

With TF 1.0!



h_2



x_2

Lab 12

RNN

h_t



x_t

h_{t+1}



x_{t+1}

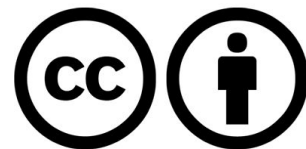
h_{t+2}



x_{t+2}

Sung Kim <hunkim+ml@gmail.com>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



Call for comments

Please feel free to add comments directly on these slides

Other slides: <https://goo.gl/jPtVNT>



With TF 1.0!

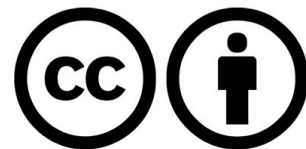


Lab 12-1





















RNN Basics

Sung Kim <hunkim+ml@gmail.com>

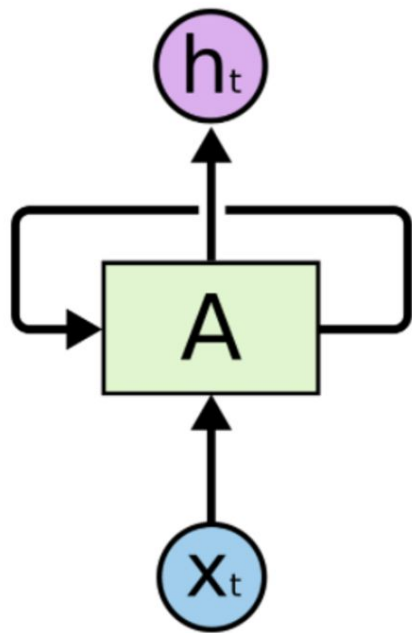
Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



<https://github.com/hunkim/DeepLearningZeroToAll/>

 sxscience 1 commit / 350 ++ / 0 --	#12	 jennykang 19 commits / 940 ++ / 253 --	#2	 GzuPark 15 commits / 49 ++ / 39 --	#3	 kkweon 12 commits / 1,087 ++ / 340 --	#4
 BlueMelon715 5 commits / 55 ++ / 44 --	#5	 jihobak 3 commits / 244 ++ / 1,289 --	#6	 FuZer 2 commits / 37 ++ / 30 --	#7	 jin-chong 2 commits / 4 ++ / 4 --	#8
 zeran4 1 commit / 5 ++ / 4 --	#9	 cynthia 1 commit / 28 ++ / 28 --	#10	 kaka120011 1 commit / 1 ++ / 1 --	#11	 keon 1 commit / 3 ++ / 3 --	#12
 allieus 1 commit / 55 ++ / 59 --	#13	 togheppi 1 commit / 1,280 ++ / 0 --	#14	 davinnovation 1 commit / 64 ++ / 0 --	#15	 skyer9 1 commit / 94 ++ / 69 --	#16
 redongjun 1 commit / 78 ++ / 0 --	#17	 bkRyusim 1 commit / 1 ++ / 1 --	#18	 maestrojeong 1 commit / 50 ++ / 0 --	#19	 mjc92 1 commit / 1,215 ++ / 0 --	#13

RNN in TensorFlow

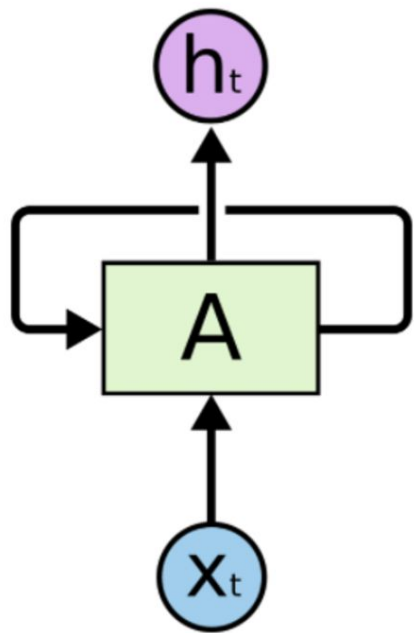


```
cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)
```

```
...
```

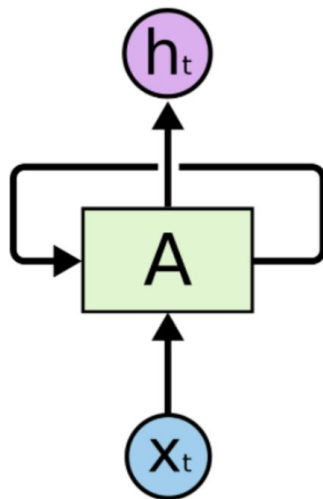
```
outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
```

RNN in TensorFlow



```
cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)  
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size)  
...  
outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
```

One node: 4 (*input-dim*) in 2 (*hidden_size*)



One hot encoding

$h = [1, 0, 0, 0]$

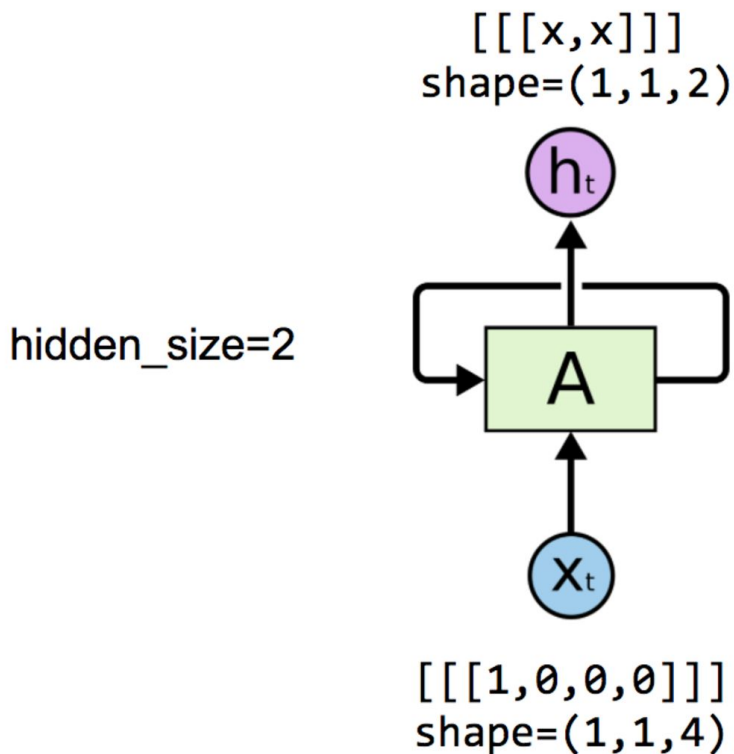
$e = [0, 1, 0, 0]$

$l = [0, 0, 1, 0]$

$o = [0, 0, 0, 1]$

$[[[1, 0, 0, 0]]]$
 $\text{shape}=(1, 1, 4)$

One node: 4 (*input-dim*) in 2 (*hidden_size*)



One hot encoding
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]

One node: 4 (*input-dim*) in 2 (*hidden_size*)

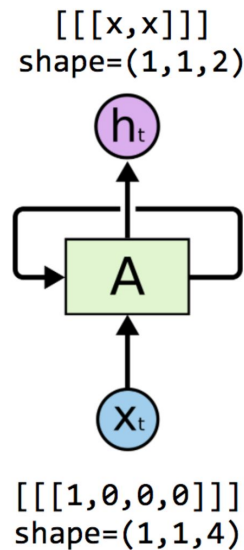
```
# One cell RNN input_dim (4) -> output_dim (2)
hidden_size = 2
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size)

x_data = np.array([[[[1,0,0,0]]], dtype=np.float32)
outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)

sess.run(tf.global_variables_initializer())
pp.pprint(outputs.eval())
```

hidden_size=2

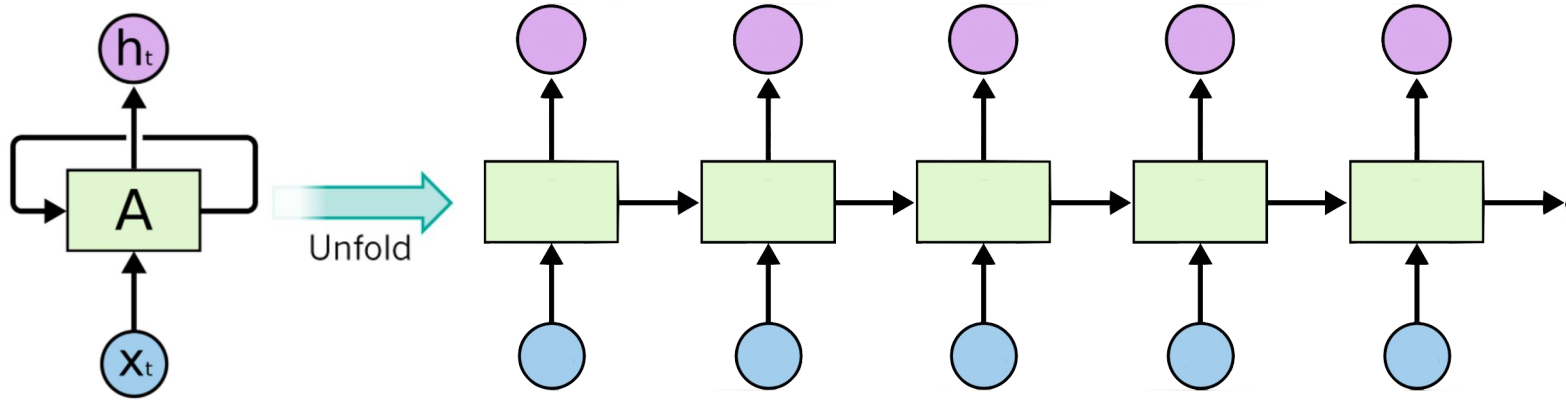
```
array([[[[-0.42409304,  0.64651132]]]])
```



Unfolding to n sequences

Hidden_size=2
sequence_length=5

shape=(1,5,2): $[[[x,x], [x,x], [x,x], [x,x], [x,x]]]$



shape=(1,5,4): $[[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]$

h e l l o

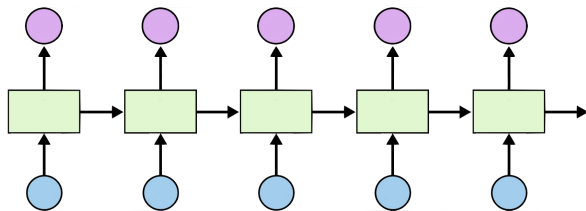
Unfolding to n sequences

One cell RNN input_dim (4) -> output_dim (2). sequence: 5

```
hidden_size = 2
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size)
x_data = np.array([[h, e, 1, 1, o]], dtype=np.float32)
print(x_data.shape)
pp.pprint(x_data)
outputs, states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
sess.run(tf.global_variables_initializer())
pp.pprint(outputs.eval())
```

Hidden_size=2
sequence_length=5

shape=(1,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]]]



shape=(1,5,4): [[[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]]
h e 1 1 o

One hot encoding

h = [1, 0, 0, 0]

e = [0, 1, 0, 0]

l = [0, 0, 1, 0]

o = [0, 0, 0, 1]

X_data = array

```
([[[ 1., 0., 0., 0.],
 [ 0., 1., 0., 0.],
 [ 0., 0., 1., 0.],
 [ 0., 0., 1., 0.],
 [ 0., 0., 0., 1.]]], dtype=float32)
```

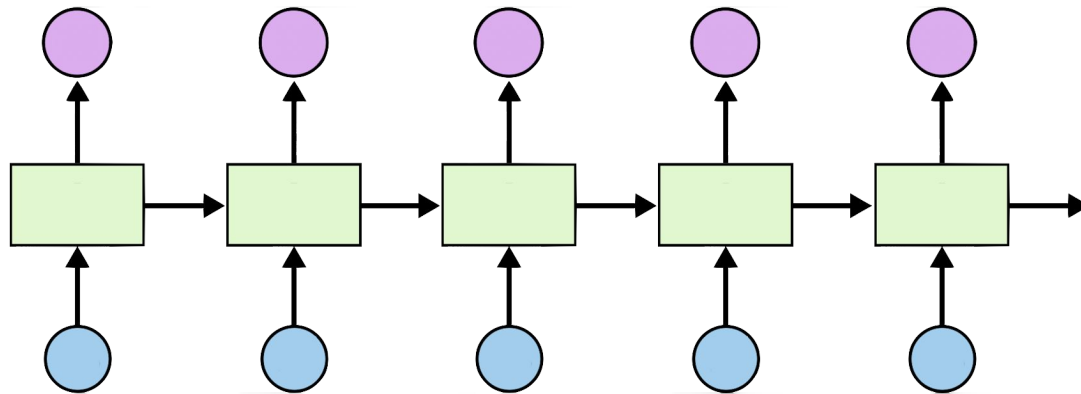
Outputs = array

```
([[[ 0.19709368, 0.24918222],
 [-0.11721198, 0.1784237 ],
 [-0.35297349, -0.66278851],
 [-0.70915914, -0.58334434],
 [-0.38886023, 0.47304463]]], dtype=float32)
```

Hidden_size=2
sequence_length=5
batch_size=3

Batching input

shape=(3,5,2): $\begin{bmatrix} [x,x] & [x,x] & [x,x] & [x,x] & [x,x] \\ [x,x] & [x,x] & [x,x] & [x,x] & [x,x] \\ [x,x] & [x,x] & [x,x] & [x,x] & [x,x] \end{bmatrix}$



shape=(3,5,4): $\begin{bmatrix} [1,0,0,0] & [0,1,0,0] & [0,0,1,0] & [0,0,1,0] & [0,0,0,1] \\ [0,1,0,0] & [0,0,0,1] & [0,0,1,0] & [0,0,1,0] & [0,0,1,0] \\ [0,0,1,0] & [0,0,1,0] & [0,1,0,0] & [0,1,0,0] & [0,0,1,0] \end{bmatrix}$ # hello
eolll
lleeel

Batching input

One cell RNN input_dim (4) -> output_dim (2). sequence: 5, batch 3

3 batches 'hello', 'eolll', 'lleel'

```
x_data = np.array([[h, e, l, l, o],  
                  [e, o, l, l, l],  
                  [l, l, e, e, l]], dtype=np.float32)
```

```
pp.pprint(x_data)
```

```
cell = rnn.BasicLSTMCell(num_units=2, state_is_tuple=True)
```

```
outputs, _states = tf.nn.dynamic_rnn(cell, x_data,  
                                     dtype=tf.float32)
```

```
sess.run(tf.global_variables_initializer())
```

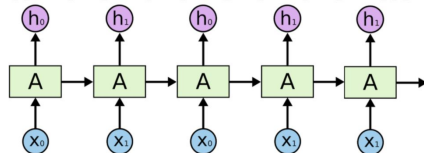
```
pp.pprint(outputs.eval())
```

```
array([[[[ 1.,  0.,  0.,  0.],  
         [ 0.,  1.,  0.,  0.],  
         [ 0.,  0.,  1.,  0.],  
         [ 0.,  0.,  1.,  0.],  
         [ 0.,  0.,  0.,  1.]],
```

```
[[[ 0.,  1.,  0.,  0.],  
   [ 0.,  0.,  0.,  1.],  
   [ 0.,  0.,  1.,  0.],  
   [ 0.,  0.,  1.,  0.],  
   [ 0.,  0.,  1.,  0.]],
```

```
[[[ 0.,  0.,  1.,  0.],  
   [ 0.,  0.,  1.,  0.],  
   [ 0.,  1.,  0.,  0.],  
   [ 0.,  1.,  0.,  0.],  
   [ 0.,  0.,  1.,  0.]]],
```

```
shape=(3,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]],  
                [[x,x], [x,x], [x,x], [x,x], [x,x]],  
                [[x,x], [x,x], [x,x], [x,x], [x,x]]]
```



Hidden_size=2
sequence_length=5
batch_size=3

```
shape=(3,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]], # hello  
                [[0,1,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,0], [0,0,1,0]], # eolll  
                [[0,0,1,0], [0,0,1,0], [0,1,0,0], [0,1,0,0], [0,0,1,0]]] # lleel
```

Batching input

One cell RNN input_dim (4) -> output_dim (2). sequence: 5, batch 3
 # 3 batches 'hello', 'eolll', 'lleel'

```
x_data = np.array([[h, e, l, l, o],
                  [e, o, l, l, l],
                  [l, l, e, e, l]], dtype=np.float32)
```

```
pp.pprint(x_data)
```

```
cell = rnn.BasicLSTMCell(num_units=2, state_is_tuple=True)
outputs, _states = tf.nn.dynamic_rnn(cell, x_data,
                                     dtype=tf.float32)
```

```
sess.run(tf.global_variables_initializer())
pp.pprint(outputs.eval())
```

```
array([[[[ 1.,  0.,  0.,  0.],
          [ 0.,  1.,  0.,  0.],
          [ 0.,  0.,  1.,  0.],
          [ 0.,  0.,  1.,  0.],
          [ 0.,  0.,  0.,  1.]],
```

```
[[ 0.,  1.,  0.,  0.],
 [ 0.,  0.,  0.,  1.],
 [ 0.,  0.,  1.,  0.],
 [ 0.,  0.,  1.,  0.],
 [ 0.,  0.,  1.,  0.],
```

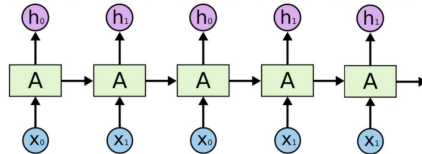
```
[[ 0.,  0.,  1.,  0.],
 [ 0.,  0.,  1.,  0.],
 [ 0.,  1.,  0.,  0.],
 [ 0.,  1.,  0.,  0.],
 [ 0.,  0.,  1.,  0.]],
```

```
array([[[[-0.0173022 , -0.12929453],
          [-0.14995177, -0.23189341],
          [ 0.03294011,  0.01962204],
          [ 0.12852104,  0.12375218],
          [ 0.13597946,  0.31746736]],
```

```
[[[-0.15243632, -0.14177315],
 [ 0.04586344,  0.12249056],
 [ 0.14292534,  0.15872268],
 [ 0.18998367,  0.21004884],
 [ 0.21788891,  0.24151592]],
```

```
[[ 0.10713603,  0.11001928],
 [ 0.17076059,  0.1799853 ],
 [-0.03531617,  0.08993293],
 [-0.1881337 , -0.08296411],
 [-0.00404597,  0.07156041]]],
```

```
shape=(3,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]],
                [[x,x], [x,x], [x,x], [x,x], [x,x]],
                [[x,x], [x,x], [x,x], [x,x], [x,x]]]
```



Hidden_size=2
sequence_length=5
batch_size=3

```
shape=(3,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]], # hello
                [[0,1,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,0], [0,0,1,0]], # eolll
                [[0,0,1,0], [0,0,1,0], [0,1,0,0], [0,1,0,0], [0,0,1,0]]] # lleel
```

With TF 1.0!

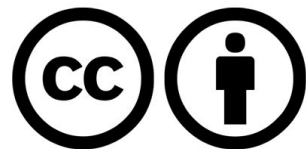


Lab 12-2




















Hi Hello RNN

Sung Kim <hunkim+ml@gmail.com>

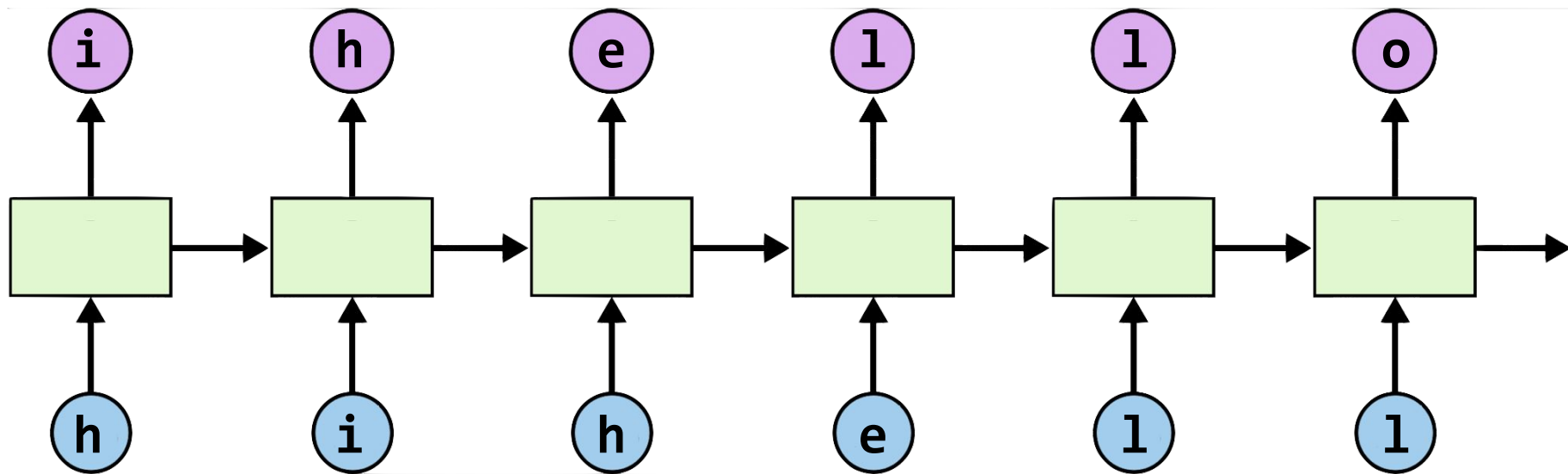
Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



<https://github.com/hunkim/DeepLearningZeroToAll/>

 sxscience 1 commit / 350 ++ / 0 --	#12	 jennykang 19 commits / 940 ++ / 253 --	#2	 GzuPark 15 commits / 49 ++ / 39 --	#3	 kkweon 12 commits / 1,087 ++ / 340 --	#4
 BlueMelon715 5 commits / 55 ++ / 44 --	#5	 jihobak 3 commits / 244 ++ / 1,289 --	#6	 FuZer 2 commits / 37 ++ / 30 --	#7	 jin-chong 2 commits / 4 ++ / 4 --	#8
 zeran4 1 commit / 5 ++ / 4 --	#9	 cynthia 1 commit / 28 ++ / 28 --	#10	 kaka120011 1 commit / 1 ++ / 1 --	#11	 keon 1 commit / 3 ++ / 3 --	#12
 allieus 1 commit / 55 ++ / 59 --	#13	 togheppi 1 commit / 1,280 ++ / 0 --	#14	 davinnovation 1 commit / 64 ++ / 0 --	#15	 skyer9 1 commit / 94 ++ / 69 --	#16
 redongjun 1 commit / 78 ++ / 0 --	#17	 bkRyusim 1 commit / 1 ++ / 1 --	#18	 maestrojeong 1 commit / 50 ++ / 0 --	#19	 mjc92 1 commit / 1,215 ++ / 0 --	#13

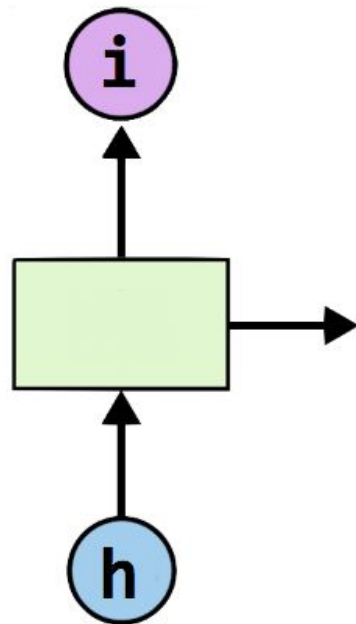
Teach RNN 'hihello'



- text: 'hihello'
- unique chars (vocabulary, voc):
h, i, e, l, o
- voc index:
h:0, i:1, e:2, l:3, o:4

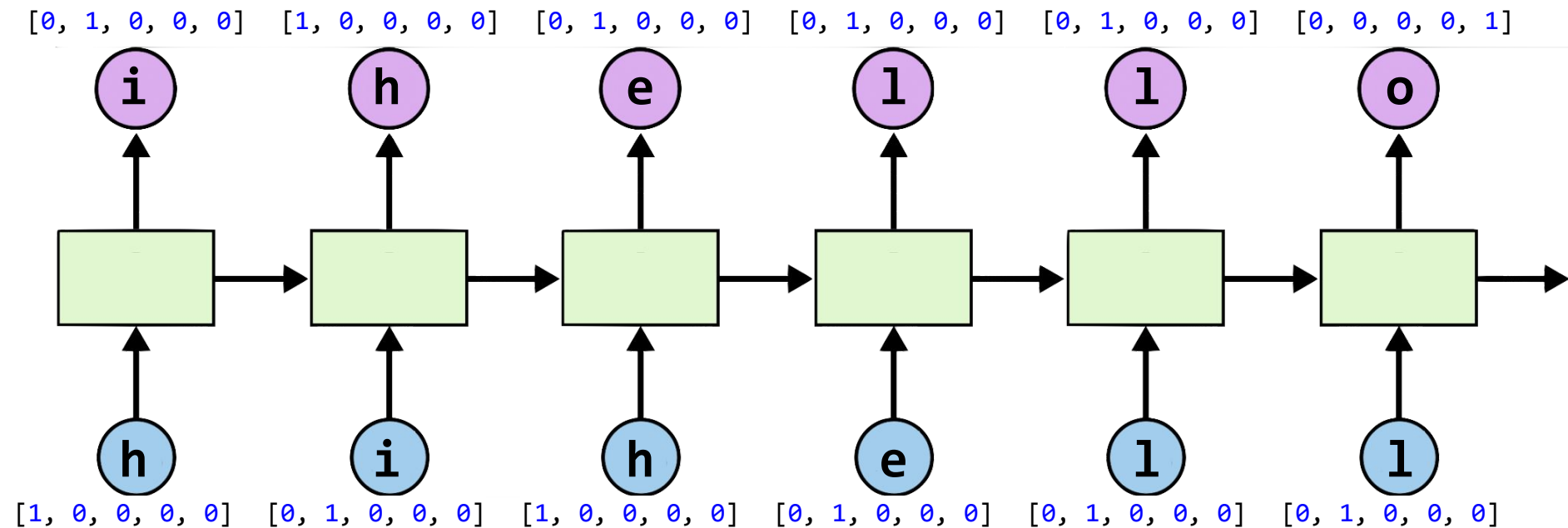
One-hot encoding

[1, 0, 0, 0, 0],	# h 0
[0, 1, 0, 0, 0],	# i 1
[0, 0, 1, 0, 0],	# e 2
[0, 0, 0, 1, 0],	# l 3
[0, 0, 0, 0, 1],	# o 4



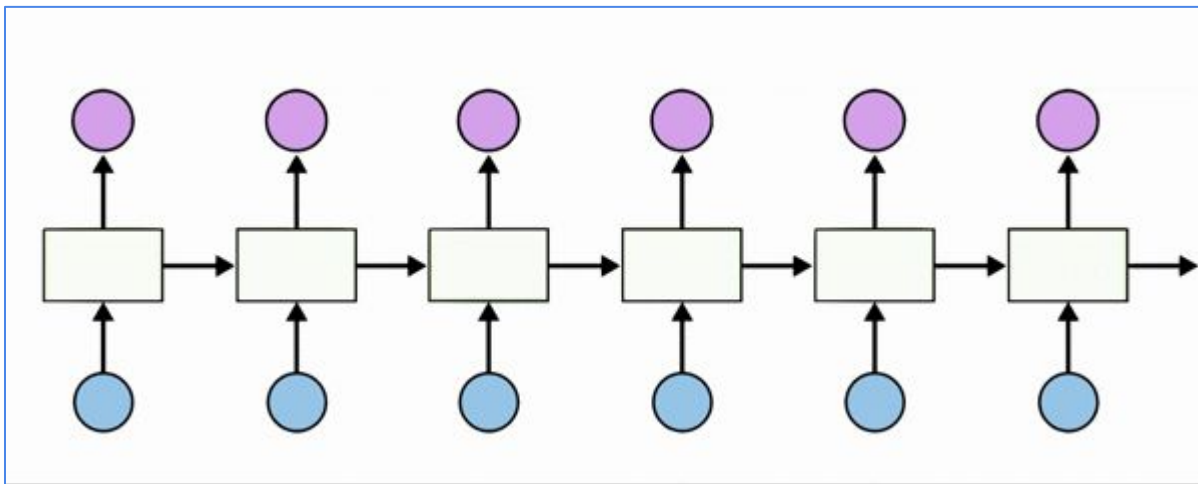
Teach RNN 'hihello'

$[1, 0, 0, 0, 0]$,	# h 0
$[0, 1, 0, 0, 0]$,	# i 1
$[0, 0, 1, 0, 0]$,	# e 2
$[0, 0, 0, 1, 0]$,	# l 3
$[0, 0, 0, 0, 1]$,	# o 4



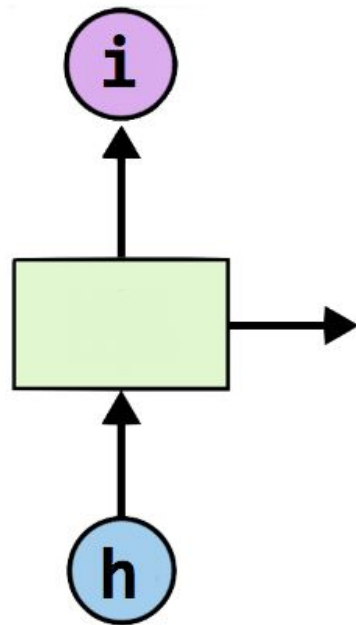
Teach RNN 'hihello'

$[1, 0, 0, 0, 0]$,	# h 0
$[0, 1, 0, 0, 0]$,	# i 1
$[0, 0, 1, 0, 0]$,	# e 2
$[0, 0, 0, 1, 0]$,	# l 3
$[0, 0, 0, 0, 1]$,	# o 4



Creating rnn cell

```
# RNN model  
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)
```



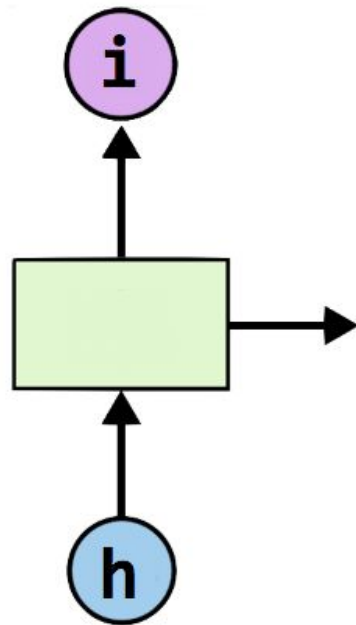
Creating rnn cell

RNN model

```
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)
```

```
rnn_cell = rnn_cell. BasicLSTMCell(rnn_size)
```

```
rnn_cell = rnn_cell. GRUCell(rnn_size)
```



Execute RNN

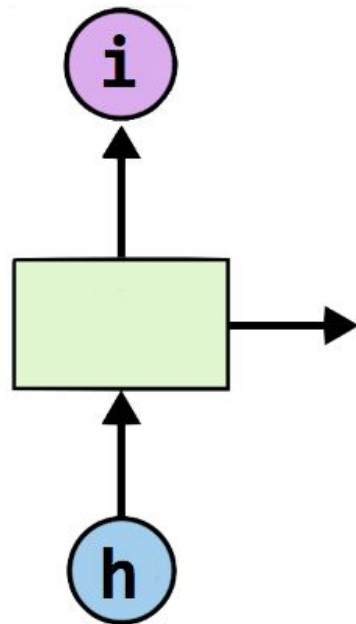
```
# RNN model
```

```
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)
```

```
outputs, _states = tf.nn.dynamic_rnn(  
    rnn_cell,
```

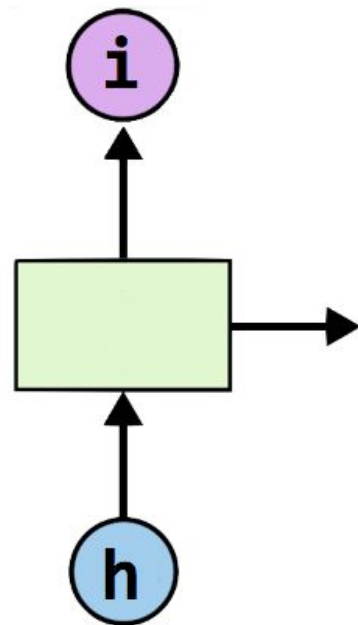
```
    2 x,  
    initial_state=initial_state,  
    dtype=tf.float32)
```

hidden_rnn_size



RNN parameters

```
hidden_size = 5      # output from the LSTM  
input_dim = 5        # one-hot size  
batch_size = 1       # one sentence  
sequence_length = 6  # |ihello| == 6
```



Data creation

```
idx2char = ['h', 'i', 'e', 'l', 'o'] # h=0, i=1, e=2, l=3, o=4
x_data = [[0, 1, 0, 2, 3, 3]] # hihell
x_one_hot = [[[1, 0, 0, 0, 0], # h 0
               [0, 1, 0, 0, 0], # i 1
               [1, 0, 0, 0, 0], # h 0
               [0, 0, 1, 0, 0], # e 2
               [0, 0, 0, 1, 0], # l 3
               [0, 0, 0, 1, 0]]] # l 3

y_data = [[1, 0, 2, 3, 3, 4]] # ihello
X = tf.placeholder(tf.float32,
                   [None, sequence_length, input_dim]) # X one-hot
Y = tf.placeholder(tf.int32, [None, sequence_length]) # Y label
```

Feed to RNN

```
x_one_hot = [[[1, 0, 0, 0, 0], # h 0
               [0, 1, 0, 0, 0], # i 1
               [1, 0, 0, 0, 0], # h 0
               [0, 0, 1, 0, 0], # e 2
               [0, 0, 0, 1, 0], # l 3
               [0, 0, 0, 1, 0]]] # l 3

y_data = [[1, 0, 2, 3, 3, 4]] # i hello
```

```
X = tf.placeholder(
    tf.float32, [None, sequence_length, hidden_size]) # X one-hot
Y = tf.placeholder(tf.int32, [None, sequence_length]) # Y label
```

```
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size,
    state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, _states = tf.nn.dynamic_rnn(
    cell, X, initial_state=initial_state, dtype=tf.float32)
```

Cost: sequence_loss

```
# [batch_size, sequence_length]
y_data = tf.constant([[1, 1, 1]])

# [batch_size, sequence_length, emb_dim]
prediction = tf.constant([[[0.2, 0.7], [0.6, 0.2], [0.2, 0.9]]], dtype=tf.float32)

# [batch_size * sequence_length]
weights = tf.constant([[1, 1, 1]], dtype=tf.float32)

sequence_loss = tf.contrib.seq2seq.sequence_loss(logits=prediction, targets=y_data, weights=weights)
sess.run(tf.global_variables_initializer())
print("Loss: ", sequence_loss.eval())
```

Loss: 0.596759

Cost: sequence_loss

```
# [batch_size, sequence_length]
```

```
y_data = tf.constant([[1, 1, 1]])
```

```
# [batch_size, sequence_length, emb_dim ]
```

```
prediction1 = tf.constant([[[0.3, 0.7], [0.3, 0.7], [0.3, 0.7]]],  
dtype=tf.float32)
```

```
prediction2 = tf.constant([[[0.1, 0.9], [0.1, 0.9], [0.1, 0.9]]],  
dtype=tf.float32)
```

```
# [batch_size * sequence_length]
```

```
weights = tf.constant([[1, 1, 1]], dtype=tf.float32)
```

```
sequence_loss1 = tf.contrib.seq2seq.sequence_loss(prediction1, y_data,  
weights)
```

```
sequence_loss2 = tf.contrib.seq2seq.sequence_loss(prediction2, y_data,  
weights)
```

```
sess.run(tf.global_variables_initializer())
```

```
print("Loss1: ", sequence_loss1.eval(),  
      "Loss2: ", sequence_loss2.eval())
```

Loss1: 0.513015

Loss2: 0.371101

Cost: sequence_loss

```
outputs, _states = tf.nn.dynamic_rnn(  
    cell, X, initial_state=initial_state, dtype=tf.float32)  
weights = tf.ones([batch_size, sequence_length])  
  
sequence_loss = tf.contrib.seq2seq.sequence_loss(  
    logits=outputs, targets=Y, weights=weights)  
loss = tf.reduce_mean(sequence_loss)  
train = tf.train.AdamOptimizer(learning_rate=0.1).minimize(loss)
```

Training

```
prediction = tf.argmax(outputs, axis=2)
```

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(2000):
        l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
        result = sess.run(prediction, feed_dict={X: x_one_hot})
        print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

    # print char using dic
    result_str = [idx2char[c] for c in np.squeeze(result)]
    print("\tPrediction str: ", ''.join(result_str))
```

```
prediction = tf.argmax(outputs, axis=2)
```

Results

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(2000):
        l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
        result = sess.run(prediction, feed_dict={X: x_one_hot})
        print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

# print char using dic
result_str = [idx2char[c] for c in np.squeeze(result)]
print("\tPrediction str: ", ''.join(result_str))
```

```
0 loss: 1.55474 prediction:  [[3 3 3 3 4 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  lllllo
1 loss: 1.55081 prediction:  [[3 3 3 3 4 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  lllllo
2 loss: 1.54704 prediction:  [[3 3 3 3 4 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  lllllo
3 loss: 1.54342 prediction:  [[3 3 3 3 4 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  lllllo
...
1998 loss: 0.75305 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  ihello
1999 loss: 0.752973 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]] Prediction str:  ihello
```

With TF 1.0!

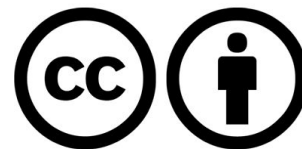


Lab 12-3





















RNN with long sequences

Sung Kim <hunkim+ml@gmail.com>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



<https://github.com/hunkim/DeepLearningZeroToAll/>

 sxscience 1 commit / 350 ++ / 0 --	#12	 jennykang 19 commits / 940 ++ / 253 --	#2	 GzuPark 15 commits / 49 ++ / 39 --	#3	 kkweon 12 commits / 1,087 ++ / 340 --	#4
 BlueMelon715 5 commits / 55 ++ / 44 --	#5	 jihobak 3 commits / 244 ++ / 1,289 --	#6	 FuZer 2 commits / 37 ++ / 30 --	#7	 jin-chong 2 commits / 4 ++ / 4 --	#8
 zeran4 1 commit / 5 ++ / 4 --	#9	 cynthia 1 commit / 28 ++ / 28 --	#10	 kaka120011 1 commit / 1 ++ / 1 --	#11	 keon 1 commit / 3 ++ / 3 --	#12
 allieus 1 commit / 55 ++ / 59 --	#13	 togheppi 1 commit / 1,280 ++ / 0 --	#14	 davinnovation 1 commit / 64 ++ / 0 --	#15	 skyer9 1 commit / 94 ++ / 69 --	#16
 redongjun 1 commit / 78 ++ / 0 --	#17	 bkRyusim 1 commit / 1 ++ / 1 --	#18	 maestrojeong 1 commit / 50 ++ / 0 --	#19	 mjc92 1 commit / 1,215 ++ / 0 --	#13

Manual data creation

```
idx2char = ['h', 'i', 'e', 'l', 'o']
x_data = [[0, 1, 0, 2, 3, 3]]      # hihell
x_one_hot = [[[1, 0, 0, 0, 0],      # h 0
               [0, 1, 0, 0, 0],      # i 1
               [1, 0, 0, 0, 0],      # h 0
               [0, 0, 1, 0, 0],      # e 2
               [0, 0, 0, 1, 0],      # l 3
               [0, 0, 0, 1, 0]]]     # l 3
y_data = [[1, 0, 2, 3, 3, 4]]      # ihello
```

Better data creation

```
sample = " if you want you"
idx2char = list(set(sample)) # index -> char
char2idx = {c: i for i, c in enumerate(idx2char)} # char -> idx

sample_idx = [char2idx[c] for c in sample] # char to index
x_data = [sample_idx[:-1]] # X data sample (0 ~ n-1) hello: hell
y_data = [sample_idx[1:]] # Y label sample (1 ~ n) hello: ello

X = tf.placeholder(tf.int32, [None, sequence_length]) # X data
Y = tf.placeholder(tf.int32, [None, sequence_length]) # Y label

X_one_hot = tf.one_hot(X, num_classes) # one hot: 1 -> 0 1 0 0 0 0 0 0 0 0
```

Hyper parameters

```
sample = " if you want you"
idx2char = list(set(sample)) # index -> char
char2idx = {c: i for i, c in enumerate(idx2char)} # char -> idx

# hyper parameters
dic_size = len(char2idx) # RNN input size (one hot size)
rnn_hidden_size = len(char2idx) # RNN output size
num_classes = len(char2idx) # final output size (RNN or softmax, etc.)
batch_size = 1 # one sample data, one batch
sequence_length = len(sample) - 1 # number of lstm unfolding (unit #)
```

LSTM and Loss

```
X = tf.placeholder(tf.int32, [None, sequence_length]) # X data
Y = tf.placeholder(tf.int32, [None, sequence_length]) # Y label
```

```
X_one_hot = tf.one_hot(X, num_classes) # one hot: 1 -> 0 1 0 0 0 0 0 0 0 0
```

```
cell = tf.contrib.rnn.BasicLSTMCell(num_units=rnn_hidden_size, state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, _states = tf.nn.dynamic_rnn(
    cell, X_one_hot, initial_state=initial_state, dtype=tf.float32)
```

```
weights = tf.ones([batch_size, sequence_length])
sequence_loss = tf.contrib.seq2seq.sequence_loss(logits=outputs, targets=Y, weights=weights)
loss = tf.reduce_mean(sequence_loss)
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(loss)
```

```
prediction = tf.argmax(outputs, axis=2)
```

Training and Results

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(3000):
        l, _ = sess.run([loss, train], feed_dict={X: x_data, Y: y_data})
        result = sess.run(prediction, feed_dict={X: x_data})
        # print char using dic
        result_str = [idx2char[c] for c in np.squeeze(result)]
        print(i, "loss:", l, "Prediction:", ''.join(result_str))
```

0 loss: 2.29895 Prediction: nnuffuunnuuyuy

1 loss: 2.29675 Prediction: nnuffuunnuuyuy

...

1418 loss: 1.37351 Prediction: if you want you

1419 loss: 1.37331 Prediction: if you want you

Really long sentence?

```
sentence = ("if you want to build a ship, don't drum up people together to "  
            "collect wood and don't assign them tasks and work, but rather "  
            "teach them to long for the endless immensity of the sea.")
```

Really long sentence?

sentence = ("if you want to build a ship, don't drum up people together to "
"collect wood and don't assign them tasks and work, but rather "
"teach them to long for the endless immensity of the sea.")

training dataset

0 if you wan -> f you want

1 f you want -> you want

2 you want -> you want t

3 you want t -> ou want to

...

168 of the se -> of the sea

169 of the sea -> f the sea.


```
char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}
```

```
dataX = []
dataY = []
```

```
for i in range(0, len(sentence) - seq_length):
    x_str = sentence[i:i + seq_length]
    y_str = sentence[i + 1: i + seq_length + 1]
    print(i, x_str, '->', y_str)
```

```
x = [char_dic[c] for c in x_str] # x str to index
y = [char_dic[c] for c in y_str] # y str to index
```

```
dataX.append(x)
dataY.append(y)
```

Making dataset

training dataset

0 if you wan -> f you want

1 f you want -> you want

2 you want -> you want t

3 you want t -> ou want to

...

168 of the se -> of the sea

169 of the sea -> f the sea.

RNN parameters

```
char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}

data_dim = len(char_set)
hidden_size = len(char_set)
num_classes = len(char_set)
seq_length = 10 # Any arbitrary number

batch_size = len(dataX)
```

training dataset

0 if you wan -> f you want

1 f you want -> you want

2 you want -> you want t

3 you want t -> ou want to

...

168 of the se -> of the sea

169 of the sea -> f the sea.

LSTM and Loss

```
X = tf.placeholder(tf.int32, [None, sequence_length]) # X data
Y = tf.placeholder(tf.int32, [None, sequence_length]) # Y label
```

```
X_one_hot = tf.one_hot(X, num_classes) # one hot: 1 -> 0 1 0 0 0 0 0 0 0 0
```

```
cell = tf.contrib.rnn.BasicLSTMCell(num_units=rnn_hidden_size, state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, _states = tf.nn.dynamic_rnn(
    cell, X_one_hot, initial_state=initial_state, dtype=tf.float32)
```

```
weights = tf.ones([batch_size, sequence_length])
sequence_loss = tf.contrib.seq2seq.sequence_loss(logits=outputs, targets=Y, weights=weights)
loss = tf.reduce_mean(sequence_loss)
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(loss)
```

```
prediction = tf.argmax(outputs, axis=2)
```

Exercise

- Run long sequence RNN
- Why it does not work?

With TF 1.0!

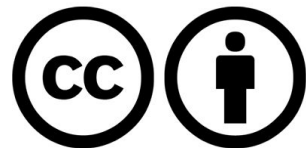


Lab 12-4




















RNN with long sequences: Stacked
RNN + Softmax layer

Sung Kim <hunkim+ml@gmail.com>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



<https://github.com/hunkim/DeepLearningZeroToAll/>

 sxscience 1 commit / 350 ++ / 0 --	#12	 jennykang 19 commits / 940 ++ / 253 --	#2	 GzuPark 15 commits / 49 ++ / 39 --	#3	 kkweon 12 commits / 1,087 ++ / 340 --	#4
 BlueMelon715 5 commits / 55 ++ / 44 --	#5	 jihobak 3 commits / 244 ++ / 1,289 --	#6	 FuZer 2 commits / 37 ++ / 30 --	#7	 jin-chong 2 commits / 4 ++ / 4 --	#8
 zeran4 1 commit / 5 ++ / 4 --	#9	 cynthia 1 commit / 28 ++ / 28 --	#10	 kaka120011 1 commit / 1 ++ / 1 --	#11	 keon 1 commit / 3 ++ / 3 --	#12
 allieus 1 commit / 55 ++ / 59 --	#13	 togheppi 1 commit / 1,280 ++ / 0 --	#14	 davinnovation 1 commit / 64 ++ / 0 --	#15	 skyer9 1 commit / 94 ++ / 69 --	#16
 redongjun 1 commit / 78 ++ / 0 --	#17	 bkRyusim 1 commit / 1 ++ / 1 --	#18	 maestrojeong 1 commit / 50 ++ / 0 --	#19	 mjc92 1 commit / 1,215 ++ / 0 --	#13

Really long sentence?

```
sentence = ("if you want to build a ship, don't drum up people together to "  
            "collect wood and don't assign them tasks and work, but rather "  
            "teach them to long for the endless immensity of the sea.")
```

```
char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}
```

```
dataX = []
dataY = []
```

```
for i in range(0, len(sentence) - seq_length):
    x_str = sentence[i:i + seq_length]
    y_str = sentence[i + 1: i + seq_length + 1]
    print(i, x_str, '->', y_str)
```

```
x = [char_dic[c] for c in x_str] # x str to index
y = [char_dic[c] for c in y_str] # y str to index
```

```
dataX.append(x)
dataY.append(y)
```

Making dataset

training dataset

0 if you wan -> f you want

1 f you want -> you want

2 you want -> you want t

3 you want t -> ou want to

...

168 of the se -> of the sea

169 of the sea -> f the sea.

RNN parameters

```
char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}

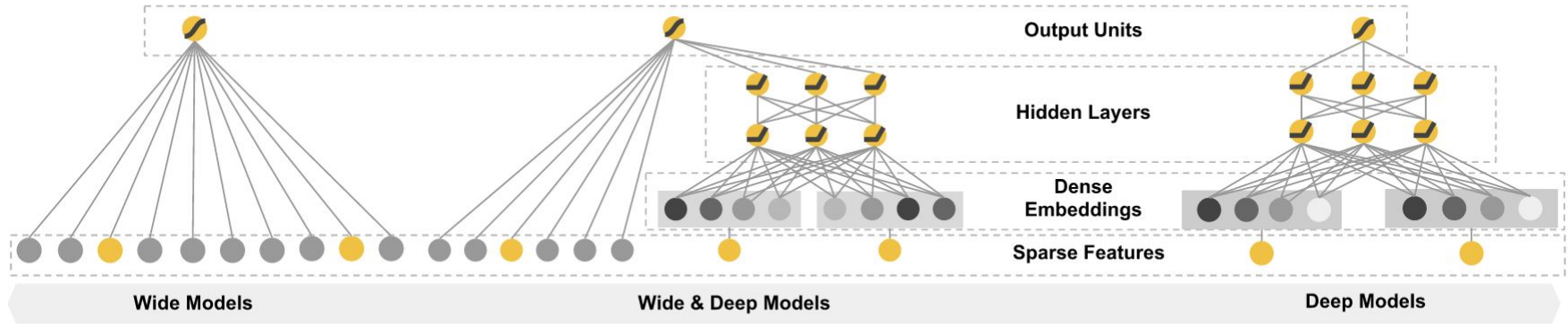
data_dim = len(char_set)
hidden_size = len(char_set)
num_classes = len(char_set)
seq_length = 10 # Any arbitrary number

batch_size = len(dataX)
```

training dataset

0 if you wan -> f you want
1 f you want -> you want
2 you want -> you want t
3 you want t -> ou want to
...
168 of the se -> of the sea
169 of the sea -> f the sea.

Wide & Deep



Stacked RNN

```
X = tf.placeholder(tf.int32, [None, seq_length])  
Y = tf.placeholder(tf.int32, [None, seq_length])
```

```
# One-hot encoding
```

```
X_one_hot = tf.one_hot(X, num_classes)
```

```
print(X_one_hot) # check out the shape
```

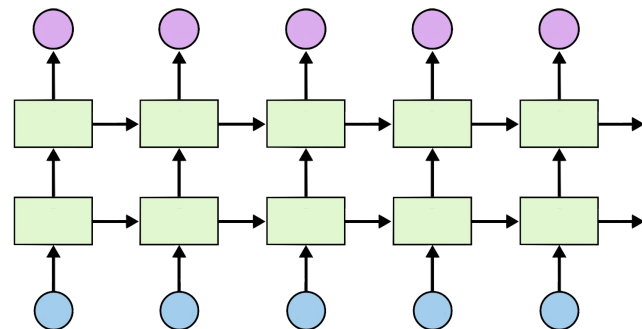
```
# Make a lstm cell with hidden_size (each unit output vector size)
```

```
cell = rnn.BasicLSTMCell(hidden_size, state_is_tuple=True)
```

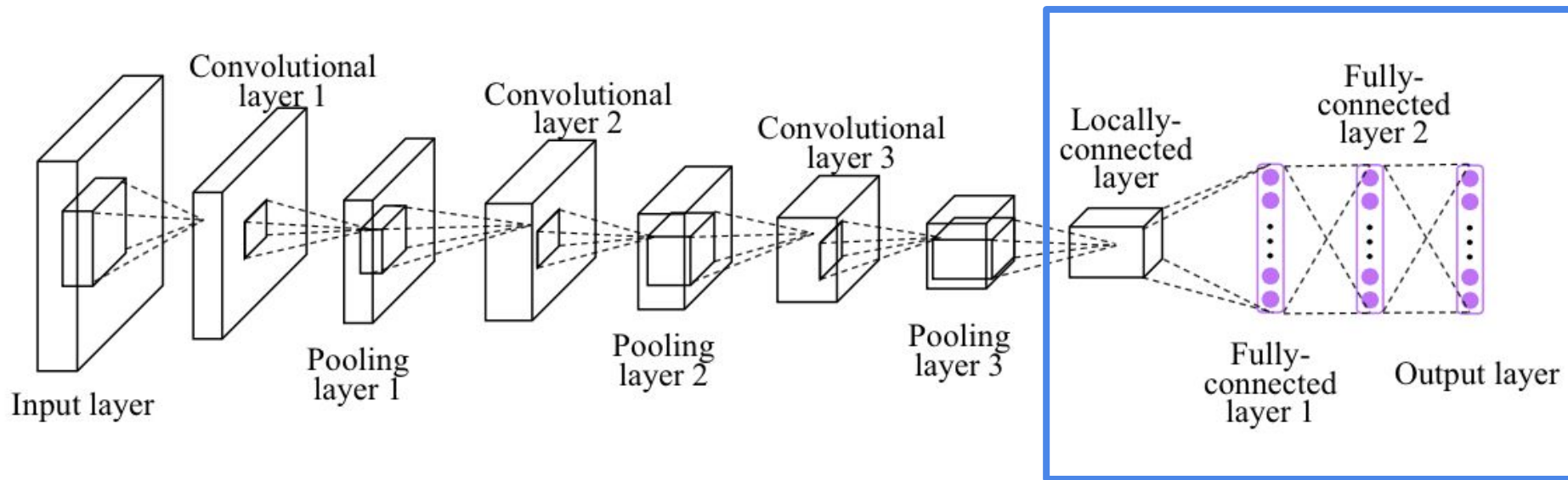
```
cell = rnn.MultiRNNCell([cell] * 2, state_is_tuple=True)
```

```
# outputs: unfolding size x hidden size, state = hidden size
```

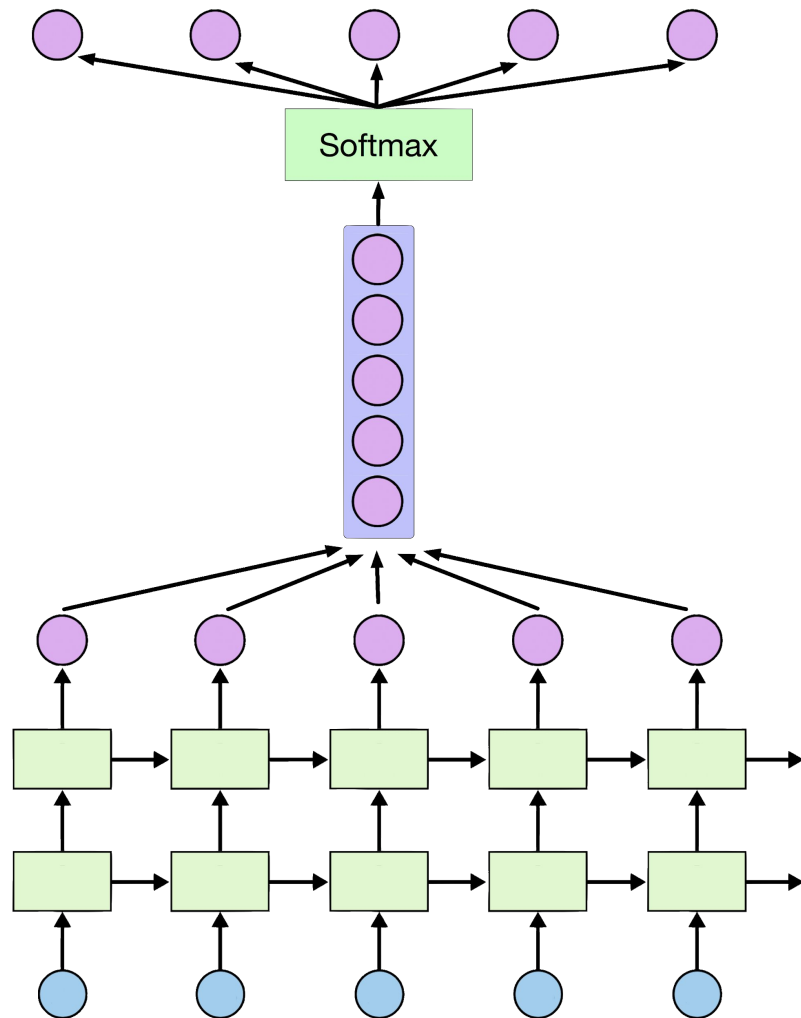
```
outputs, _states = tf.nn.dynamic_rnn(cell, X_one_hot, dtype=tf.float32)
```



Softmax (FC) in Deep CNN



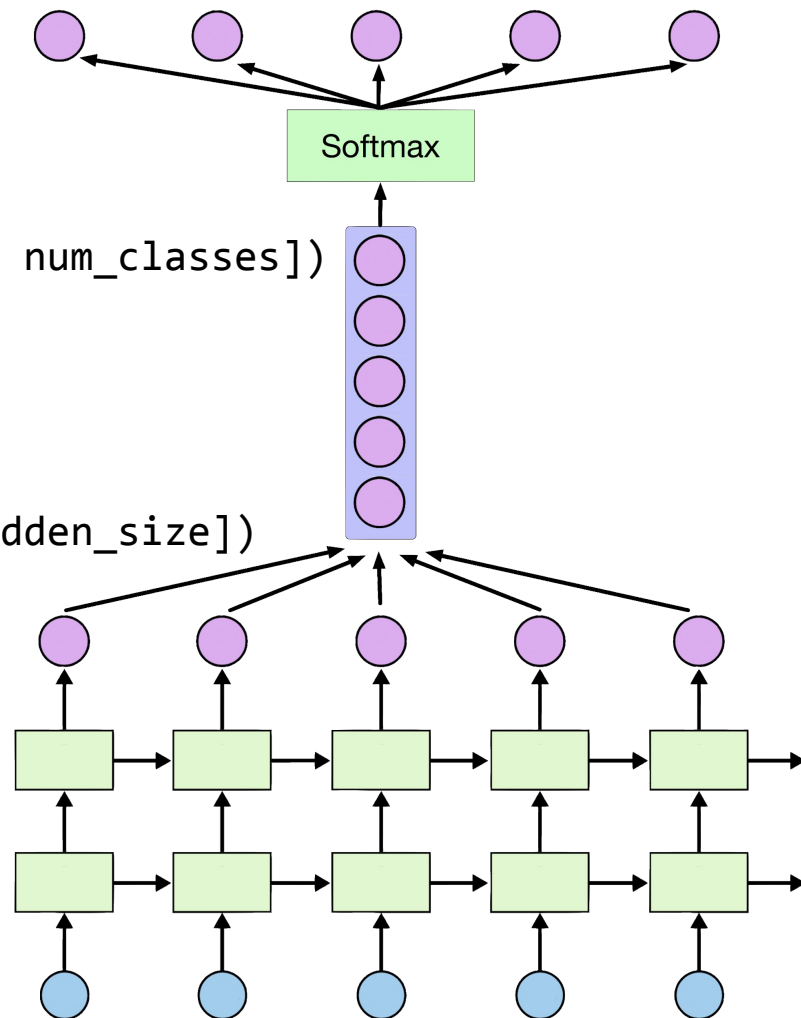
Softmax



Softmax

```
outputs = tf.reshape(outputs,  
[batch_size, seq_length, num_classes])
```

```
X_for_softmax = tf.reshape(outputs,  
[-1, hidden_size])
```



Softmax

(optional) softmax layer

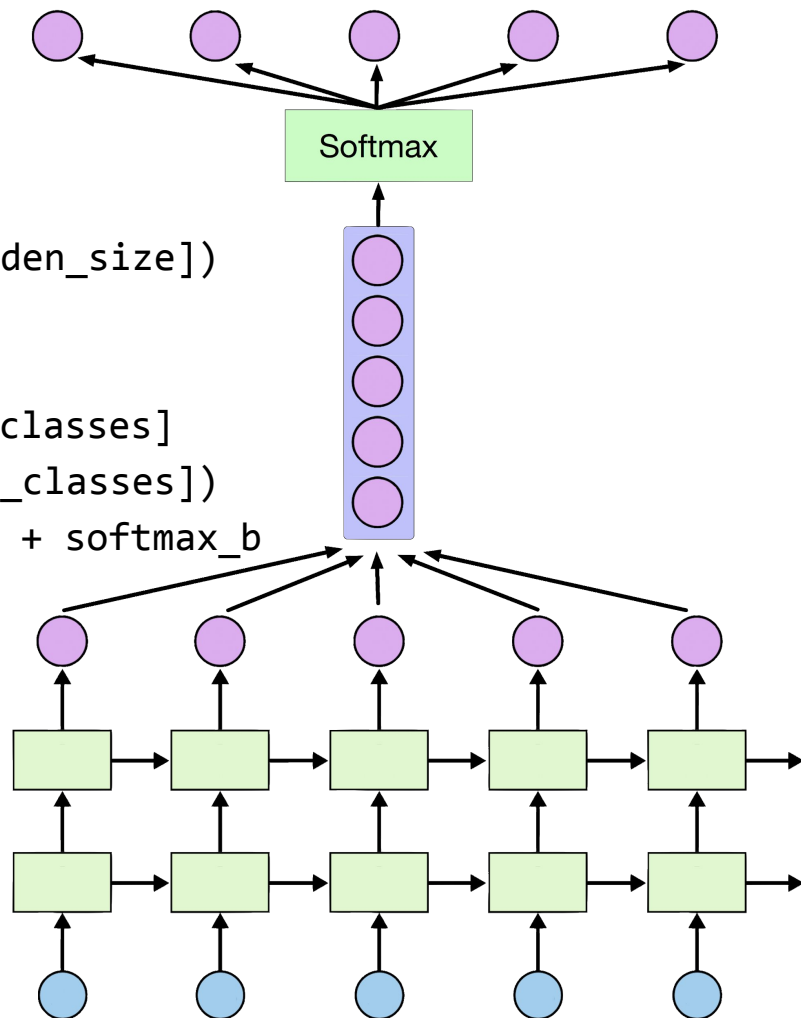
```
X_for_softmax = tf.reshape(outputs, [-1, hidden_size])
```

```
softmax_w = tf.get_variable("softmax_w",  
                             [hidden_size, num_classes])
```

```
softmax_b = tf.get_variable("softmax_b", [num_classes])
```

```
outputs = tf.matmul(X_for_softmax, softmax_w) + softmax_b
```

```
outputs = tf.reshape(outputs,  
                      [batch_size, seq_length, num_classes])
```

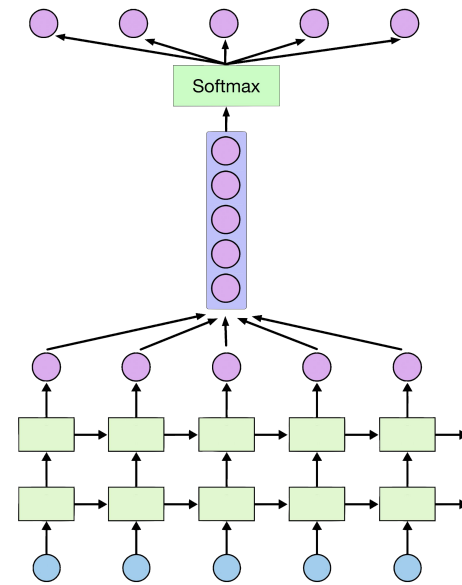


Loss

```
# reshape out for sequence_loss
outputs = tf.reshape(outputs,
                        [batch_size, seq_length, num_classes])
# All weights are 1 (equal weights)
weights = tf.ones([batch_size, seq_length])

sequence_loss = tf.contrib.seq2seq.sequence_loss(
    logits=outputs, targets=Y, weights=weights)
mean_loss = tf.reduce_mean(sequence_loss)

train_op =
    tf.train.AdamOptimizer(learning_rate=0.1).minimize(mean_loss)
```



Training and print results

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(500):
    _, l, results = sess.run(
        [train_op, mean_loss, outputs],
        feed_dict={X: dataX, Y: dataY})

    for j, result in enumerate(results):
        index = np.argmax(result, axis=1)
        print(i, j, ''.join([char_set[t] for t in index]), l)
```

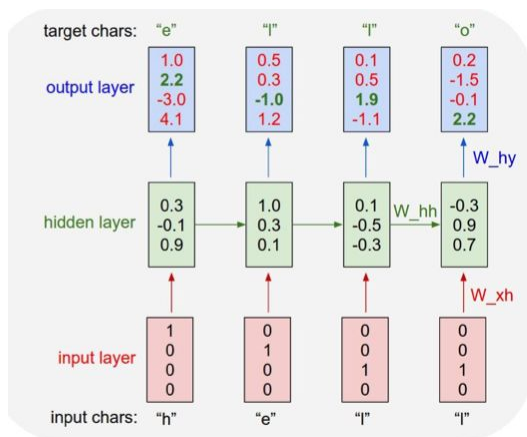
```
0 167 tttttttttt 3.23111
0 168 tttttttttt 3.23111
0 169 tttttttttt 3.23111
...
499 167 oof the se 0.229306
499 168 tf the sea 0.229306
499 169 n the sea. 0.229306
```

Training and print results

```
# Let's print the last char of each result to check it works
results = sess.run(outputs, feed_dict={X: dataX})
for j, result in enumerate(results):
    index = np.argmax(result, axis=1)
    if j is 0: # print all for the first result to make a sentence
        print(''.join([char_set[t] for t in index]), end='')
    else:
        print(char_set[index[-1]], end='')
```

g you want to build a ship, don't drum up people together to collect wood and don't assign them tasks and work, but rather teach them to long for the endless immensity of the sea.

char-rnn



Shakespeare

It looks like we can learn to spell English words. But how about if there is more structure and style in the data? To examine this I downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file. We can now afford to train a larger network, in this case lets try a 3-layer RNN with 512 hidden nodes on each layer. After we train the network for a few hours we obtain samples such as:

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nudes begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

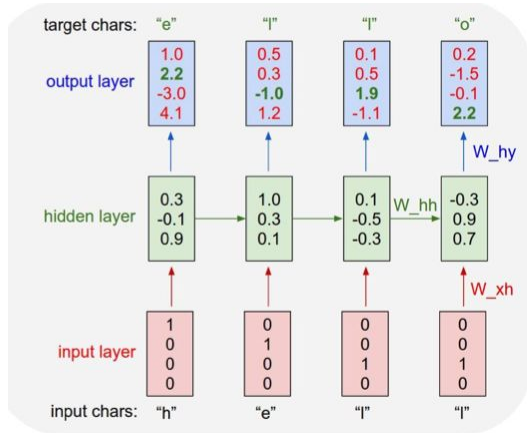
Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

Linux Source Code

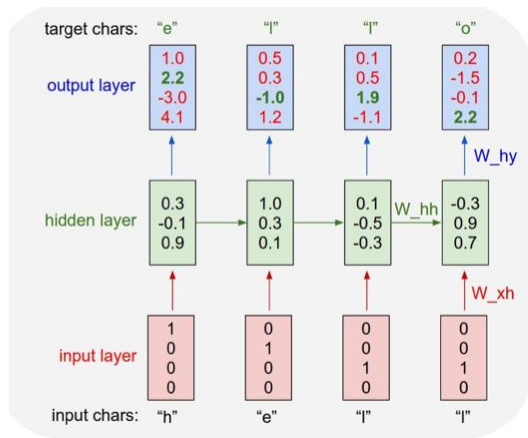
I wanted to push structured data to its limit, so for the final challenge I decided to use code. In particular, I took all the source and header files found in the [Linux repo on Github](#), concatenated all of them in a single giant file (474MB of C code) (I was originally going to train only on the kernel but that by itself is only ~16MB). Then I trained several as-large-as-fits-on-my-GPU 3-layer LSTMs over a period of a few days. These models have about 10 million parameters, which is still on the lower end for RNN models. The results are superfun:



```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
        if (fd) {
            current = blocked;
        }
    }
    rw->name = "Getjbbregs";
    bprm_self_clearl(&iv->version);
    regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
    return segtable;
}
```

<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

char/word rnn (char/word level n to n model)



<https://github.com/sherjilozair/char-rnn-tensorflow>

<https://github.com/hunkim/word-rnn-tensorflow>

With TF 1.0!

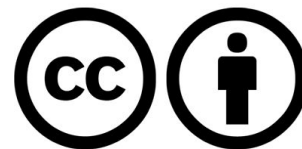


Lab 12-5





















Dynamic RNN

Sung Kim <hunkim+ml@gmail.com>

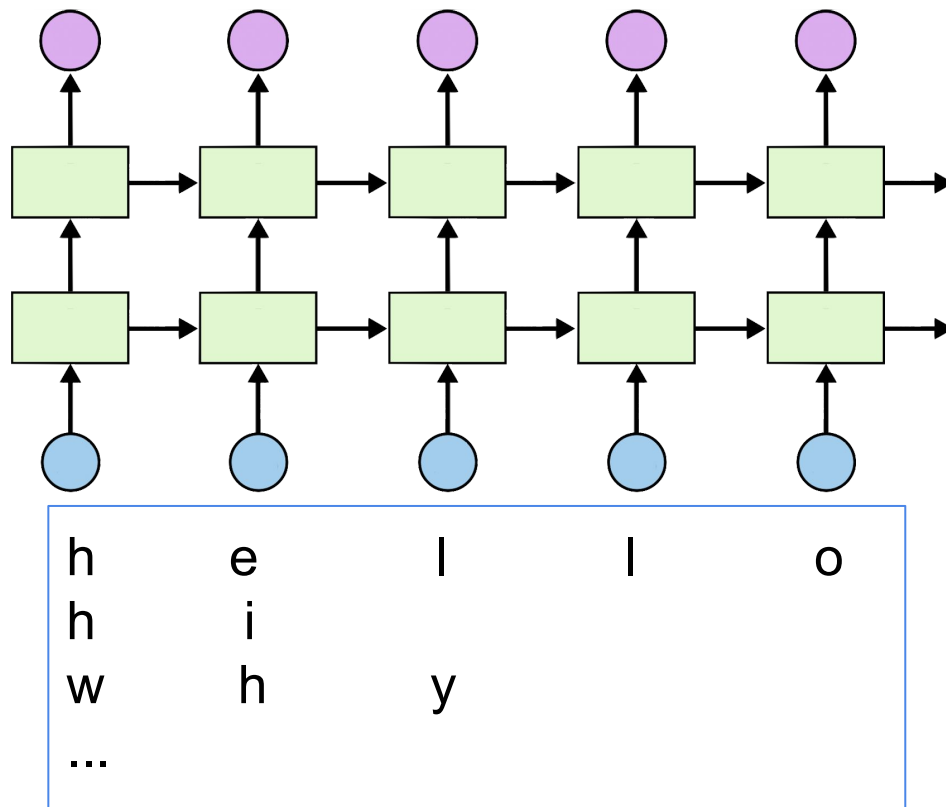
Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



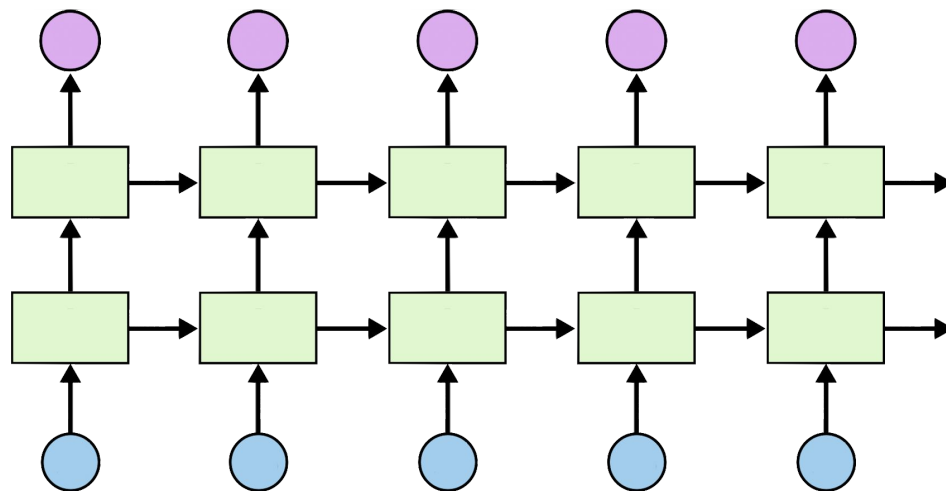
<https://github.com/hunkim/DeepLearningZeroToAll/>

 sxscience 1 commit / 350 ++ / 0 --	#12	 jennykang 19 commits / 940 ++ / 253 --	#2	 GzuPark 15 commits / 49 ++ / 39 --	#3	 kkweon 12 commits / 1,087 ++ / 340 --	#4
 BlueMelon715 5 commits / 55 ++ / 44 --	#5	 jihobak 3 commits / 244 ++ / 1,289 --	#6	 FuZer 2 commits / 37 ++ / 30 --	#7	 jin-chong 2 commits / 4 ++ / 4 --	#8
 zeran4 1 commit / 5 ++ / 4 --	#9	 cynthia 1 commit / 28 ++ / 28 --	#10	 kaka120011 1 commit / 1 ++ / 1 --	#11	 keon 1 commit / 3 ++ / 3 --	#12
 allieus 1 commit / 55 ++ / 59 --	#13	 togheppi 1 commit / 1,280 ++ / 0 --	#14	 davinnovation 1 commit / 64 ++ / 0 --	#15	 skyer9 1 commit / 94 ++ / 69 --	#16
 redongjun 1 commit / 78 ++ / 0 --	#17	 bkRyusim 1 commit / 1 ++ / 1 --	#18	 maestrojeong 1 commit / 50 ++ / 0 --	#19	 mjc92 1 commit / 1,215 ++ / 0 --	#13

Different sequence length



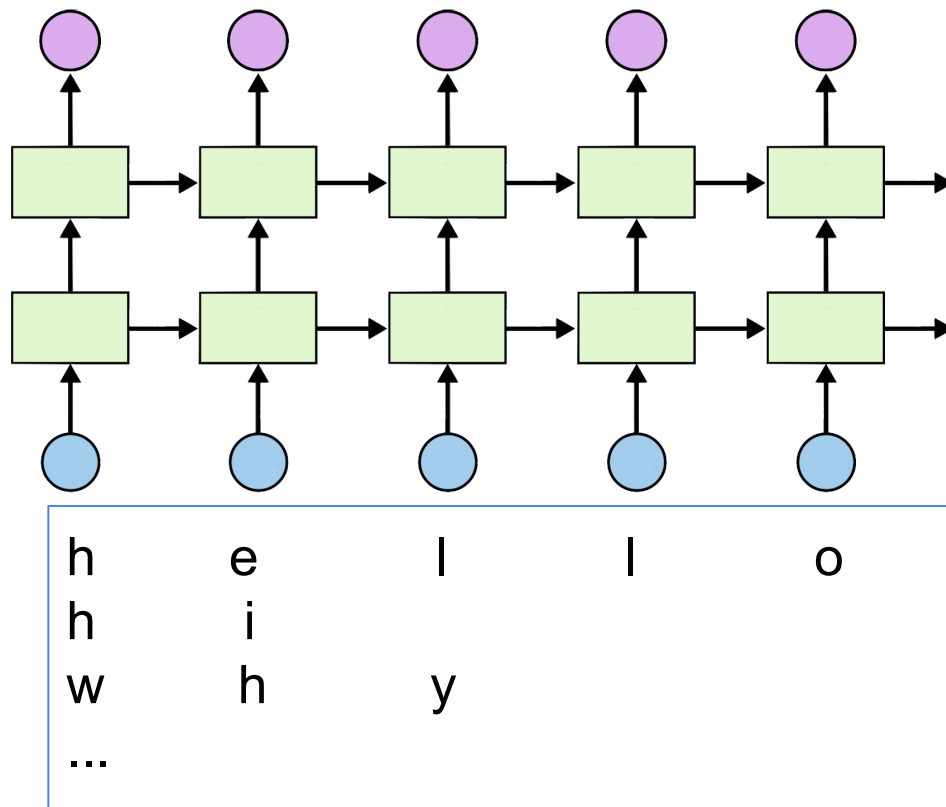
Different sequence length



h	e	l	l	o
h	i	<pad>	<pad>	<pad>
w	h	y	<pad>	<pad>
...				

Different sequence length

sequence_length=[5,2,3]



Dynamic RNN

```
# 3 batches 'hello', 'eolll', 'lleel'
x_data = np.array([[[...]]], dtype=np.float32)

hidden_size = 2
cell = rnn.BasicLSTMCell(num_units=hidden_size,
                          state_is_tuple=True)
outputs, _states = tf.nn.dynamic_rnn(
    cell, x_data, sequence_length=[5,3,4],
    dtype=tf.float32)
sess.run(tf.global_variables_initializer())
print(outputs.eval())
```

```
array([[[[-0.17904168, -0.08053244],
          [-0.01294809, 0.01660814],
          [-0.05754048, -0.1368292 ],
          [-0.08655578, -0.20553185],
          [ 0.07297077, -0.21743253]],

        [[ 0.10272847, 0.06519825],
          [ 0.20188759, -0.05027055],
          [ 0.09514933, -0.16452041],
          [ 0.          , 0.          ],
          [ 0.          , 0.          ]],

        [[-0.04893036, -0.14655617],
          [-0.07947272, -0.20996611],
          [ 0.06466491, -0.02576563],
          [ 0.15087658, 0.05166111],
          [ 0.          , 0.          ]]]], dtype=float32)
```

With TF 1.0!

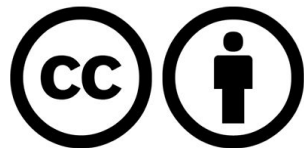


Lab 12-6





















RNN with time series data (stock)

Sung Kim <hunkim+ml@gmail.com>

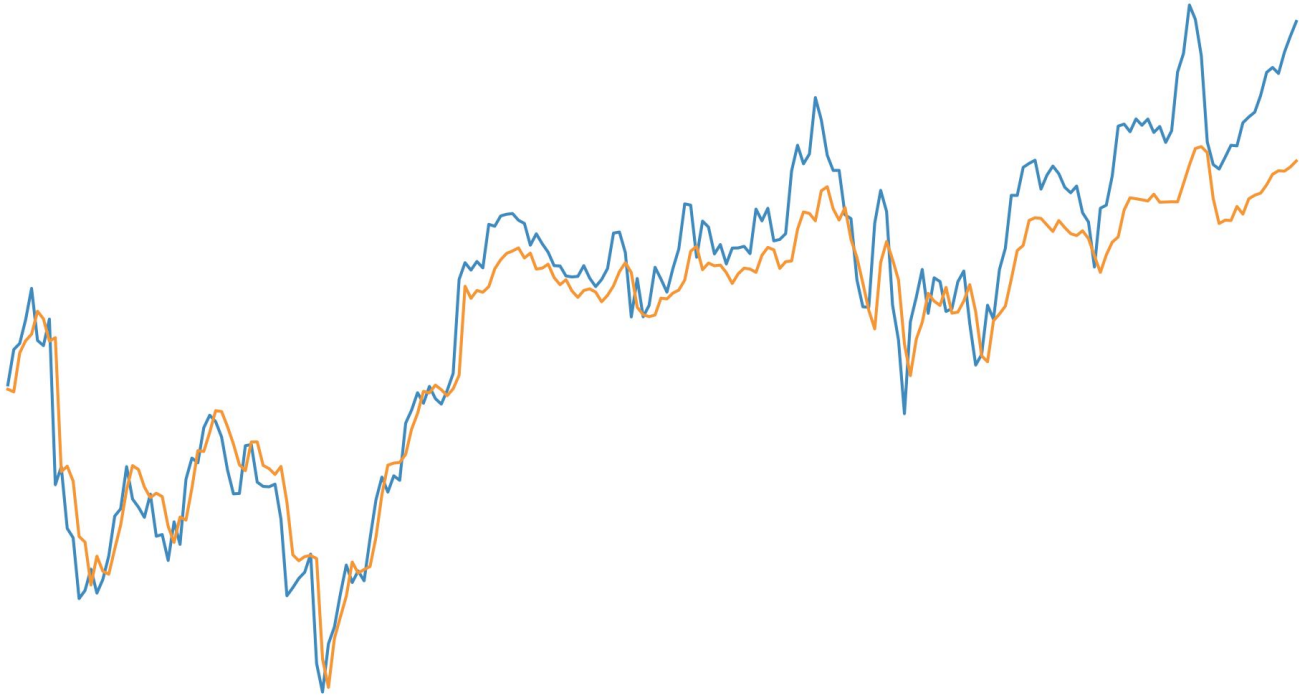
Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



<https://github.com/hunkim/DeepLearningZeroToAll/>

 sxscience 1 commit / 350 ++ / 0 --	#12	 jennykang 19 commits / 940 ++ / 253 --	#2	 GzuPark 15 commits / 49 ++ / 39 --	#3	 kkweon 12 commits / 1,087 ++ / 340 --	#4
 BlueMelon715 5 commits / 55 ++ / 44 --	#5	 jihobak 3 commits / 244 ++ / 1,289 --	#6	 FuZer 2 commits / 37 ++ / 30 --	#7	 jin-chong 2 commits / 4 ++ / 4 --	#8
 zeran4 1 commit / 5 ++ / 4 --	#9	 cynthia 1 commit / 28 ++ / 28 --	#10	 kaka120011 1 commit / 1 ++ / 1 --	#11	 keon 1 commit / 3 ++ / 3 --	#12
 allieus 1 commit / 55 ++ / 59 --	#13	 togheppi 1 commit / 1,280 ++ / 0 --	#14	 davinnovation 1 commit / 64 ++ / 0 --	#15	 skyer9 1 commit / 94 ++ / 69 --	#16
 redongjun 1 commit / 78 ++ / 0 --	#17	 bkRyusim 1 commit / 1 ++ / 1 --	#18	 maestrojeong 1 commit / 50 ++ / 0 --	#19	 mjc92 1 commit / 1,215 ++ / 0 --	#13

Time series data

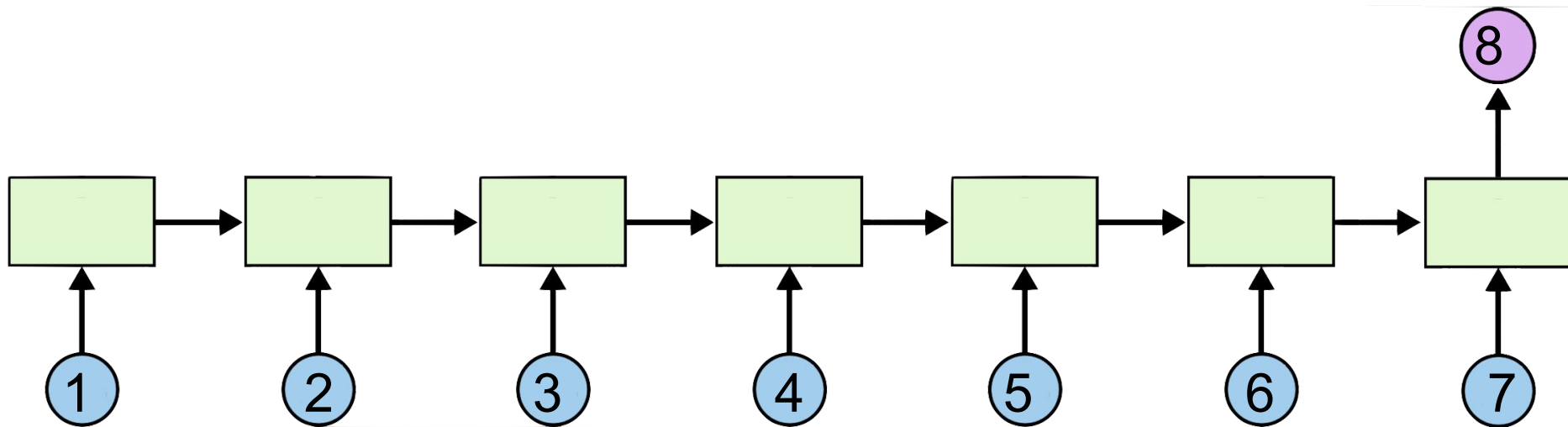


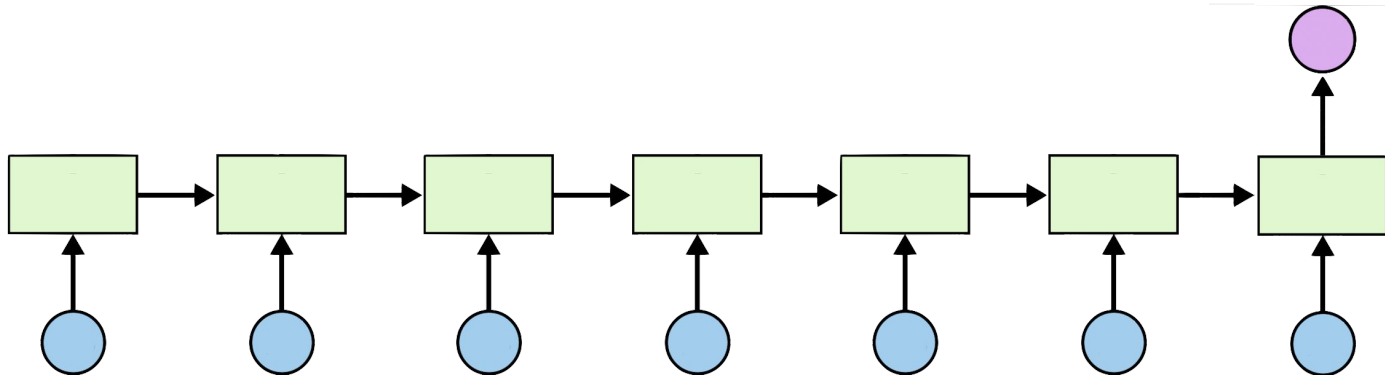
Time series data

Open	High	Low	Volume	Close
828.659973	833.450012	828.349976	1247700	831.659973
823.02002	828.070007	821.655029	1597800	828.070007
819.929993	824.400024	818.97998	1281700	824.159973
819.359985	823	818.469971	1304000	818.97998
819	823	816	1053600	820.450012
816	820.958984	815.48999	1198100	819.23999
811.700012	815.25	809.780029	1129100	813.669983
809.51001	810.659973	804.539978	989700	809.559998
807	811.840027	803.190002	1155300	808.380005

'data-02-stock_daily.csv'

Many to one





Open	High	Low	Volume	Close
828.659973	833.450012	828.349976	1247700	831.659973
823.02002	828.070007	821.655029	1597800	828.070007
819.929993	824.400024	818.97998	1281700	824.159973
819.359985	823	818.469971	1304000	818.97998
819	823	816	1053600	820.450012
816	820.958984	815.48999	1198100	819.23999
811.700012	815.25	809.780029	1129100	813.669983
809.51001	810.659973	804.539978	989700	?
807	811.840027	803.190002	1155300	?

Reading data

```
timesteps = seq_length = 7
data_dim = 5
output_dim = 1

# Open, High, Low, Close, Volume
xy = np.loadtxt('data-02-stock_daily.csv', delimiter=',')
xy = xy[::-1] # reverse order (chronically ordered)
xy = MinMaxScaler(xy)
x = xy
y = xy[:, [-1]] # Close as label

dataX = []
dataY = []
for i in range(0, len(y) - seq_length):
    _x = x[i:i + seq_length]
    _y = y[i + seq_length] # Next close price
    print(_x, "->", _y)
    dataX.append(_x)
    dataY.append(_y)
```

```
[ 0.18667876  0.20948057  0.20878184  0.
 0.21744815]
```

```
[ 0.30697388  0.31463414  0.21899367
 0.01247647  0.21698189]
```

```
[ 0.21914211  0.26390721  0.2246864
 0.45632338  0.22496747]
```

```
[ 0.23312993  0.23641916  0.16268272
 0.57017119  0.14744274]
```

```
[ 0.13431201  0.15175877  0.11617252
 0.39380658  0.13289962]
```

```
[ 0.13973232  0.17060429  0.15860382
 0.28173344  0.18171679]
```

```
[ 0.18933069  0.20057799  0.19187983
 0.29783096  0.2086465 ]]
```

```
-> [ 0.14106001]
```

Training and test datasets

split to train and testing

```
train_size = int(len(dataY) * 0.7)
```

```
test_size = len(dataY) - train_size
```

```
trainX, testX = np.array(dataX[0:train_size]),  
                np.array(dataX[train_size:len(dataX)])
```

```
trainY, testY = np.array(dataY[0:train_size]),  
                np.array(dataY[train_size:len(dataY)])
```

input placeholders

```
X = tf.placeholder(tf.float32, [None, seq_length, data_dim])
```

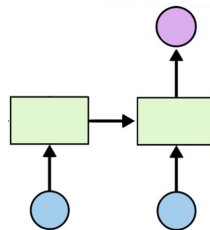
```
Y = tf.placeholder(tf.float32, [None, 1])
```

LSTM and Loss

input placeholders

```
X = tf.placeholder(tf.float32, [None, seq_length, data_dim])
```

```
Y = tf.placeholder(tf.float32, [None, 1])
```



```
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_dim, state_is_tuple=True)
```

```
outputs, _states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)
```

```
Y_pred = tf.contrib.layers.fully_connected(  
    outputs[:, -1], output_dim, activation_fn=None)
```

We use the last cell's output

cost/loss

```
loss = tf.reduce_sum(tf.square(Y_pred - Y)) # sum of the squares
```

optimizer

```
optimizer = tf.train.AdamOptimizer(0.01)
```

```
train = optimizer.minimize(loss)
```

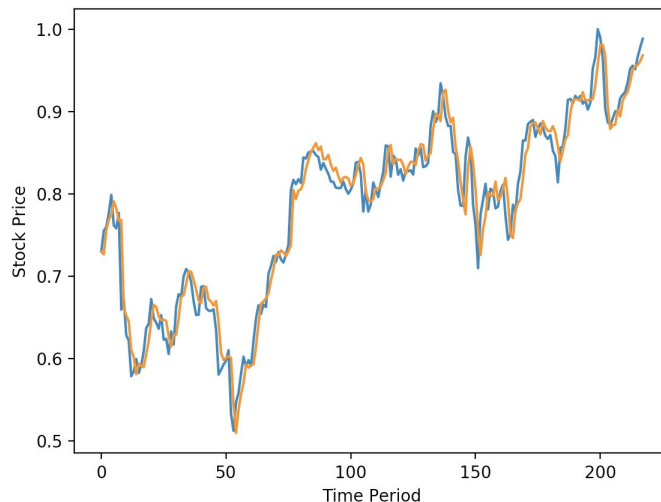
Training and Results

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(1000):
    _, l = sess.run([train, loss],
                    feed_dict={X: trainX, Y: trainY})
    print(i, l)

testPredict = sess.run(Y_pred, feed_dict={X: testX})

import matplotlib.pyplot as plt
plt.plot(testY)
plt.plot(testPredict)
plt.show()
```



Exercise

- Implement stock prediction using linear regression only
- Improve results using more features such as keywords and/or sentiments in top news

Other RNN applications

- Language Modeling
- Speech Recognition
- Machine Translation
- Conversation Modeling/Question Answering
- Image/Video Captioning
- Image/Music/Dance Generation