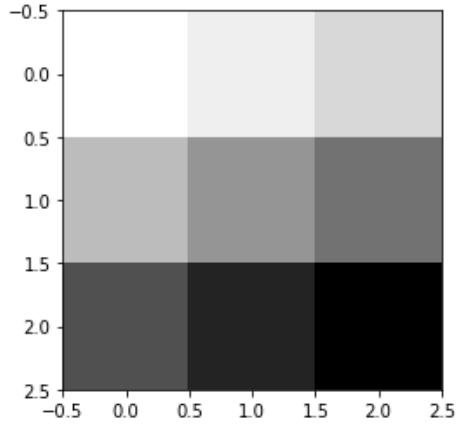
ML_Lab 11-1.

Test.

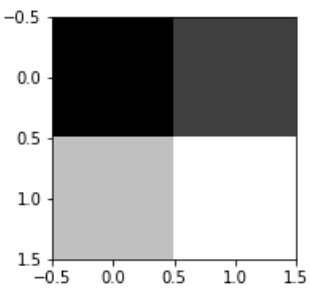
```
sess = tf.InteractiveSession()
image = np.array([[[[1],[2],[3]],
                   [[4],[5],[6]],
                   [[7],[8],[9]]]], dtype=np.float32)
print(image.shape)
plt.imshow(image.reshape(3,3), cmap='Greys') #.imshow()
```

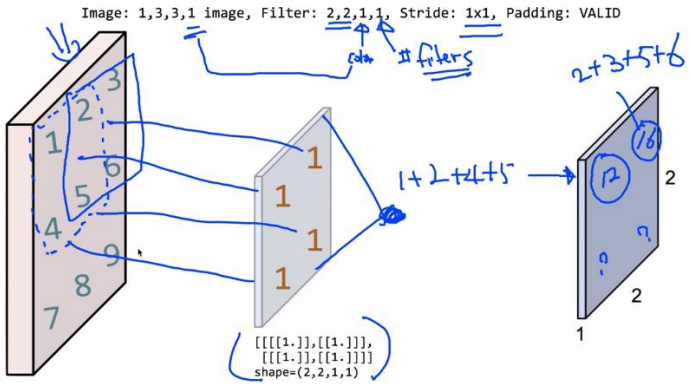
```
(1, 3, 3, 1)
/usr/local/lib/python3.6/dist-packages/tensorflow/python/c
  warnings.warn('An interactive session is already active.
<matplotlib.image.AxesImage at 0x7f63185be630>
```




```
print("image.shape", image.shape)
weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='VALID') #.nn.conv2d()
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3) #.swapaxes()
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(2,2))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(2,2), cmap='gray') #.subplot()
```

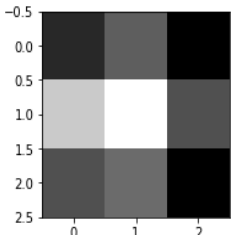
```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 2, 2, 1)
[[12. 16.]
 [24. 28.]]
```





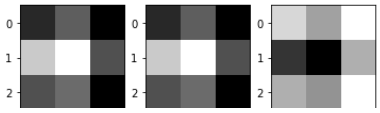
```
print("image.shape", image.shape)

weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d_img.shape (1, 3, 3, 1)
[[12. 16.  9.]
 [24. 28. 15.]
 [15. 17.  9.]]
```




```
print("image.shape", image.shape)
weight = tf.constant([[[[1.,10.,-1.]],[[1.,10.,-1.]]],[[[1.,10.,-1.]],[[1.,10.,-1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,3,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

```
image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 3)
conv2d_img.shape (1, 3, 3, 3)
[[12. 16.  9.]
 [24. 28. 15.]
 [15. 17.  9.]]
[[120. 160.  90.]
 [240. 280. 150.]
 [150. 170.  90.]]
[[-12. -16.  -9.]
 [-24. -28. -15.]
 [-15. -17.  -9.]]
```




```
image = np.array([[[[4],[3]],
                   [[2],[1]]]], dtype=np.float32)
pool = tf.nn.max_pool(image, ksize=[1, 2, 2, 1], #.nn.max_pool()
                      strides=[1, 1, 1, 1], padding='VALID')
print(pool.shape)
print(pool.eval())
```

```
(1, 1, 1, 1)
[[[[4.]]]]
```

```
image = np.array([[[[4],[3]],
                   [[2],[1]]]], dtype=np.float32)
pool = tf.nn.max_pool(image, ksize=[1, 2, 2, 1],
                      strides=[1, 1, 1, 1], padding='SAME')
print(pool.shape)
print(pool.eval())
```

```
(1, 2, 2, 1)
[[[[4.]
   [3.]]

  [[2.]
   [1.]]]]
```

ML_Lab 11-2. 3.

//Colab 에서 Tensorflow 1.15.2 를 불러와 실행할 수 있다. 시간이 상당히 많이 소요된다.

//이전에 GitHub, 강의 사진으로 대체했던 Lab 도 1.15.2 를 불러와 해결할 수 있었다.

```
Learning started. It takes sometime.
Epoch: 0001 cost = 0.365078180
Epoch: 0002 cost = 0.102141358
Epoch: 0003 cost = 0.074173531
Epoch: 0004 cost = 0.060685895
Epoch: 0005 cost = 0.050624732
Epoch: 0006 cost = 0.046288653
Epoch: 0007 cost = 0.042086853
Epoch: 0008 cost = 0.039240586
Epoch: 0009 cost = 0.036376112
Epoch: 0010 cost = 0.035613905
Epoch: 0011 cost = 0.028902437
Epoch: 0012 cost = 0.030085018
Epoch: 0013 cost = 0.027165487
Epoch: 0014 cost = 0.026869907
Epoch: 0015 cost = 0.026782614
Learning Finished!
Accuracy: 0.9945
Label:  [1]
Prediction:  [1]
```

//실제로 1.15.2 를 불러와 mnist_cnn 의 결과가 나왔다.

https://raw.githubusercontent.com/hunkim/DeepLearningZeroToAll/master/lab-11-1-mnist_cnn.py

https://raw.githubusercontent.com/hunkim/DeepLearningZeroToAll/master/lab-11-2-mnist_deep_cnn.py

https://raw.githubusercontent.com/hunkim/DeepLearningZeroToAll/master/lab-11-3-mnist_cnn_class.py

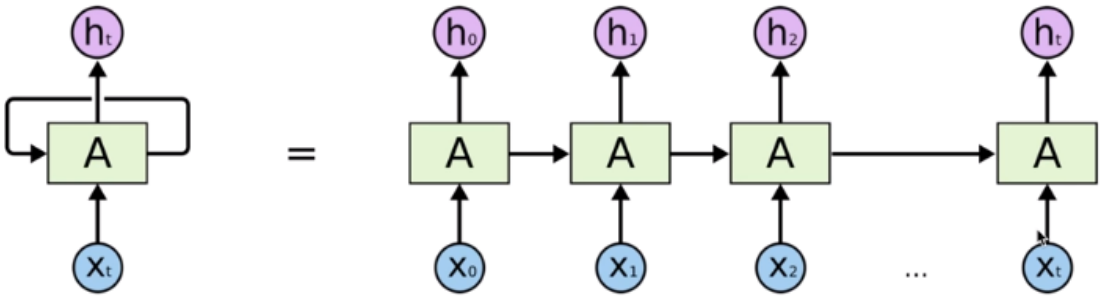
https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-11-4-mnist_cnn_layers.py

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-11-5-mnist_cnn_ensemble_layers.py

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-11-X-mnist_cnn_low_memory.py

ML_Lec 12.

Sequence Data.



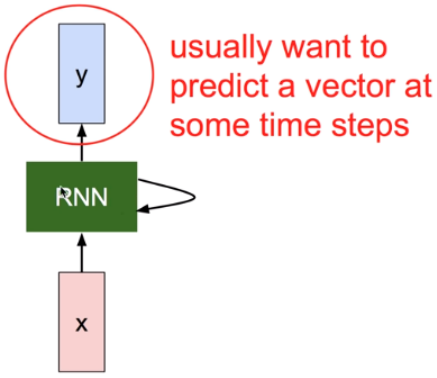
1. We don't understand one word only.
2. We understand based on the previous words + this word. (time series)
3. NN/CNN cannot do this.
4. Notice: the same function and the same set of parameters are used at every time step.

Recurrent Neural Network. //순환신경망

$$h_t = f_W(h_{t-1}, x_t)$$

new state / old state / input vector at some time step

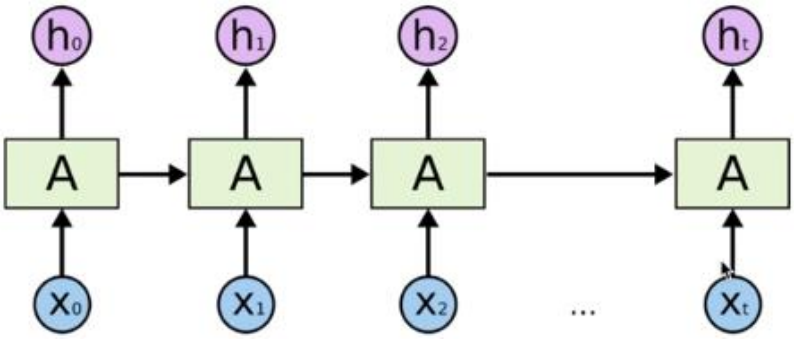
some function with parameters W

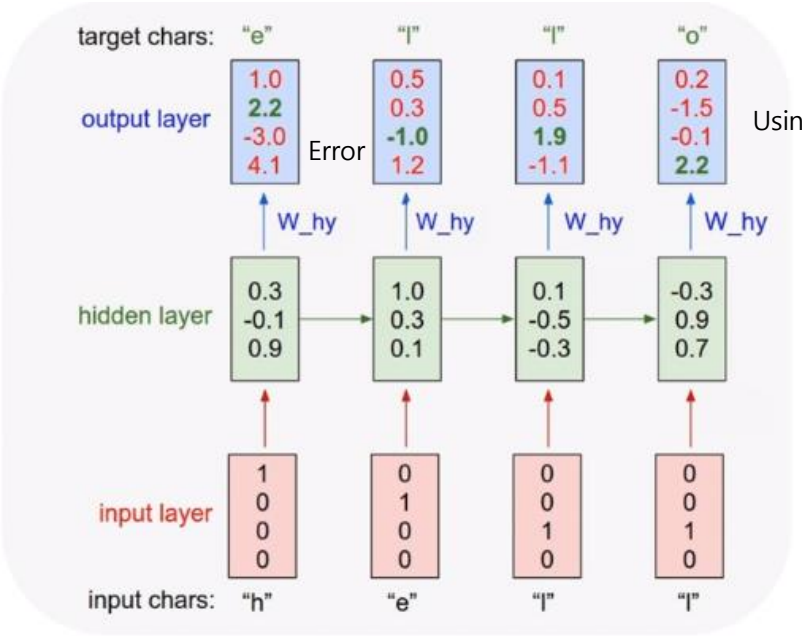


♠ Vanilla //The state consists of a single "hidden" vector h

1) $h_t = f_w(h_{t-1}, x_t)$
2) $h_t = tanh(W_{hh}h_{t-1}, w_{xh}x_t)$
3) $y_t = W_{hy}h_t$

Character-level Language Model Example.



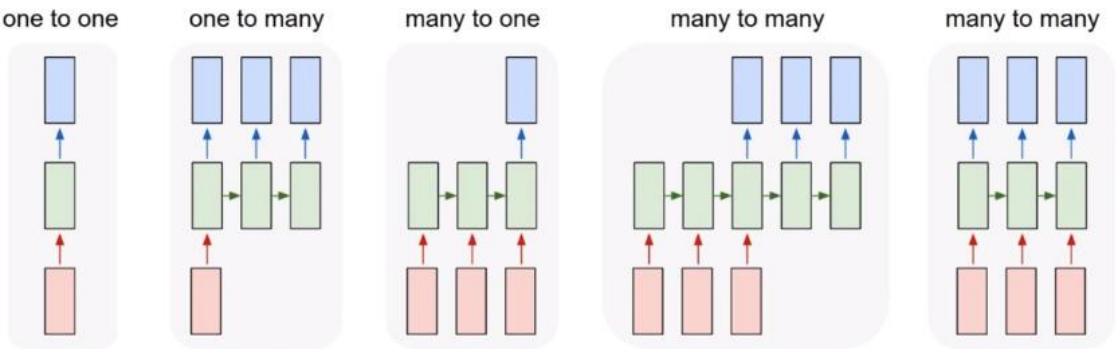


Using Softmax

Vocabulary: [h, e, l, o]

Example Training Sequence: "hello"

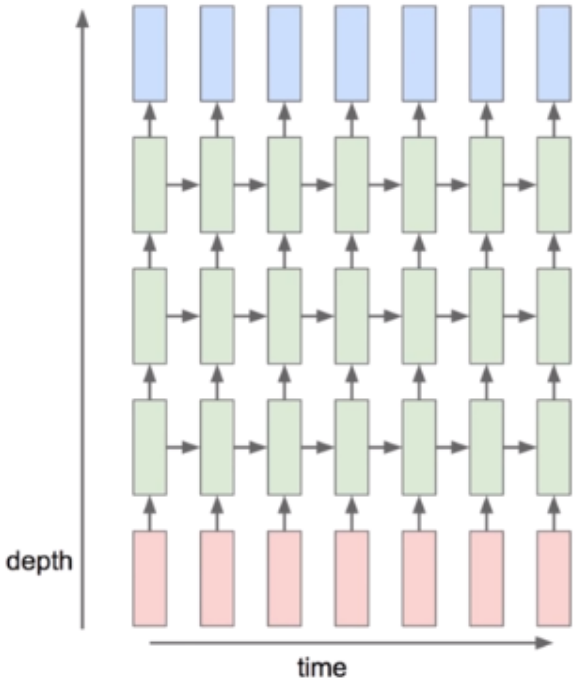
RNN Applications.



Vanilla Image Captioning Sentiment Classification Machine Translation Video Classification

1. Language Modeling
2. Speech Recognition
3. Machine Translation
4. Conversation Modeling/Question Answering
5. Image/Video Captioning
6. Image/Video/Dance Generation

Multi-Layer RNN.



Training RNNs is Challenging.

♠ Several Advanced Models

1) Long Short Term Memory (LSTM)
2) GRU by Cho et al.2014

ML_Lab 12-1.

Test.

```
1 import tensorflow as tf
2 import numpy as np
3 from tensorflow.contrib import rnn
4 import pprint
5 pp = pprint.PrettyPrinter(indent=4)
6 sess = tf.InteractiveSession()
```

```
[4]  1 h = [1, 0, 0, 0]
     2 e = [0, 1, 0, 0]
     3 l = [0, 0, 1, 0]
     4 o = [0, 0, 0, 1]
```

```
[5]  1 # One cell RNN input_dim (4) -> output_dim (2)
     2 hidden_size = 2
     3 cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size) #.contrib.rnn.BasicRNNCell()
     4
     5 x_data = np.array([[h]], dtype=np.float32)
     6 outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32) #.nn.dynamic_rnn()
     7
     8 sess.run(tf.global_variables_initializer())
     9 pp.pprint(outputs.eval())
```

```
1 # One cell RNN input_dim (4) -> output_dim (2). sequence: 5
2 hidden_size = 2
3 cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)
4 x_data = np.array([[h, e, l, l, o]], dtype=np.float32)
5 print(x_data.shape)
6 pp.pprint(x_data)
7 outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
8 sess.run(tf.global_variables_initializer())
9 pp.pprint(outputs.eval())
```

```
[ ]  1 # One cell RNN input_dim (4) -> output_dim (2). sequence: 5, batch 3
     2 x_data = np.array([[h, e, l, l, o],
     3                    [e, o, l, l, l],
     4                    [l, l, e, e, l]], dtype=np.float32)
     5 pp.pprint(x_data)
     6
     7 hidden_size = 2
     8 cell = rnn.BasicLSTMCell(num_units=hidden_size, state_is_tuple=True) #.BasicLSTMCell()
     9 outputs, _states = tf.nn.dynamic_rnn(
    10     cell, x_data, dtype=tf.float32)
    11 sess.run(tf.global_variables_initializer())
    12 pp.pprint(outputs.eval())
```

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

ML_Lab 12-2.

Test.

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)
```

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

num_classes = 5
input_dim = 5
hidden_size = 5
batch_size = 1
sequence_length = 6
learning_rate = 0.1

X = tf.placeholder(
    tf.float32, [None, sequence_length, input_dim])  # 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) #.zero_state()
outputs, _states = tf.nn.dynamic_rnn(
    cell, X, initial_state=initial_state, dtype=tf.float32)

# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])
# fc_w = tf.get_variable("fc_w", [hidden_size, num_classes])
# fc_b = tf.get_variable("fc_b", [num_classes])
# outputs = tf.matmul(X_for_fc, fc_w) + fc_b
outputs = tf.contrib.layers.fully_connected( #.contrib.layers.fully_connected()
    inputs=X_for_fc, num_outputs=num_classes, activation_fn=None)

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

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

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

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(50):
        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)

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

```

```
        Prediction str:  ihello
23 loss: 0.01314158 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
24 loss: 0.010803193 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
```

ML_Lab 12-3.

Test.

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

# 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 rollings (unit #)
learning_rate = 0.2

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

# flatten the data (ignore batches for now). No effect if the batch size is 1
X_one_hot = tf.one_hot(X, num_classes)  # one hot: 1 -> 0 1 0 0 0 0 0 0 0 0
X_for_softmax = tf.reshape(X_one_hot, [-1, rnn_hidden_size])

# softmax layer (rnn_hidden_size -> num_classes)
softmax_w = tf.get_variable("softmax_w", [rnn_hidden_size, num_classes])
softmax_b = tf.get_variable("softmax_b", [num_classes]) #.get_variable()
outputs = tf.matmul(X_for_softmax, softmax_w) + softmax_b
```

```
# expend the data (revive the batches)
outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])
weights = tf.ones([batch_size, sequence_length])

# Compute sequence cost/loss
sequence_loss = tf.contrib.seq2seq.sequence_loss(
    logits=outputs, targets=Y, weights=weights)
loss = tf.reduce_mean(sequence_loss)  # mean all sequence loss
train = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

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

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))
```

```
2773 loss: 0.2772864 Prediction: yf you yant you
2774 loss: 0.2772864 Prediction: yf you yant you
2775 loss: 0.27728638 Prediction: yf you yant you
2776 loss: 0.27728635 Prediction: yf you yant you
2777 loss: 0.27728635 Prediction: yf you yant you
2778 loss: 0.27728626 Prediction: yf you yant you
```

ML_Lab 12-4.

Test.

```
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)}

data_dim = len(char_set)
hidden_size = len(char_set)
num_classes = len(char_set)
sequence_length = 10  # Any arbitrary number
learning_rate = 0.1

dataX = []
dataY = []
for i in range(0, len(sentence) - sequence_length):
    x_str = sentence[i:i + sequence_length]
    y_str = sentence[i + 1: i + sequence_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)

batch_size = len(dataX)
```

```
X = tf.placeholder(tf.int32, [None, sequence_length])
Y = tf.placeholder(tf.int32, [None, sequence_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)
def lstm_cell():
    cell = rnn.BasicLSTMCell(hidden_size, state_is_tuple=True)
    return cell

multi_cells = rnn.MultiRNNCell([lstm_cell() for _ in range(2)], state_is_tuple=True)
#.MultiRNNCell()

# outputs: unfolding size x hidden size, state = hidden size
outputs, _states = tf.nn.dynamic_rnn(multi_cells, X_one_hot, dtype=tf.float32)

# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])
outputs = tf.contrib.layers.fully_connected(X_for_fc, num_classes, activation_fn=None)

# reshape out for sequence_loss
outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])

# All weights are 1 (equal weights)
weights = tf.ones([batch_size, sequence_length])
```

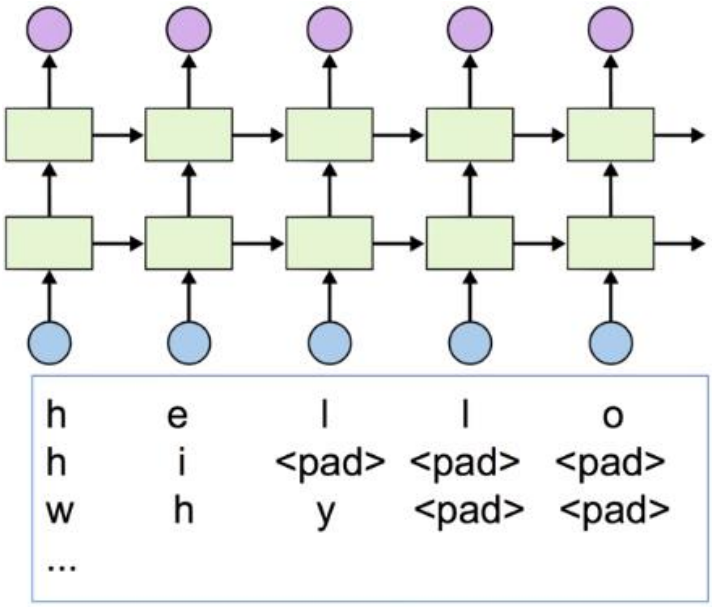
```
sequence_loss = tf.contrib.seq2seq.sequence_loss(
    logits=outputs, targets=Y, weights=weights)
mean_loss = tf.reduce_mean(sequence_loss)
train_op = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(mean_loss)

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)

# 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='')
```

```
499 116 tather tea 0.22881413
499 117  ther teac 0.22881413
499 118 nher teach 0.22881413
499 119  er teach  0.22881413
```

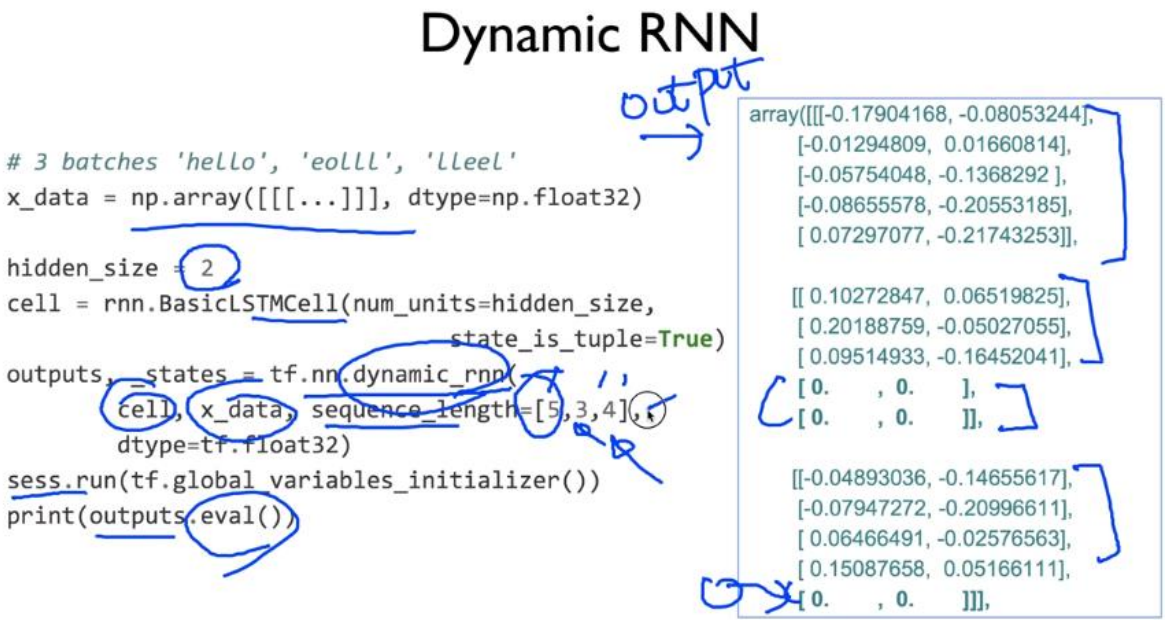
ML_Lab 12-5.

Different Sequence Length.

Sequence_length=[5, 2, 3]



Dynamic RNN.

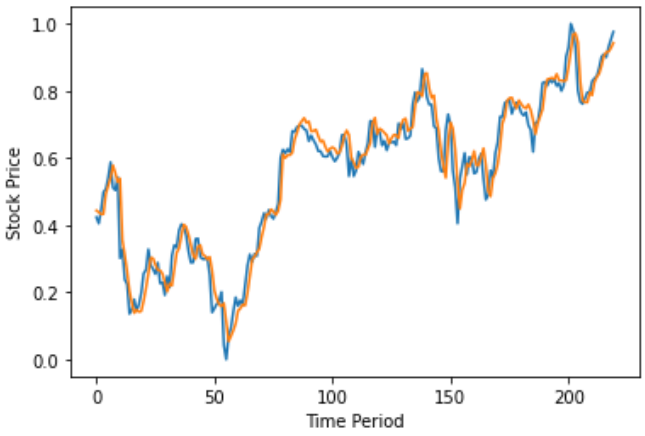


ML_Lab 12-6.

Test.

```
[step: 475] loss: 0.7169285416603088
[step: 476] loss: 0.7162967920303345
[step: 477] loss: 0.7156656980514526
[step: 478] loss: 0.7150356769561768
[step: 479] loss: 0.7144066095352173
[step: 480] loss: 0.713778018951416
[step: 481] loss: 0.7131505608558655
[step: 482] loss: 0.7125238180160522
[step: 483] loss: 0.7118977308273315
[step: 484] loss: 0.7112727165222168
[step: 485] loss: 0.710648238658905
[step: 486] loss: 0.710024356842041
[step: 487] loss: 0.7094016075134277
[step: 488] loss: 0.7087793350219727
[step: 489] loss: 0.7081579566001892
[step: 490] loss: 0.7075372338294983
[step: 491] loss: 0.706917226314547
[step: 492] loss: 0.7062978744506836
[step: 493] loss: 0.705679178237915
[step: 494] loss: 0.7050612568855286
[step: 495] loss: 0.7044441103935242
[step: 496] loss: 0.7038273811340332
[step: 497] loss: 0.7032114863395691
[step: 498] loss: 0.7025963068008423
[step: 499] loss: 0.7019817233085632
RMSE: 0.05135541409254074
```

```
1 # Plot predictions
2     plt.plot(testY)
3     plt.plot(test_predict)
4     plt.xlabel("Time Period") #.xlabel()
5     plt.ylabel("Stock Price") #.ylabel()
6     plt.show()
```



https://raw.githubusercontent.com/hunkim/DeepLearningZeroToAll/master/lab-12-5-rnn_stock_prediction.py

Memo.

.constant()	//상수
.Session()	//tensor에 데이터를 넣어 흐르게 함.
.run()	//실행
.add()	//더하기
.placeholeder(), feed_dict={a:a_data}	//변수, 값을 나중에 할당.
.Variable()	//변수, 자동으로 업데이트.
.random_normal(Shapes)	//랜덤 값 반환
.reduce_mean()	//평균
.square()	//제곱
.GradientDescentOptimizer()	//미니 배치 확률적 경사하강법(SGD) 구현.
.minimize()	//최소화
.global_variables_initializer()	//.Variable()를 초기화.
.append()	//append
.plot()	//plot
.show()	//show
.reduce_sum()	//총합
.assign()	//.Variable()의 값 변경.
.compute_gradients()	//compute_gradients
.apply_gradients()	//apply_gradients
.matmul()	//matmul
.loadtext()	//text 불러오기.
.set_random_seed()	//랜덤 값 시드, 다른 환경에서도 같다.
.string_input_producer()	//Queue, text 를 Filename Queue 에 쌓기.
.TextLineReader()	//Queue, text 를 Reader 로 연결.
.read()	//Queue, text 읽기.

```
.decode_csv()                          //Queue, text decode
.batch()                               //Queue, text batch
.Coordinator()                         //Queue, Coordinator 생성.
.start_queue_runners()                 //Queue, Queue 를 Thread 로 시작.
.request_stop()                        //Queue, 중지
.join()                                //Queue, 대기
.sigmoid()                             //S 자 곡선
.log()                                 //로그
.cast()                                //새로운 자료형
.equal()                               //값이 같은지
.softmax()                             //softmax
.arg_max()                             //arg_max
.one_hot()                             //one_hot
.reshape()                             //reshape
.softmax_cross_entropy_with_logits()   //softmax_cross_entropy_with_logits
.format()                              //format
.flatten()                             //flatten
.PrettyPrinter()                       //PrettyPrinter
.InteractiveSession()                  //InteractiveSession
.array()                               //Array
.pprint()                              //Pprint
.shape()                               //Shape
.eval()                                //Eval
.squeeze()                             //Array 정리
.expand_dims()                         //Array 정렬
.stack()                               //Array 쌓기
```

.ones_like() //One 으로 바꿈.

.zeros_like() //Zero 로 바꿈.

zip() //Zip

.nn.relu() //Relu

.random.randn() //지정 범위 내 랜덤 값 반환

.nn.dropout() //Dropout

.imshow() //Imshow

.nn.conv2d() //Conv2d

.swapaxes() //Swapaxes

.subplot() //Subplot

.nn.max_pool() //Maxpool

.contrib.rnn.BasicRNNCell() //Basic RNN Cell

.nn.dynamic_rnn() //Dynamic Rnn

.BasicLSTMCell() //Basic LSTM Cell

.zero_state() //Zero State

.contrib.layers.fully_connected() //Fully Connected

.ones() //Ones

.contrib.seq2seq.sequence_loss() //Sequence Loss

.AdamOptimizer() //AdamOptimizer

.get_variable() //Get Variable

.MultiRNNCell() //Multi RNN Cell

.min() //최소

.max() //최대

MinMaxScaler //MinMaxScaler

.xlabel() //X Label

.ylabel() //Y Label

//Lab으로 연결 https://colab.research.google.com/drive/1gaTpEufmhoK2CsEsNyfDDtyynQ_HRpSu

//14폰트, 12폰트, 10폰트

//1. 1) a. *♠

//0.71 1.34