قسمت دوم کد برای دیباگ و مشاهده خروجی پیش‌بینی شده قرار داده شده‌است.



```
# ######################## Testing ######################## #
for test_num in range(1, test + 1):

    # states_output = sess.run(states, feed_dict={X: np.repeat([test_string_gen(test_num)], 1, axis=0)})
    # states_output = np.array(states_output)
    # print(states_output.shape)

    counter = 0
    input = ''
    result = ''
    for i in range(len(test_string_gen(test_num))):
        # Gets the four element array and returns the corresponding character for printing input (for debugging)
        char = np.argmax(np.array(test_string_gen(test_num)[i]))
        input += dictionary[char]
    print("INPUT IS: ", input)

    outputs = sess.run(prediction, feed_dict={X: np.repeat([test_string_gen(test_num)], 1, axis=0)})
    outputs = np.argmax(outputs, 1)

    for i in range(len(outputs)):
        result += dictionary[outputs[i]]
    # print(result)
    next_input = test_string_gen(test_num)
    while (result[-1] != 'e' or counter == 0) and len(result) <= 2 * test_num + 6:

        counter += 1
        outputs = sess.run(prediction, feed_dict={X: np.repeat([next_input], 1, axis=0)})
        outputs = np.argmax(outputs, 1)
        # print(outputs)
        result = ''
        for i in range(len(outputs)):
            result += dictionary[outputs[i]]

        next_input = next_input + [vocabulary[outputs[-1]]]
    print("K is: " + str(test_num) + "\tResult is:" + result)
```

در این قسمت تست برای رشته‌های به طول متفاوت انجام می‌شود. ابتدا برای بررسی و دیباگ کردن ورودی چاپ شده و سپس خروجی حرف بعدی مشخص می‌شود و تا وقتی به پایان نرسیده‌ایم (e) حرف آخر به ادامه ورودی اضافه شده و با این دو حرف بعدی پیش‌بینی می‌شود و این روند ادامه دارد. در پایان حلقه‌ها، خروجی تولید شده چاپ می‌شود.

شرط دوم در حلقه while برای حالت‌هایی بود که درست آموزش ندیده بود و می‌خواستم تعداد غیرمنطقی کاراکتر چاپ نکند. پس از آموزش درست دیگر این مشکل و نیاز به این شرط نبود ولی برای احتیاط آن‌را نگه داشتم.

```
# ###################### Testing for 15 ###################### #

counter = 0
input = ''
result = ''
for i in range(len(test_string_gen(test))):
    char = np.argmax(np.array(test_string_gen(test)[i]))
    input += dictionary[char]
# print(input)
outputs = sess.run(prediction, feed_dict={X: np.repeat([test_string_gen(test)], 1, axis=0)})
outputs = np.argmax(outputs, 1)

for i in range(len(outputs)):
    result += dictionary[outputs[i]]
# print(result)
next_input = test_string_gen(test)
while (result[-1] != 'e' or counter == 0) and len(result) <= 2 * test + 6:

    summary = sess.run(merge, feed_dict={X: np.repeat([next_input], 1, axis=0)})
    file_writer.add_summary(summary, counter)

    counter += 1
    outputs = sess.run(prediction, feed_dict={X: np.repeat([next_input], 1, axis=0)})
    outputs = np.argmax(outputs, 1)
    # print(outputs)
    result = ''
    for i in range(len(outputs)):
        result += dictionary[outputs[i]]

    next_input = next_input + [vocabulary[outputs[-1]]]
print("K is: " + str(test) + "\tResult is:" + result)
```

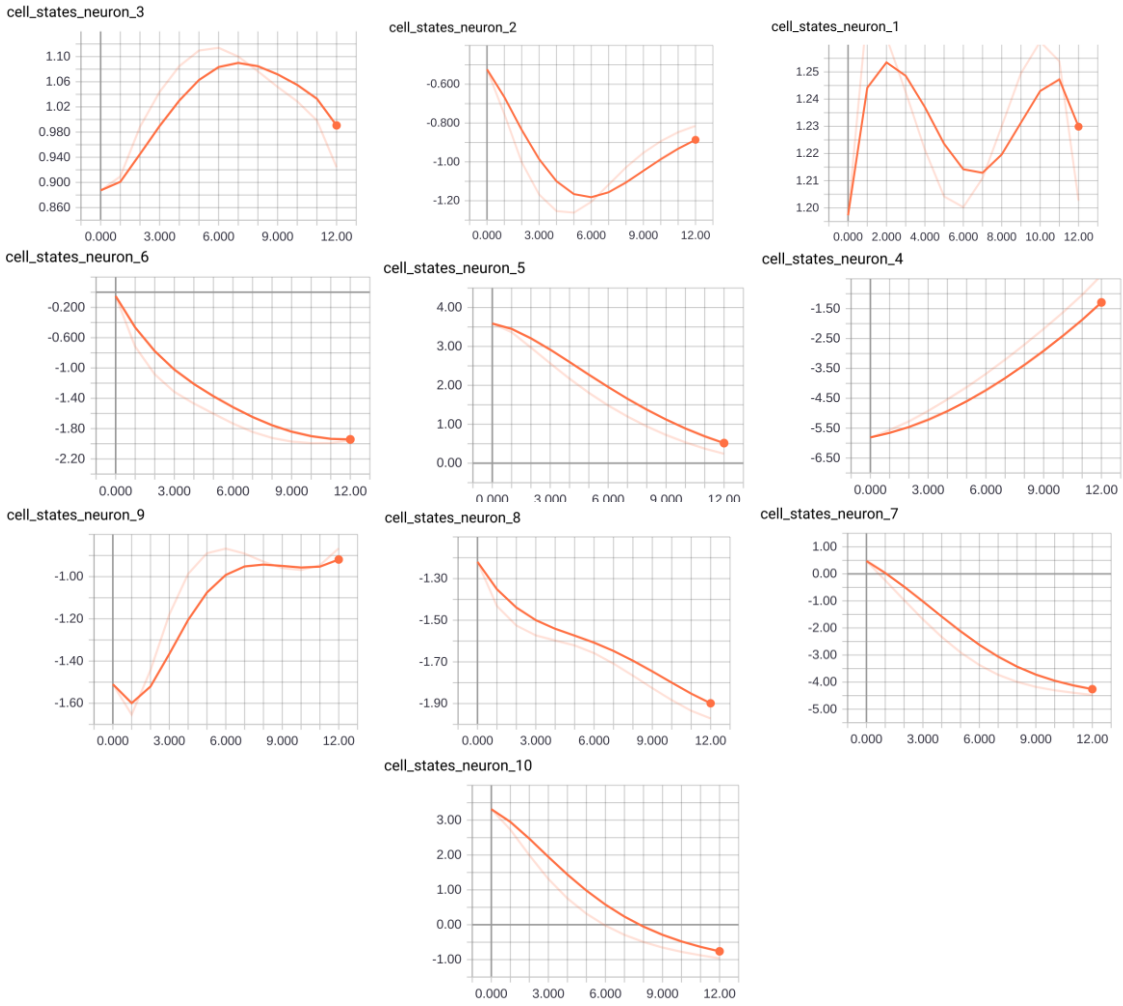
در این قسمت برای دنباله با $K = 15$ تست انجام می‌شود و مقادیر بردار حالت نورون‌های لایه مخفی برای رسم در تنسوربرد ذخیره می‌شوند.

۲. خروجی این مدل آموزش دیده شده در زیر مشاهده می‌شود.

```
aaaaabbbbbe
Iteration: 100  Step 6, Minibatch Loss= 0.2390, Training Accuracy= 0.923
aaaaaabbbbbbe
Iteration: 100  Step 7, Minibatch Loss= 0.2285, Training Accuracy= 0.933
aaaaaaabbbbbbbe
Iteration: 100  Step 8, Minibatch Loss= 0.2294, Training Accuracy= 0.941
aaaaaaaabbbbbbbbe
Iteration: 100  Step 9, Minibatch Loss= 0.2384, Training Accuracy= 0.947
aaaaaaaaabbbbbbbbbe
Iteration: 100  Step 10, Minibatch Loss= 0.2528, Training Accuracy= 0.952
aaaaaaaaaabbbbbbbbbe
Iteration: 200  Step 1, Minibatch Loss= 0.4381, Training Accuracy= 0.667
abe
Iteration: 200  Step 2, Minibatch Loss= 0.4545, Training Accuracy= 0.800
aabbe
Iteration: 200  Step 3, Minibatch Loss= 0.3536, Training Accuracy= 0.857
aaabbbe
Iteration: 200  Step 4, Minibatch Loss= 0.2694, Training Accuracy= 0.889
aaaabbbbe
Iteration: 200  Step 5, Minibatch Loss= 0.2240, Training Accuracy= 0.909
aaaaabbbbbe
Iteration: 200  Step 6, Minibatch Loss= 0.2036, Training Accuracy= 0.923
aaaaaabbbbbbe
Iteration: 200  Step 7, Minibatch Loss= 0.1968, Training Accuracy= 0.933
aaaaaaabbbbbbbe
Iteration: 200  Step 8, Minibatch Loss= 0.1973, Training Accuracy= 0.941
aaaaaaaabbbbbbbbe
Iteration: 200  Step 9, Minibatch Loss= 0.2014, Training Accuracy= 0.947
aaaaaaaaabbbbbbbbbe
Iteration: 200  Step 10, Minibatch Loss= 0.2074, Training Accuracy= 0.952
aaaaaaaaaabbbbbbbbbbe
Optimization Finished!
K is: 1 Result is:abe
K is: 2 Result is:aabbe
K is: 3 Result is:aaabbbe
K is: 4 Result is:aaaabbbbe
K is: 5 Result is:aaaaabbbbbe
K is: 6 Result is:aaaaaabbbbbbe
K is: 7 Result is:aaaaaaabbbbbbbe
K is: 8 Result is:aaaaaaaabbbbbbbbe
K is: 9 Result is:aaaaaaaaabbbbbbbbbe
K is: 10    Result is:aaaaaaaaaabbbbbbbbbe
K is: 11    Result is:aaaaaaaaaaabbbbbbbbbbe
K is: 12    Result is:aaaaaaaaaaaabbbbbbbbbbbe
K is: 13    Result is:aaaaaaaaaaaaNbbbbbbbbbbbbe
K is: 14    Result is:aaaaaaaaaaaaNNbbbbbbbbbbbbbbe
K is: 15    Result is:aaaaaaaaaaaaNNNbbbbbbbbbbbbbbbe
```

همان‌طور که مشاهده می‌شود تست برای $\mathrm{k}$ های بین ۱ تا ۱۵ انجام شده است و به جز چاپ نشدن $\mathrm{N}$، الگو به خوبی تا $k = 12$ تولید شده و در $k = 13$ حتی $\mathrm{N}$ هم چاپ شده ولی بعد از آن خروجی خوبی مطابق انتظار تولید نمی‌شود و علت آن در ادامه روشن می‌شود.

۳. نمودار فعالیت همه ۱۰ نورون در زیر آورده می‌شود:



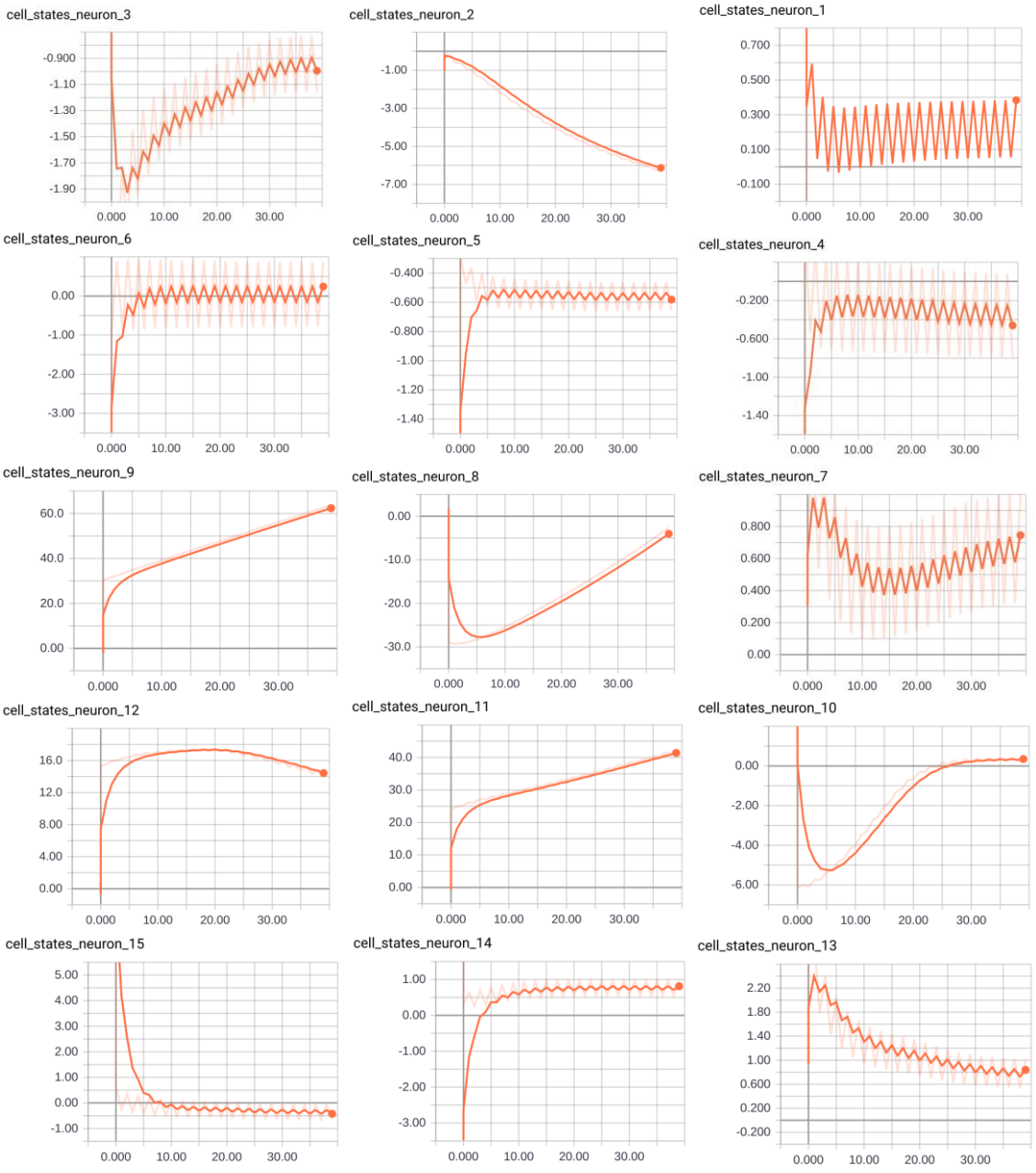

برداشت من از تصاویر فوق این است که هیچ حالت همگرایی در فعالیت اکثر نورون‌ها مشاهده نمی‌شود و همین منشأ ناتوانی در تولید الگو است. در واقع اگر فعالیت همگرا می‌شد احتمالاً الگوی تولید شده با الگوی آموزش دیده‌شده تطابق می‌داشت و حفظ می‌شد. اما نتیجه بدست آمده این است که این همگرایی در مورد الگوهای «تک‌حرفی» بدست نمی‌آید و $\mathrm{LSTM}$ در این مورد ضعیف عمل می‌کند و فقط قابلیت برون‌یابی اندکی دارد. (در مثال ما تا $k = 13$)

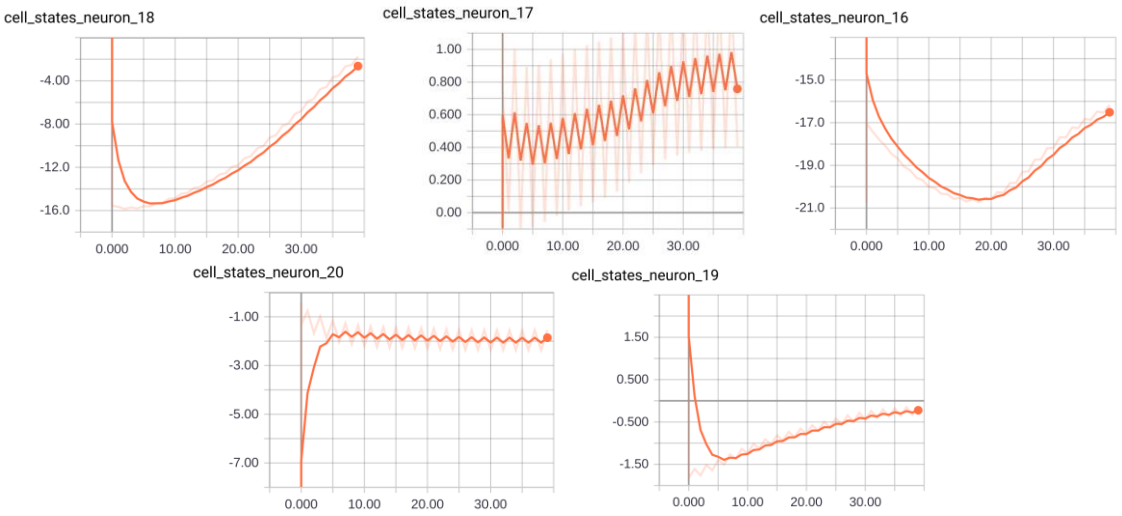
۴. از توضیح کد خودداری می‌کنیم چون شباهت بسیار زیادی با کد قبلی دارد و آوردن نتایج بسنده می‌کنیم.

خروجی مدل پس از تست کردن بر روی رشته‌های تا تعداد تکرار ۲۵ بار برای الگو در شکل زیر مشاهده می‌شود.

```
Iteration: 300  Step 9, Minibatch Loss= 0.0904, Training Accuracy= 0.973
babababababababababababababababae
Iteration: 300  Step 10, Minibatch Loss= 0.0868, Training Accuracy= 0.976
babababababababababababababababae
Iteration: 300  Step 11, Minibatch Loss= 0.0840, Training Accuracy= 0.978
babababababababababababababababae
Iteration: 300  Step 12, Minibatch Loss= 0.0818, Training Accuracy= 0.980
babababababababababababababababae
Iteration: 300  Step 13, Minibatch Loss= 0.0798, Training Accuracy= 0.981
babababababababababababababababae
Iteration: 300  Step 14, Minibatch Loss= 0.0779, Training Accuracy= 0.982
babababababababababababababababae
Iteration: 300  Step 15, Minibatch Loss= 0.0760, Training Accuracy= 0.984
babababababababababababababababae
Optimization Finished!
INPUT IS:  abN
K is: 1 Result is:babae
INPUT IS:  ababN
K is: 2 Result is:babababae
INPUT IS:  abababN
K is: 3 Result is:bababababae
INPUT IS:  ababababN
K is: 4 Result is:babababababababae
INPUT IS:  abababababN
K is: 5 Result is:babababababababababae
INPUT IS:  ababababababN
K is: 6 Result is:babababababababababababae
INPUT IS:  abababababababN
K is: 7 Result is:babababababababababababababae
INPUT IS:  ababababababababN
K is: 8 Result is:babababababababababababababababae
INPUT IS:  abababababababababN
K is: 9 Result is:babababababababababababababababababae
INPUT IS:  ababababababababababN
K is: 10    Result is:babababababababababababababababababae
INPUT IS:  abababababababababababN
K is: 11    Result is:babababababababababababababababababababae
INPUT IS:  ababababababababababababN
K is: 12    Result is:babababababababababababababababababababababae
INPUT IS:  abababababababababababababN
K is: 13    Result is:babababababababababababababababababababababababae
INPUT IS:  ababababababababababababababN
K is: 14    Result is:babababababababababababababababababababababababababae
INPUT IS:  abababababababababababababababN
K is: 15    Result is:babababababababababababababababababababababababababababae
INPUT IS:  ababababababababababababababababN
K is: 16    Result is:babababababababababababababababababababababababababababababae
INPUT IS:  abababababababababababababababababN
K is: 17    Result is:babababababababababababababababababababababababababababababababae
INPUT IS:  ababababababababababababababababababN
K is: 18    Result is:babababababababababababababababababababababababababababababababababae
INPUT IS:  abababababababababababababababababababN
K is: 19    Result is:babababababababababababababababababababababababababababababababababababae
INPUT IS:  ababababababababababababababababababababN
K is: 20    Result is:babababababababababababababababababababababababababababababababababababababae
K is: 20    Result is:babababababababababababababababababababababababababababababababababababababae
```

همان‌طور که مشاهده می‌شود به جز در تولید $N$، الگوها کاملاً با انتظار تطبیق دارند و LSTM به خوبی از پس این عملیات برآمده است. دقت‌ها نیز بسیار بالا هستند. در ادامه با بررسی فعالیت سلول‌ها علت آن را توضیح می‌دهیم. فعالیت هر ۲۰ نورون لایه مخفی سلول LSTM در زیر آورده شده‌است.


cell_states_neuron_3
-0.900
-1.10
-1.30
-1.50
-1.70
-1.90
0.000
10.00
20.00
30.00
cell_states_neuron_2
-1.00
-3.00
-5.00
-7.00
0.000
10.00
20.00
30.00
cell_states_neuron_1
0.700
0.500
0.300
0.100
-0.100
0.000
10.00
20.00
30.00
cell_states_neuron_6
0.00
-1.00
-2.00
-3.00
0.000
10.00
20.00
30.00
cell_states_neuron_5
-0.400
-0.600
-0.800
-1.00
-1.20
-1.40
0.000
10.00
20.00
30.00
cell_states_neuron_4
-0.200
-0.600
-1.00
-1.40
0.000
10.00
20.00
30.00
cell_states_neuron_9
60.0
40.0
20.0
0.00
0.000
10.00
20.00
30.00
cell_states_neuron_8
0.00
-10.0
-20.0
-30.0
0.000
10.00
20.00
30.00
cell_states_neuron_7
0.800
0.600
0.400
0.200
0.00
0.000
10.00
20.00
30.00
cell_states_neuron_12
16.0
12.0
8.00
4.00
0.00
0.000
10.00
20.00
30.00
cell_states_neuron_11
40.0
30.0
20.0
10.0
0.00
0.000
10.00
20.00
30.00
cell_states_neuron_10
0.00
-2.00
-4.00
-6.00
0.000
10.00
20.00
30.00
cell_states_neuron_15
5.00
4.00
3.00
2.00
1.00
0.00
-1.00
0.000
10.00
20.00
30.00
cell_states_neuron_14
1.00
0.00
-1.00
-2.00
-3.00
0.000
10.00
20.00
30.00
cell_states_neuron_13
2.20
1.80
1.40
1.00
0.600
0.200
-0.200
0.000
10.00
20.00
30.00




همان‌طور که در شکل‌های بالا مشاهده می‌شود در اکثر نورون‌ها، فعالیت آن به مقدار ثابتی میل کرده و این را می‌توان این‌گونه تعبیر کرد که نورون‌ها در حال یادگیری الگوی ثابت بیش از «تک‌حرفی» در رشته‌های متوالی شده‌اند و همین باعث شده بتوانند به خوبی دنباله‌های با طول بیشتر از آموزش دیده شده را تولید کنند. پس ادعای قسمت قبلمان هم تقویت می‌شود و یادگیری الگو متناظر با همگرایی فعالیت نورون‌ها به مقدار ثابت است و این همگرایی و یادگیری برای الگوهایی بیش از «تک‌حرفی» موفقیت‌آمیز هستند و ما را به سمت استفاده از کلمات در متن‌های بزرگ به جای حرف‌ها برای یادگیری راهنمایی می‌کنند.

## سوال ۲: Text Generation

کد این قسمت به علت سنگین بودن در jupyter notebook در Google Colab نوشته شده‌است و از آوردن توضیح آن، با توجه به وجود توضیحات در notebook پیوست شده، خودداری می‌شود. فقط screenshot های آن برای تسریع دسترسی به همراه نتیجه آن در زیر آورده می‌شود (با توجه به سخت بودن فارسی تایپ کردن در این نوت‌بوک، کامنت‌ها انگلیسی نوشته شده‌اند!) :

**This piece of code is used to enable using tf.random.categorical**

```
[1]  !pip install -q tf-nightly
```

```
[2]  import tensorflow as tf
     tf.enable_eager_execution()

     import numpy as np
     import os
     import time
```

**First, we read the text data and then creat its corresponding vocabulary containig its unique characters.**

**After that two mappings are introduced to convert each character to its index in the dictionary and vice versa.**

**Last line has the responsibility to convert the whole text to a its index form, i.e. each char is represented by a number in text_as_int.**

```
[3]  text = open('Book.txt').read()
     vocab = sorted(set(text))
     # Creating a mapping from unique characters to indices
     char2idx = {u:i for i, u in enumerate(vocab)}
     idx2char = np.array(vocab)

     text_as_int = np.array([char2idx[c] for c in text])
```

**Here we define 40 character window for each sentence and number of examples per epoch based on that**

```
[4]  # The maximum length sentence we want for a single input in characters
     seq_length = 40
     examples_per_epoch = len(text)//seq_length

     # Create training examples / targets
     char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
```

**The batch method lets us easily convert these individual characters to sequences of the desired size.**

```
[5]  sequences = char_dataset.batch(seq_length+1, drop_remainder=True)
```

**For each sequence, I duplicate and shift it to form the input and target text by using the map to apply a function to each batch:**

```
[6]  def split_input_target(chunk):
         input_text = chunk[:-1]
         target_text = chunk[1:]
         return input_text, target_text

     dataset = sequences.map(split_input_target)
```

**Shuffling and packing data into batches**

```
[7]  # Batch size
     BATCH_SIZE = 64
     steps_per_epoch = examples_per_epoch//BATCH_SIZE

     BUFFER_SIZE = 10000

     dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)
```

**Defining the model using Keras and testing whether there is GPU available (since I am using Google Colab)**

**I use GRU with sigmoid activation function.**

```
[8]  # Length of the vocabulary in chars
     vocab_size = len(vocab)

     # The embedding dimension
     embedding_dim = 256

     # Number of RNN units
     rnn_units = 128

     if tf.test.is_gpu_available():
       rnn = tf.keras.layers.CuDNNGRU
     else:
       import functools
       rnn = functools.partial(
         tf.keras.layers.GRU, recurrent_activation='sigmoid')
```

**tf.keras.layers.Dense: The output layer, with vocab_size outputs.**

```
[9]  def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
       model = tf.keras.Sequential([
         tf.keras.layers.Embedding(vocab_size, embedding_dim,
                                   batch_input_shape=[batch_size, None]),
         rnn(rnn_units,
             return_sequences=True,
             recurrent_initializer='glorot_uniform',
             stateful=True),
         tf.keras.layers.Dense(vocab_size)
       ])
       return model
```

**For each character the model looks up the embedding, runs the GRU one timestep with the embedding as input, and applies the dense layer to generate logits predicting the log-liklihood of the next character**

```
[10] model = build_model(
       vocab_size = len(vocab),
       embedding_dim=embedding_dim,
       rnn_units=rnn_units,
       batch_size=BATCH_SIZE)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/embedding_ops.py:132: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a fut
Instructions for updating:
Colocations handled automatically by placer.
```

```
[11] model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (64, None, 256)           19712
_________________________________________________________________
gru (GRU)                    (64, None, 128)           147840
_________________________________________________________________
dense (Dense)                (64, None, 77)            9933
=================================================================
Total params: 177,485
Trainable params: 177,485
Non-trainable params: 0
_________________________________________________________________
```

## Training

We use the previous RNN state, and the input of this time step to predict the class of the next character.

We also define the loss function.

```
[12] def loss(labels, logits):
       return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)
```

### Configuring the training procedure using the Model.compile method

```
[13] learning_rate = 0.01
     model.compile(optimizer = tf.train.RMSPropOptimizer(learning_rate), loss = loss)
```

### Saving Checkpoints during training.

```
[14] # Directory where the checkpoints will be saved
     checkpoint_dir = './training_checkpoints'
     # Name of the checkpoint files
     checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

     checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(
         filepath=checkpoint_prefix,
         save_weights_only=True)
```

```
[15] EPOCHS=20
```

```
[16] history = model.fit(dataset.repeat(), epochs=EPOCHS, steps_per_epoch=steps_per_epoch, callbacks=[checkpoint_callback])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/data/ops/dataset_ops.py:1730: DatasetV1.make_one_shot_iterator (from tensorflow.python.data.ops.dataset_ops) is deprecate
Instructions for updating:
Use `for ... in dataset:` to iterate over a dataset. If using `tf.estimator`, return the `Dataset` object directly from your input function. As a last resort, you can use `tf.compat.v1.data.make_one_sho
Epoch 1/20
267/267==============================] - 73s 275ms/step - loss: 2.8110
Epoch 2/20
267/267==============================] - 71s 265ms/step - loss: 1.6108
Epoch 3/20
267/267==============================] - 71s 265ms/step - loss: 1.5098
Epoch 4/20
267/267==============================] - 71s 266ms/step - loss: 1.4781
Epoch 5/20
267/267==============================] - 71s 265ms/step - loss: 1.4620
Epoch 6/20
267/267==============================] - 71s 267ms/step - loss: 1.4567
Epoch 7/20
267/267==============================] - 71s 265ms/step - loss: 1.4524
Epoch 8/20
267/267==============================] - 71s 265ms/step - loss: 1.4503
Epoch 9/20
267/267==============================] - 71s 265ms/step - loss: 1.4470
Epoch 10/20
267/267==============================] - 71s 266ms/step - loss: 1.4501
Epoch 11/20
267/267==============================] - 70s 264ms/step - loss: 1.4471
Epoch 12/20
267/267==============================] - 71s 265ms/step - loss: 1.4523
Epoch 13/20
267/267==============================] - 71s 267ms/step - loss: 1.4509
Epoch 14/20
267/267==============================] - 72s 268ms/step - loss: 1.4537
Epoch 15/20
267/267==============================] - 71s 265ms/step - loss: 1.4613
Epoch 16/20
267/267==============================] - 71s 266ms/step - loss: 1.4638
Epoch 17/20
267/267==============================] - 71s 265ms/step - loss: 1.4683
Epoch 18/20
267/267==============================] - 71s 265ms/step - loss: 1.4747
Epoch 19/20
267/267==============================] - 71s 266ms/step - loss: 1.4819
Epoch 20/20
267/267==============================] - 70s 264ms/step - loss: 1.4898
```

## Generating Text

### Restoring the latest checkpoint

**Tensorflow note:**

Because of the way the RNN state is passed from timestep to timestep, the model only accepts a fixed batch size once built.

To run the model with a different batch_size, we need to rebuild the model and restore the weights from the checkpoint.

```
[17] tf.train.latest_checkpoint(checkpoint_dir)
```

```
'./training_checkpoints/ckpt_20'
```

```
[22] model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)

     model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))

     model.build(tf.TensorShape([1, None]))
```

```
[23] model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (1, None, 256)            19712
_________________________________________________________________
gru_2 (GRU)                  (1, None, 128)            147840
_________________________________________________________________
dense_2 (Dense)              (1, None, 77)             9933
=================================================================
Total params: 177,485
Trainable params: 177,485
Non-trainable params: 0
_________________________________________________________________
```

## The prediction loop

**The following code block generates the text:**

**It Starts by choosing a start string, initializing the RNN state and setting the number of characters to generate.**

**Get the prediction distribution of the next character using the start string and the RNN state.**

**Then, use a multinomial distribution to calculate the index of the predicted character. Use this predicted character as our next input to the model.**

**The RNN state returned by the model is fed back into the model so that it now has more context, instead than only one word. After predicting the next word, the modified RNN states are again fed back into the model, which is how it learns as it gets more context from the previously predicted words.**

```
[24] def generate_text(model, start_string):
       # Evaluation step (generating text using the learned model)

       # Number of characters to generate
       num_generate = 400

       # You can change the start string to experiment
       start_string = 'It is a truth univer'

       # Converting our start string to numbers (vectorizing)
       input_eval = [char2idx[s] for s in start_string]
       input_eval = tf.expand_dims(input_eval, 0)

       # Empty string to store our results
       text_generated = []

       # Low temperatures results in more predictable text.
       # Higher temperatures results in more surprising text.
       # Experiment to find the best setting.
       temperature = 1.0

       # Here batch size == 1
       model.reset_states()
       for i in range(num_generate):
           predictions = model(input_eval)
           # remove the batch dimension
           predictions = tf.squeeze(predictions, 0)

           # using a multinomial distribution to predict the word returned by the model
           predictions = predictions / temperature
           predicted_id = tf.multinomial(predictions, num_samples=1)[-1,0].numpy()

           # We pass the predicted word as the next input to the model
           # along with the previous hidden state
           input_eval = tf.expand_dims([predicted_id], 0)

           text_generated.append(idx2char[predicted_id])

       return (start_string + ''.join(text_generated))
```

```
[25] print(generate_text(model, start_string="Why, my dear, you mu"))
```

```
WARNING:tensorflow:From <ipython-input-24-3884b09dee4a>:31: multinomial (from tensorflow.python.ops.random_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.random.categorical instead.
It is a truth univernes of his aspection, well, the same own hon he felt
I could hape-ply reparing him into the leasure.”

Elizabeth of there was. It mintinged. The who length; which and though that
his partia here most an is unnewisture, you will an, eless. We could nature
And the prodeigabld undest,
as she were having to thinkinable monumple, or a compart it
was the now prode cir
this
cothers prompt impromant of Mr
```

دقت شود که برای اجرا شدن فایل notebook باید book.txt به آن ضمیمه شده باشد. هم‌چنین دقت شود که در خروجی کلمات نسبتاً معنا داری ساخته شده‌اند ولی جمله‌ها معنی خاصی ندارند و احتمالاً به epochهای بیشتری برای تولید «جملات» بامعنی نیاز است ولی همین حفظ ساختار متن و پاراگراف‌ها نیز می‌تواند به عنوان موفقیت دیده‌شود!

# سوال ۳: Sentiment Analysis

۱. مانند سوال یک قسمت‌های کد را آورده و به طور خلاصه عملکرد آن‌ها را شرح می‌دهیم.

```
import gensim
from sklearn.model_selection import train_test_split
from numpy import array, append, zeros, reshape
import tensorflow as tf
from tensorflow.contrib import rnn
import numpy as np

model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)

# ################################################ Word Embedding Process ################################################

# Creating a dictionary of phrases and their index

phrase_index_dictionary = {}

with open('dictionary.txt', 'r') as dictionary:
    for line in dictionary:
        phrase, index = line.split('|')
        phrase_index_dictionary[phrase] = int(index)


# Creating a dictionary of phrase indexes and their corresponding score

phraseindex_score_dictionary = {}

with open('sentiment_labels.txt', 'r') as sentiment_labels:
    sentiment_labels.readline()  # skip the first line

    for line in sentiment_labels:
        phraseindex, score = line.split('|')
        phraseindex_score_dictionary[int(phraseindex)] = float(score)
```

در این قسمت ابتدا مدل گوگل به وسیله gensim که برای load کردن مدل‌های گوگل لازم است لود می‌شود. سپس در فایل متنی دیکشنری که عبارت‌ها و اندیس آن‌ها به وسیله | از هم جدا شده‌اند، خوانده شده و عبارت‌ها و اندیس آن‌ها در یک دیکشنری دیگر که در پایتون تعریف شده‌است ذخیره می‌شوند. در ادامه از این‌ها استفاده می‌کنیم.

```
sentences = []
vectorized_sentences = []
sentiment_scores = []

with open('datasetSentences.txt', 'r') as sentences_file:
    sentences_file.readline()  # skip the first line

    for line in sentences_file:

        sentence = line.split('\t')[1]
        sentence = sentence[0:len(sentence) - 1]  # removing \n
        try:
            # phrase_index_dictionary[sentence]  # if the sentence is not available in the dic -> except
            # (this is faster than using if)

            # Extracting words
            sentences.append(sentence.split(' '))
            sentiment_scores.append(phraseindex_score_dictionary[phrase_index_dictionary[sentence]])
            embedded_sentence = []
            for word in sentence:
                try:
                    embedded_sentence.append(model[word])
                    # print('embedded sentence is: ', embedded_sentence)
                except KeyError:
                    pass

            vectorized_sentences.append(np.array(embedded_sentence))

        except KeyError:
            pass

x_train, x_test, y_train, y_test = train_test_split(np.array(vectorized_sentences), np.array(sentiment_scores),
                                                    test_size=0.4)
```

در این قسمت جملات که با فاصله tab جدا شده‌اند، در هر خط بدست می‌آیند و سپس کلمات جمله‌ها جدا شده و عبارت‌های متناظر این‌ها در دیکشنری که در قسمت قبل ساختیم، جستجو شده و مقدار score آن به عنوان score برای کلمات متن، در یک دیکشنری جدید ذخیره می‌شود. در نهایت کلمه‌های هر جمله به صورت بردار شده در کنار هم قرار می‌گیرند. در انتهای این قسمت نیز داده‌های آموزش و تست به همراه برچسب آن‌ها به نسبت ۶۰ و ۴۰ درصد تقسیم می‌شوند.

```
max_sentence_len = 0
x_train_lengths = []
x_test_lengths = []

for x in [x_train, x_test]:
    for sentence in x:
        max_sentence_len = sentence.shape[0] if sentence.shape[0] > max_sentence_len else max_sentence_len

for i in range(len(x_train)):
    x_train_lengths.append(len(x_train[i]))
    while x_train[i].shape[0] < max_sentence_len:
        x_train[i] = append(x_train[i], zeros([1, len(x_train[i][0])]), axis=0)

for i in range(len(x_test)):
    x_test_lengths.append(len(x_test[i]))
    while len(x_test[i]) < max_sentence_len:
        x_test[i] = append(x_test[i], zeros([1, len(x_test[i][0])]), axis=0)
```

از آن‌جایی که طول batch ها متفاوت است برای آموزش آن‌ها باید از zero padding استفاده کرد و در این‌جا اول ماکزیمم طول یک جمله بدست آمده و سپس داده‌های آموزش و تست برحسب آن zero pad می‌شوند.

## قسمت شبکه عمیق مسأله:

در قطعه زیر، دو تابع اول از پستی در اینترنت کپی شدند که برای استفاده در زمانی است که طول ورودی متغیر بوده و ما از zero padding استفاده کرده‌ایم و باید با ظرافت خاصی خروجی آخر RNN را به predictor دادا! هدف این دو تابع همین است! (لینک کد توابع: variable-sequence-lengths-in-tensorflow)

در ادامه پارامترهای مدل و placeholder ها و وزن و بایاس تعریف شده‌اند.

```
# ################################################## NETWORK ##################################################


def length(sequence):
    used = tf.sign(tf.reduce_max(tf.abs(sequence), 2))
    length_ = tf.reduce_sum(used, 1)
    length_ = tf.cast(length_, tf.int32)
    return length_


def last_relevant(output, length_):
    batch_size_ = tf.shape(output)[0]
    max_length = tf.shape(output)[1]
    out_size = int(output.get_shape()[2])
    index_f = tf.range(0, batch_size_) * max_length + (length_ - 1)
    flat = tf.reshape(output, [-1, out_size])
    relevant = tf.gather(flat, index_f)
    return relevant


learning_rate = 0.01
hidden_neuron_num = 256
display_step = 100
vocab_size = len(x_train[0][0])
train_epochs = 5
batch_size = 64

# tf Graph input
x = tf.placeholder("float", [None, None, vocab_size])
y = tf.placeholder("float", [None, 1])

# Define weights
weights = tf.Variable(tf.random_normal([hidden_neuron_num, 1]))
biases = tf.Variable(tf.random_normal([1]))
```

```
lstm_cell = tf.nn.rnn_cell.LSTMCell(hidden_neuron_num)

# Get lstm cell output
outputs, states = tf.nn.dynamic_rnn(lstm_cell, x, dtype=tf.float32, sequence_length=length(x))

# prediction = tf.matmul(outputs[:, -1, :], weights) + biases
rnn_prediction = tf.matmul(last_relevant(outputs, length(x)), weights) + biases

cost = tf.losses.mean_squared_error(labels=y, predictions=rnn_prediction)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)


init = tf.global_variables_initializer()
```

در این قسمت سلول lstm تعریف شده و خروجی آن با همان ظرافتی که اشاره شد (length(x)) گرفته شده و با عبور از لایه مخفی برای محاسبه تابع هزینه به losses.mean_squared_error داده می‌شود. بهینه‌ساز و initializer نیز تعریف شده‌اند.

```
with tf.Session() as sess:
    sess.run(init)

    # ############################################ Training ############################################ #

    for epoch in range(train_epochs):
        for i in range(0, len(x_train), batch_size):
            x_batch = array(x_train[i: i + batch_size])
            y_batch = array(y_train[i: i + batch_size])

            reshaped_x_batch = array([x_train[i]])
            for j in range(i + 1, min(i + batch_size, len(x_train))):
                reshaped_x_batch = append(reshaped_x_batch, array([x_train[j]]), axis=0)

            reshaped_y_batch = reshape(y_batch, newshape=[
                len(y_batch), 1
            ])

            _, loss, prediction = sess.run([optimizer, cost, rnn_prediction],
                                           feed_dict={x: reshaped_x_batch,
                                                      y: reshaped_y_batch})
            if i % 20 == 0:
                print('Epoch Number:', epoch, 'Batch:', i)
                print('loss:', loss)
```

در این قسمت آموزش انجام می‌شود و در قطعه کد بعدی تست انجام می‌شود و نکته خاصی به جز تغییر در شکل آن‌ها برای مطابقت با feed_dict وجود ندارد. تنها موضوع قابل اشاره این‌که عدد ۰/۰۵ برای حد خطای MSE

نتیجه‌ی دقت تست بسیار بدی داشت (زیر ۵ درصد) و حدس ما این بود که احتمالاً این عدد کم انتخاب شده و افزایش آن به ۰/۱ دقت به طرز چشم‌گیری افزایش یافت:

```
    # ########################################## Testing ########################################## #

counter = 0
for i in range(0, len(x_test), batch_size):
    x_batch = array(x_test[i: i + batch_size])
    y_batch = array(y_test[i: i + batch_size])

    reshaped_x_batch = array([x_test[i]])
    for j in range(i + 1, min(i + batch_size, len(x_test))):
        reshaped_x_batch = append(reshaped_x_batch, array([x_test[j]]), axis=0)

    reshaped_y_batch = reshape(y_batch, newshape=[
        len(y_batch), 1
    ])

    loss, prediction = sess.run([cost, rnn_prediction],
                                feed_dict={x: reshaped_x_batch,
                                           y: reshaped_y_batch})

    if loss < 0.1:
        counter += 1

print('Test Accuracy: ', counter / len(range(0, len(x_test), batch_size)))
```

```
loss: 0.049623713
epoch: 4 i: 320
loss: 0.06424567
epoch: 4 i: 640
loss: 0.06453371
eenshot i: 960
loss: 0.08323996
epoch: 4 i: 1280
loss: 0.080185965
epoch: 4 i: 1600
loss: 0.069540165
epoch: 4 i: 1920
loss: 0.06499288
epoch: 4 i: 2240
loss: 0.07291743
epoch: 4 i: 2560
loss: 0.06674148
epoch: 4 i: 2880
loss: 0.059124507
epoch: 4 i: 3200
loss: 0.05753001
epoch: 4 i: 3520
loss: 0.05734839
epoch: 4 i: 3840
loss: 0.062694676
epoch: 4 i: 4160
loss: 0.062360052
epoch: 4 i: 4480
loss: 0.053660415
epoch: 4 i: 4800
loss: 0.07276732
epoch: 4 i: 5120
loss: 0.07828818
epoch: 4 i: 5440
loss: 0.07445374
epoch: 4 i: 5760
loss: 0.065325364
epoch: 4 i: 6080
loss: 0.07346146
epoch: 4 i: 6400
loss: 0.0757631
epoch: 4 i: 6720
loss: 0.07248192
test accuracy:  0.971830985915493
```

مشاهده می‌شود که دقت پیش‌بینی برچسب sentiment ها بسیار خوب است و شبکه موفقیت‌آمیز عمل کرده‌است.