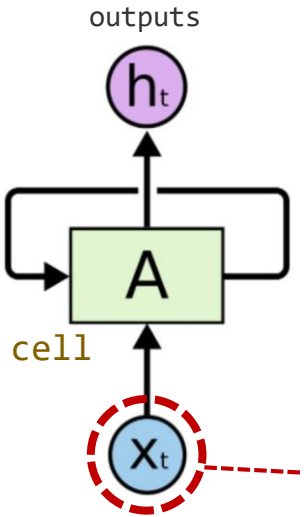
RNN, LSTM

Lecture 12

Contents

1 RNN Bssics

RNN in TensorFlow



cell 생성

1

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

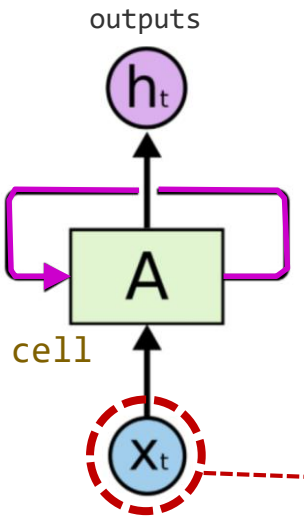
Output size = Hidden size

2 cell 작동

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

입력 값

RNN in TensorFlow



다양한 성능평가를 위해서
cell name을 교체해가면서 test

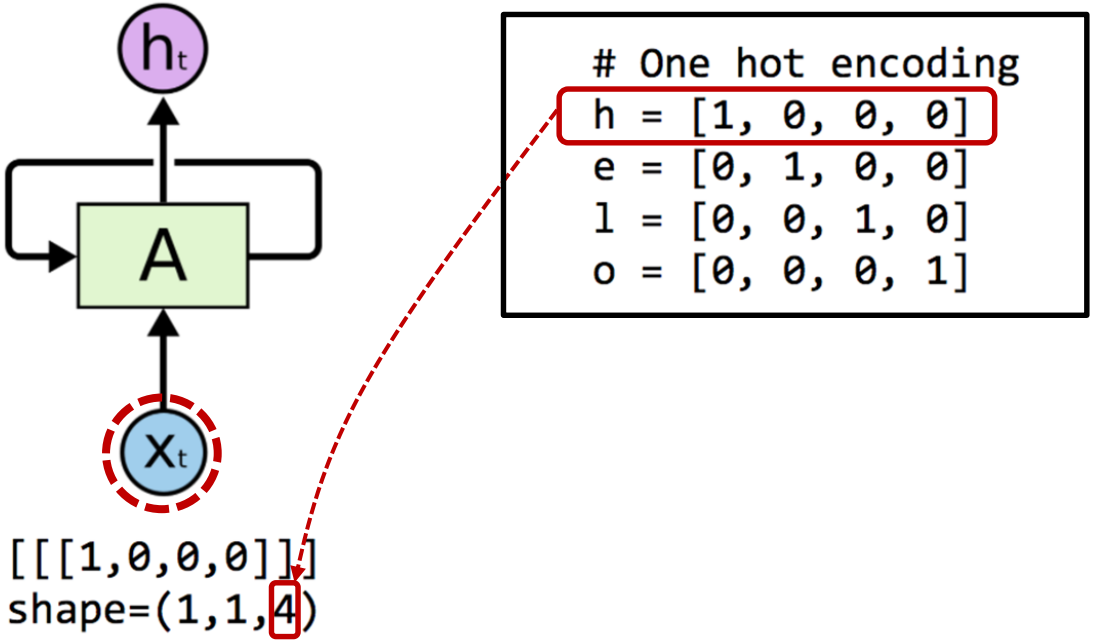
1

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

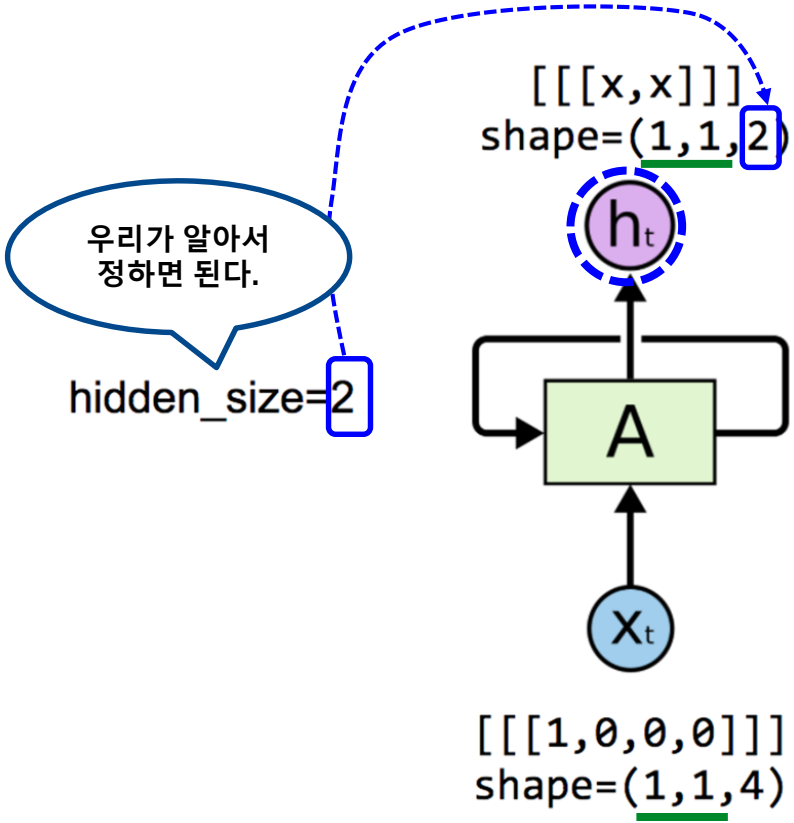
2

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

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



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]
```

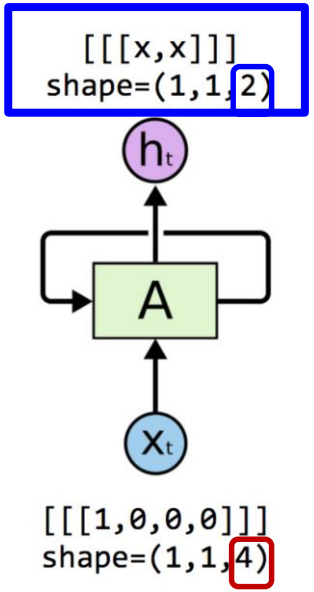
One node: 4 (*input-dim*) in 2 (*hidden_size*)

```
# 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())
```

1 2

2 ↓

hidden_size=2



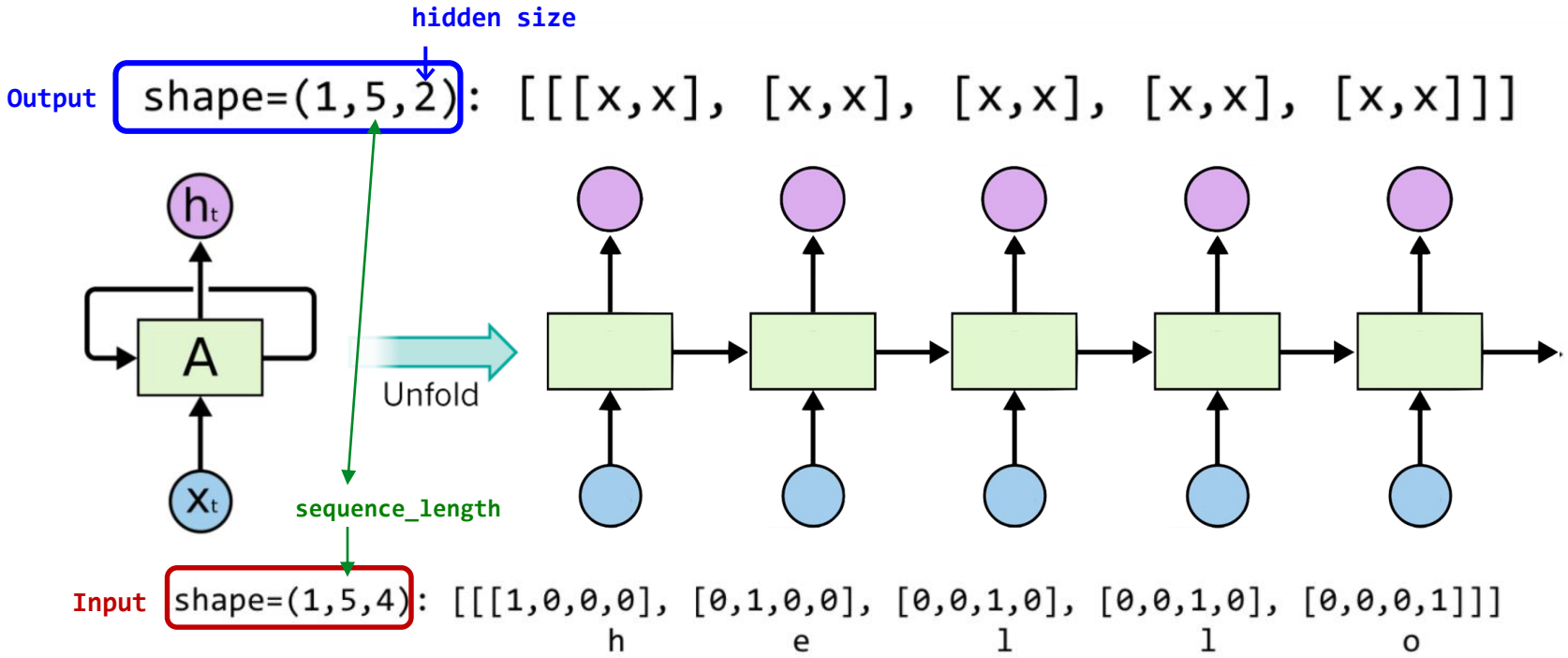
outputs

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

Output size = Hidden size

Unfolding to n sequences

hidden_size=2
sequence_length = 5

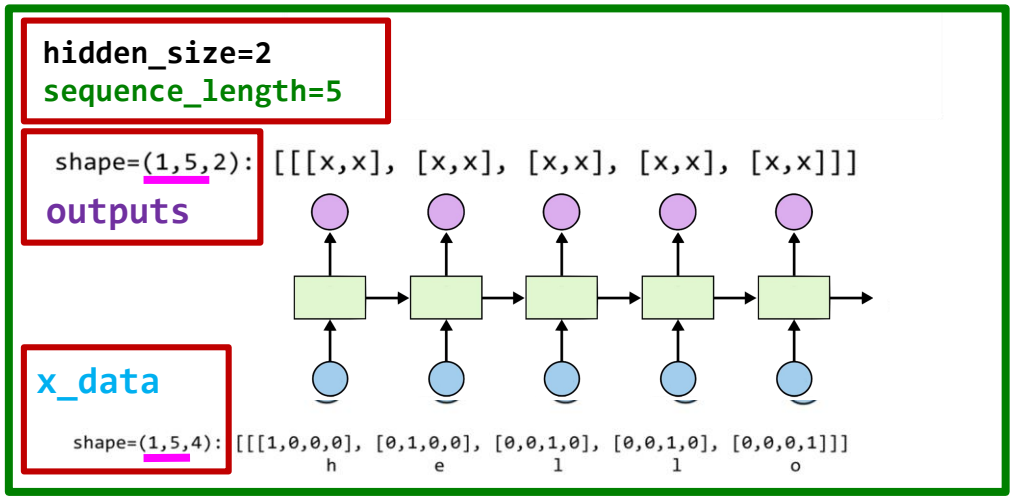


Unfolding to n sequences

```
# 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, l, l, 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())
```

5개이므로 sequence length = 5

```
# 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)

X_data

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

Outputs

```
hidden_size=2
sequence_length=5
batch_size=3
```

Batching input

hidden size

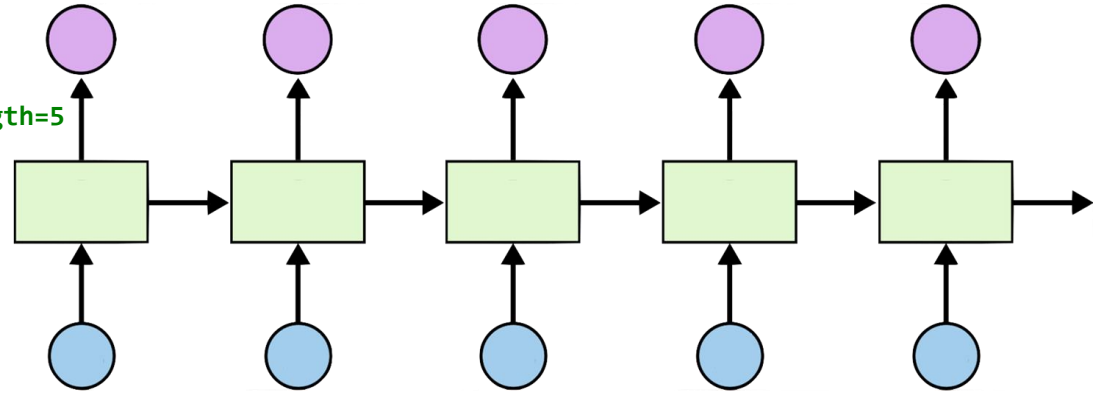
Output

```
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]]]
```

equals

sequence_length=5

Batch_size=3



Input

```
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)
```

One_hot_encoding

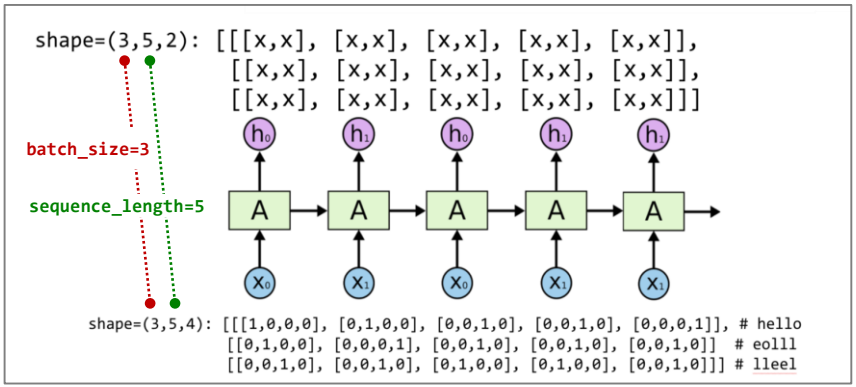
hidden_size

```
1 cell = rnn.BasicLSTMCell(num_units=2, state_is_tuple=True)
2 outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
sess.run(tf.global_variables_initializer())
pp.pprint(outputs.eval())
```

```
# 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.]],

       [[ 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.]]],
```




```
Hidden_size=2
sequence_length=5
batch_size=3
```

Batching output

```
# 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)

1 cell = rnn.BasicLSTMCell(num_units=2, state_is_tuple=True)
2 outputs, _states = tf.nn.dynamic_rnn(cell, x_data,
                                       dtype=tf.float32)
sess.run(tf.global_variables_initializer())
pp.pprint(outputs.eval())
```

hidden_size

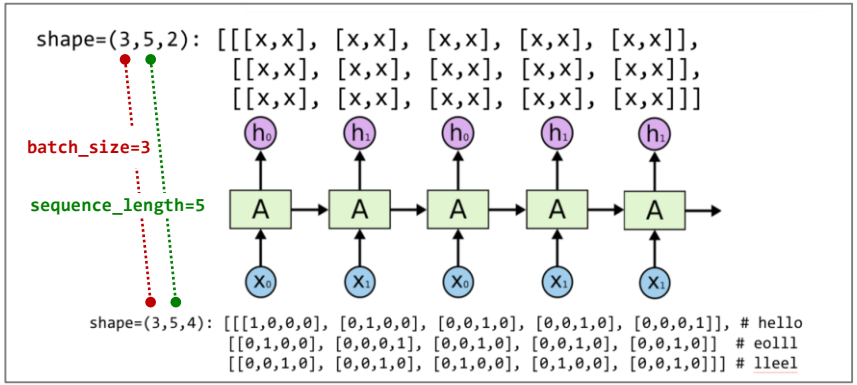
hidden size : 2

outputs

```
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]]],
```

x_data

```
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.]]],
```



ex12_1.ipynb

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
import pprint
tf.reset_default_graph()

pp = pprint.PrettyPrinter(indent=4)
# 배열등을 출력할 때 가독성을 높이기 위해 사용하는 방법으로
# \t 대신에 indentation을 임의의 숫자로 설정하여 예쁘게 프린트함
sess = tf.InteractiveSession()
```

```
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

```
========== one_cell ==========
cell.output_size :  2
cell.state_size :  2

x_data
array([[[1., 0., 0., 0.]]], dtype=float32)

outputs
array([[[-0.09269013,  0.44182047]]], dtype=float32)
```

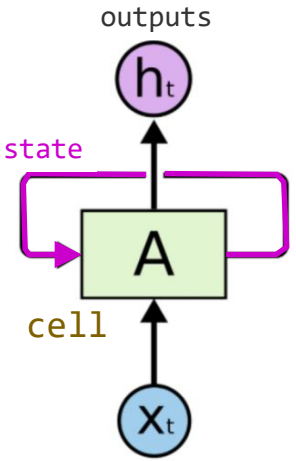
```
with tf.variable_scope('one_cell') as scope:
    # One cell RNN input_dim (4) -> output_dim (2)
    hidden_size = 2
1   cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)
    print("="*20, 'one_cell', "="*20)
    print("cell.output_size : ", cell.output_size) # output size 확인
    print( " cell.state_size :  ", cell.state_size) # state size 확인
    x_data = np.array([[h]], dtype=np.float32) # x_data = [[[1,0,0,0]]]
    print("\nx_data")
    pp.pprint(x_data)

2   outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
    sess.run(tf.global_variables_initializer())
    print("\noutputs")
    pp.pprint(outputs.eval())
```

cell 생성

cell 작동

hidden_size=2

sequence_length=1
batch_size=1



ex12_2.ipynb

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
import pprint
tf.reset_default_graph()

pp = pprint.PrettyPrinter(indent=4)

sess = tf.InteractiveSession()

h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]

with tf.variable_scope('two_dimension_sequences') as scope:
# One cell RNN input_dim (1,5,4) -> output_dim (2). sequence: 5
    hidden_size = 2
1   cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)
    x_data = np.array([[h, e, l, l, o]], dtype=np.float32)
    print("="*20, 'two_dimension_sequences', "="*20)
    print("x_data.shape", x_data.shape)
    print("\nx_data")
    pp.pprint(x_data)

2   outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
    sess.run(tf.global_variables_initializer())
    print("\noutputs")
    pp.pprint(outputs.eval())
```

```
================ two_dimension_sequences ================
x_data.shape (1, 5, 4)

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([[[-0.42810306, -0.61416245],
        [-0.4976399 , -0.574681  ],
        [ 0.45450127,  0.14287288],
        [ 0.28191647,  0.3143787 ],
        [ 0.59221447,  0.67575467]]], dtype=float32)
```

sequence_length=5

sequence_length=5

hidden_size=2

sequence_length=5
batch_size=1

ex12_3.ipynb

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
import pprint
tf.reset_default_graph()
pp = pprint.PrettyPrinter(indent=4)
sess = tf.InteractiveSession()
```

```
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

hidden_size=2

sequence_length=5
batch_size=3

```
with tf.variable_scope('3_batches') as scope:
# One cell RNN input_dim (3,5,4) -> 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)
    print("="*20, '3_batches', "="*20)
    print("x_data.shape", x_data.shape)
    print("\nx_data")
    pp.pprint(x_data)
    hidden_size = 2
1   cell = rnn.BasicLSTMCell(num_units=hidden_size,state_is_tuple=True)
2   outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
    sess.run(tf.global_variables_initializer())
    print("outputs.shape", outputs.shape)
    print("\noutputs")
    pp.pprint(outputs.eval())
```

batch_size=3

batch_size=3

```
==================== 3_batches ====================
x_data.shape (3, 5, 4)
x_data
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.]]], dtype=float32)
outputs.shape (3, 5, 2)
outputs
array([[[-0.04291508,  0.06511486],
        [-0.04251731,  0.14533596],
        [-0.08407226,  0.10489488],
        [-0.09450939,  0.09734165],
        [ 0.01109308,  0.05612568]],

       [[-0.02036933,  0.10128689],
        [ 0.06583957,  0.04163842],
        [ 0.01242916,  0.06397581],
        [-0.02108005,  0.07829609],
        [-0.04641416,  0.08495101]],

       [[-0.01978358,  0.04305524],
        [-0.04004388,  0.06581258],
        [-0.04333504,  0.14130065],
        [-0.06174098,  0.17141217],
        [-0.10792138,  0.11565945]]], dtype=float32)
```

sequence_length=5
batch_size=3

sequence_length=5
batch_size=3

Recurrent Neural Network

- We can process a sequence of vectors x by applying a recurrence formula at every time step:

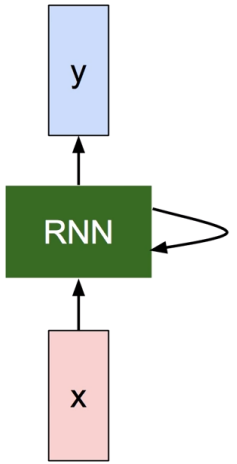
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

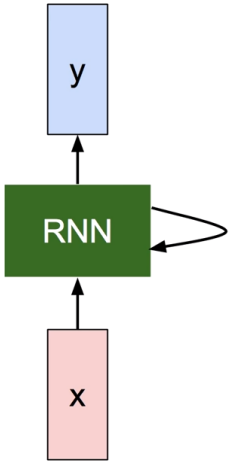
old state

input vector at some time step



(Vanilla) Recurrent Neural Network

- The state consists of a single **“hidden”** vector **h**:



$$h_t = f_w(h_t - 1, x_t)$$

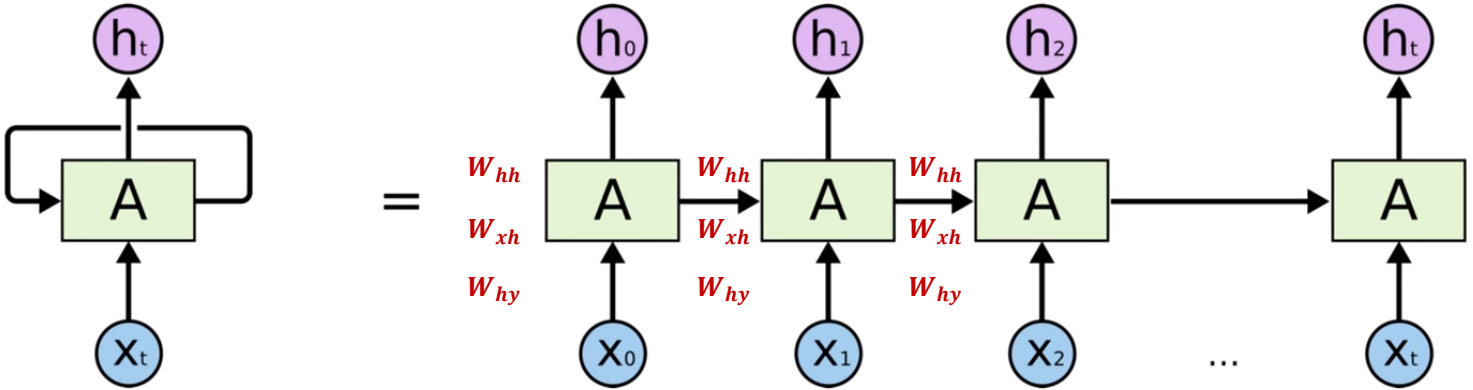
WX

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

(Vanilla) Recurrent Neural Network

- Notice : the same function and the same set of parameters are used at every time step.



ex12_4.ipynb (1/2)

hidden_size=5

sequence_length=5
batch_size=3

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
import pprint
tf.reset_default_graph()
pp = pprint.PrettyPrinter(indent=4)
sess = tf.InteractiveSession()

print("="*20, 'generated_data', "="*20)
```

```
# Create input data
batch_size=3
sequence_length=5
input_dim=3
```

45(0~44)개의 데이터

```
x_data = np.arange(45, dtype=np.float32).reshape(batch_size, sequence_length, input_dim)
print("x_data.shape", x_data.shape)
print("\nx_data")
pp.pprint(x_data)  # batch, sequence_length, input_dim
```

3 5 3

```
==================== generated_data ====================
x_data.shape (3, 5, 3)
x_data
array([[[ 0.,  1.,  2.],
        [ 3.,  4.,  5.],
        [ 6.,  7.,  8.],
        [ 9., 10., 11.],
        [12., 13., 14.]],

       [[15., 16., 17.],
        [18., 19., 20.],
        [21., 22., 23.],
        [24., 25., 26.],
        [27., 28., 29.]],

       [[30., 31., 32.],
        [33., 34., 35.],
        [36., 37., 38.],
        [39., 40., 41.],
        [42., 43., 44.]]], dtype=float32)
```

sequence_length=5
batch_size=3

ex12_4.ipynb (2/2)

```
with tf.variable_scope('generated_data') as scope:
    # One cell RNN input_dim (3, 5, 3) → output_dim (5). sequence: 5, batch: 3

1   cell = rnn.BasicLSTMCell(num_units=5, state_is_tuple=True)

    initial_state = cell.zero_state(batch_size, tf.float32)
    # cell.zero_state → RNN 의 맨 처음 상태(ht-1) 를 0 으로 초기화
    #( W_hh, W_xh , W_hy ) → batch_size만큼 0 으로 초기화한다는 뜻
2   outputs, _states = tf.nn.dynamic_rnn(cell, x_data,
                initial_state=initial_state, dtype=tf.float32)

    sess.run(tf.global_variables_initializer())
    print("outputs.shape", outputs.shape)
    print("\noutputs")
    pp.pprint(outputs.eval())
```

hidden_size

hidden_size=5

sequence_length=5
batch_size=3

```
outputs.shape (3, 5, 5)
outputs
array([[[-2.4606255e-01,  1.2046822e-01, -1.2883182e-02, -6.8152174e-02,
          1.9305602e-01],
        [-5.3959841e-01,  4.9687633e-01, -2.2469512e-01, -4.4553023e-02,
          1.9366221e-01],
        [-6.7132217e-01,  7.9844189e-01, -4.7681862e-01, -1.0833621e-02,
          1.1072669e-01],
        [-7.4149019e-01,  9.1981524e-01, -6.9001329e-01, -2.1427416e-03,
          5.4281782e-02],
        [-7.9746199e-01,  9.6612847e-01, -8.3010286e-01, -3.7983700e-04,
          2.3821915e-02]],

       [[-6.5224421e-01,  7.4955112e-01, -2.2399326e-01, -4.8006270e-05,
          6.4511434e-03],
        [-8.2909483e-01,  9.5862925e-01, -4.7308072e-01, -8.9959731e-06,
          4.0323581e-03],
        [-8.8927180e-01,  9.9282324e-01, -6.7521071e-01, -1.6080462e-06,
          1.7783853e-03],
        [-9.2214638e-01,  9.9837744e-01, -8.1134939e-01, -2.7749232e-07,
          7.2656822e-04],
        [-9.4335604e-01,  9.9950480e-01, -8.9335823e-01, -4.7115364e-08,
          2.9532250e-04]],

       [[-7.3779351e-01,  7.6147026e-01, -1.6769260e-01, -6.1012639e-09,
          7.8194695e-05],
        [-9.2635208e-01,  9.6396983e-01, -3.6434558e-01, -1.0969079e-09,
          4.8952381e-05],
        [-9.6640193e-01,  9.9503064e-01, -5.4219973e-01, -1.9279518e-10,
          2.1538010e-05],
        [-9.7949851e-01,  9.9931884e-01, -6.8236160e-01, -3.3257987e-11,
          8.8137458e-06],
        [-9.8594111e-01,  9.9990475e-01, -7.8419316e-01, -5.6825616e-12,
          3.5839121e-06]]], dtype=float32)
```

Cost: sequence_loss

if) sequence_length가 3이고, y_data가 다음과 같다면…

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

if) prediction이 다음과 같다면…

```
# prediction : one-hot → (1, 0) 또는 (0,1) 중에 하나가 출력된다.
# 0.7, 0.6, 0.9  → one-hot : (0, 1), (1, 0), (0, 1)
```

1 0 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)
```

if) weights이 다음과 같다면…

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

weights :
prediction 각각의 값의 비중
[[1,1,1]]

sequence_length 개수 만큼

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

Loss: 0.596759

ex12_5.ipynb

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
import pprint
tf.reset_default_graph()
sess = tf.InteractiveSession()

# [batch_size, sequence_length]
y_data = tf.constant([[1, 1, 1]])
print("y_data : ", y_data.eval(), "\n")

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

# 아래의 tf.contrib.seq2seq.sequence_loss 에서 logits 부분은 Wx+b 형태로 입력한다.

print("prediction \n", prediction.eval(), "\n")

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


print("weight : ", weights.eval())

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())
```

1 0 1

y_data : [[1 1 1]]

prediction
[[[0.2 0.7]
[0.6 0.2]
[0.2 0.9]]]

weight : [[1. 1. 1.]]
Loss: **0.5967595**

weights :
prediction 각각의 값의 비중
[[1,1,1]]

Hypothesis에 해당

True value Y 에 해당

sequence_length 개수 만큼

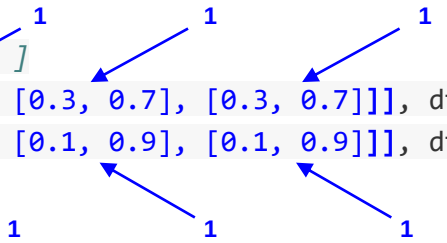
Multi Cost: sequence_loss

if) sequence_length가 3이고, y_data가 다음과 같다면…

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

Loss1: 0.513015
Loss2: 0.371101

if) prediction이 다음과 같다면…




```
# [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)
```

if) weights이 다음과 같다면…

```
# [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())
```

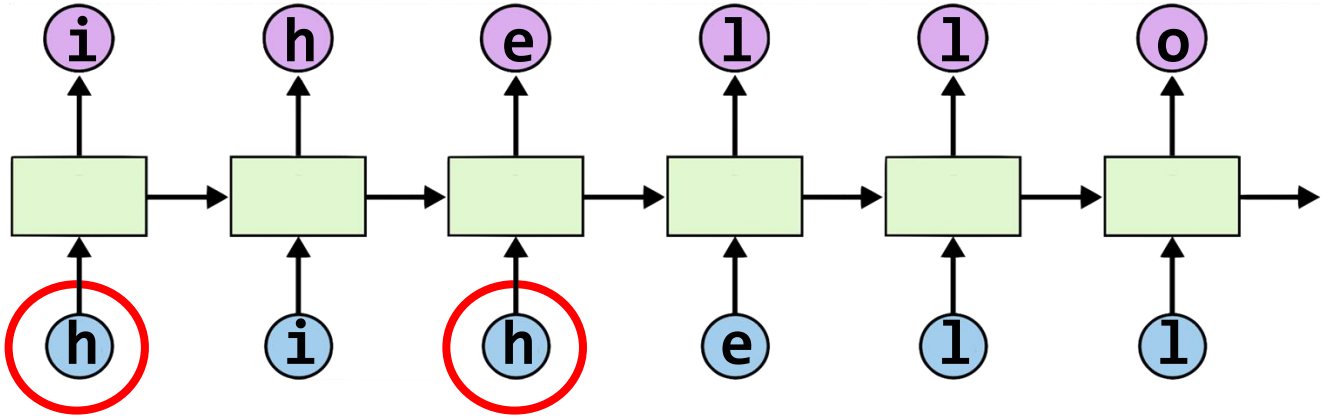
ex12_6.ipynb

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
import pprint
tf.reset_default_graph()
sess = tf.InteractiveSession()

# [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)
prediction3 = tf.constant([[ [1, 0], [1, 0], [1, 0] ]], dtype=tf.float32)
prediction4 = tf.constant([[ [0, 1], [1, 0], [0, 1] ]], 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)
sequence_loss3 = tf.contrib.seq2seq.sequence_loss(prediction3, y_data, weights)
sequence_loss4 = tf.contrib.seq2seq.sequence_loss(prediction3, y_data, weights)
sess.run(tf.global_variables_initializer())
print( "Loss1: ", sequence_loss1.eval() )
print( "Loss2: ", sequence_loss2.eval() )
print( "Loss3: ", sequence_loss3.eval() )
print( "Loss4: ", sequence_loss4.eval() )
```

```
Loss1:  0.5130153
Loss2:  0.3711007
Loss3:  1.3132616
Loss4:  1.3132616
```

2 Hi Hello **RNN**

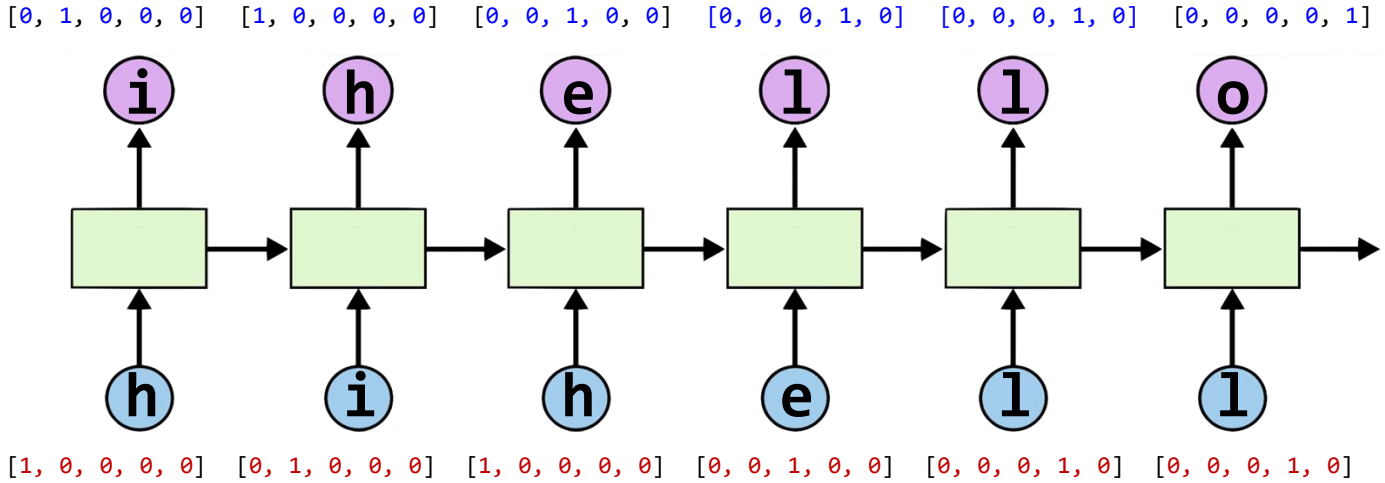
Teach RNN ‘hihello’



Teach RNN 'hihello'

```
input_dim = 5
output_dim(hidden_size) = 5
sequence_length = 6
batch_size = 1
```

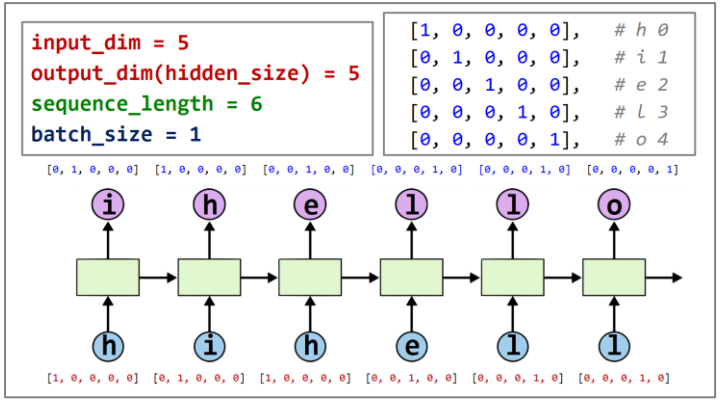
```
[1, 0, 0, 0, 0],   # h 0
[0, 1, 0, 0, 0],   # i 1
[0, 0, 1, 0, 0],   # e 2
[0, 0, 0, 1, 0],   # l 3
[0, 0, 0, 0, 1],   # o 4
```

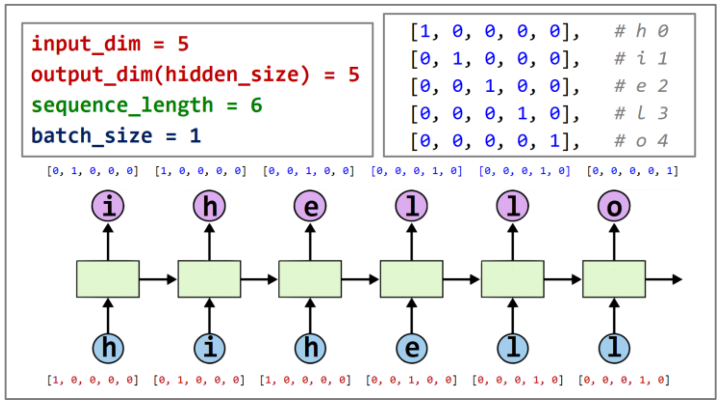


1 Creating rnn cell

output_dim (hidden size) → 5 이므로 cell size는 5 이다.

```
# 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

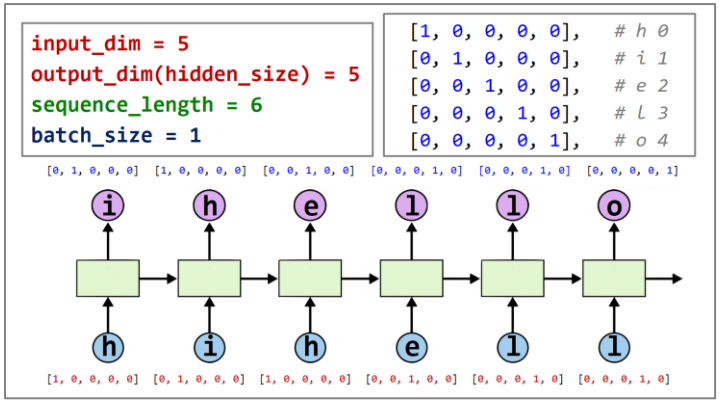
hidden_size = 5

input_dim = 5
batch_size = 1
sequence_length = 6

```
# RNN model
1 rnn_cell = rnn_cell.BasicRNNCell(rnn_size)
2 outputs, _states = tf.nn.dynamic_rnn(rnn_cell,X,initial_state=initial_state, dtype=tf.float32)
```

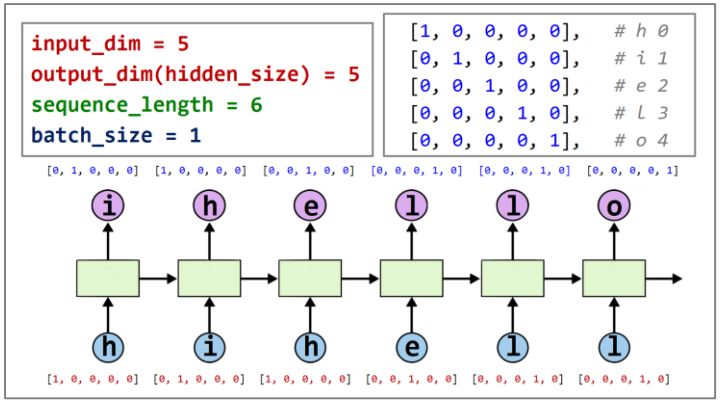
RNN parameters

```
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
```



Feed to RNN

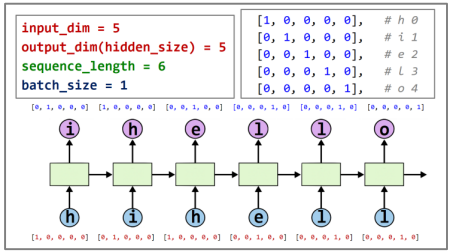
```
x_one_hot = [[[1, 0, 0, 0, 0],   # h 0
              [0, 1, 0, 0, 0],   # i 1
              [1, 0, 0, 0, 0],   # h 0
              [0, 0, 1, 0, 0],   # e 2
              [0, 0, 0, 1, 0],   # l 3
              [0, 0, 0, 1, 0]]]  # l 3

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

batch_size 6 5

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

batch_size 6

5 *(one_hot_encoding)*



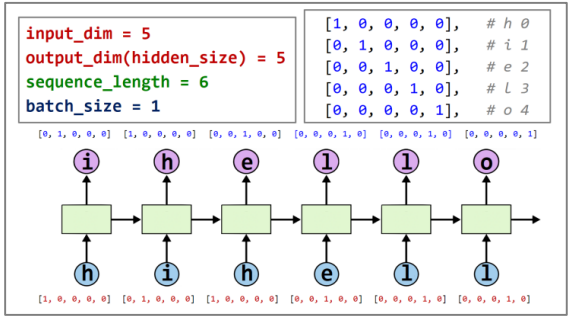
1. `cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size, state_is_tuple=True)`
   `initial_state = cell.zero_state(batch_size, tf.float32)`
2. `outputs, _states = tf.nn.dynamic_rnn(cell, X, initial_state=initial_state, dtype=tf.float32)`

ex12_7(hihello).ipynb (1/2)

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # reproducibility
# 다시 실행할 때 error가 발생하면 : Kernel → Interrupt → Restart and Clear output 후에 다시 실행한다.
```

A Data creation

```
idx2char = ['h', 'i', 'e', 'l', 'o']
# Teach hello: hihell → ihello

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
```




```
num_classes = 5  # hidden_size와 동일 (fully_connected에 사용됨)
input_dim = 5  # (input_dim == hidden_size) one-hot size
hidden_size = 5  # (input_dim == hidden_size) output from the LSTM. 5 to directly predict one-hot
batch_size = 1   # one sentence
sequence_length = 6  # |ihello| == 6

learning_rate = 0.1
X = tf.placeholder(tf.float32, [None, sequence_length, input_dim])  # x_one_hot → shape (1, 6, 5)
Y = tf.placeholder(tf.int32, [None, sequence_length])  # y_data shape→ (1, 6)

1 cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size, state_is_tuple=True)  ← cell 생성
initial_state = cell.zero_state(batch_size, tf.float32)
2 outputs, _states = tf.nn.dynamic_rnn(cell, X, initial_state=initial_state, dtype=tf.float32)  ← cell 작동
# outputs shape→ (batch_size, sequence_length, hidden_size) → (1, 6, 5)
```

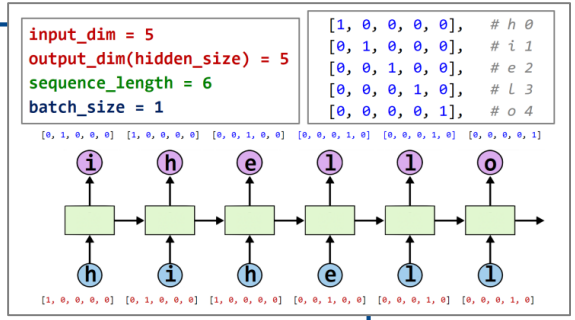
ex12_7(hihello).ipynb (2/2)

```
# outputs shape→ (batch_size, sequence_length, hidden_size) → (1, 6, 5)
# FC Layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size]) # hidden_size 5 → shape(6, 5)
# fc_w = tf.get_variable("fc_w", [hidden_size, num_classes]) # fc_w shape : (5,5)
# fc_b = tf.get_variable("fc_b", [num_classes])  # fc_b shape : (5)
# outputs = tf.matmul(X_for_fc, fc_w) + fc_b  # Wx+b 연산 → tf.matmul(X_for_fc, fc_w) + fc_b
outputs = tf.contrib.layers.fully_connected(inputs=X_for_fc, num_outputs=num_classes, activation_fn=None)
# Wx+b → x : x_for_fc, W : fully_connectd에서 제공 (num_outputs=num_classes → weight에 사용될 행의 개수만 입력)
# reshape out for sequence_loss
outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])  # outputs shape : (1, 6, 5)
weights = tf.ones([batch_size, sequence_length]) # weights shape : (1, 6) → 모두 1로 셋팅 : 출력값 6개 모두 비중을 1로 함
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=learning_rate).minimize(loss)
prediction = tf.argmax(outputs, axis=2)
```

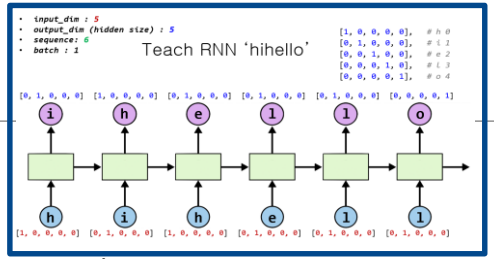
weights = [[1,1,1,1,1,1]]

B

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # print("X_for_fc \n", sess.run(X_for_fc)
    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)
        # print char using dic
        result_str = [idx2char[c] for c in np.squeeze(result)]
        # np.squeeze(result)→ 2차원의 result를 1차원으로 변경해서 c 에 할당한 후 idx2char의 인덱스로 사용하여 문자로 변환함
        print("\tPrediction str: ", ''.join(result_str) )
```



ex12_7(hihello).ipynb (Notes)




```
0 loss: 1.6078763 prediction:  [[3 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  llllll
1 loss: 1.5102623 prediction:  [[3 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  llllll
2 loss: 1.4327028 prediction:  [[3 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  llllll
3 loss: 1.3489527 prediction:  [[3 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  llllll
4 loss: 1.2551297 prediction:  [[1 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  illlll
5 loss: 1.140437 prediction:  [[1 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  illlll
6 loss: 1.0167552 prediction:  [[1 3 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  ilello
7 loss: 0.8969265 prediction:  [[1 3 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  ilello
                .
                .
                .
9995 loss: 5.960464e-08 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  ihello
9996 loss: 5.960464e-08 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  ihello
9997 loss: 5.960464e-08 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  ihello
9998 loss: 5.960464e-08 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  ihello
9999 loss: 5.960464e-08 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
                Prediction str:  ihello
```

Data creation

text: ‘hihello’

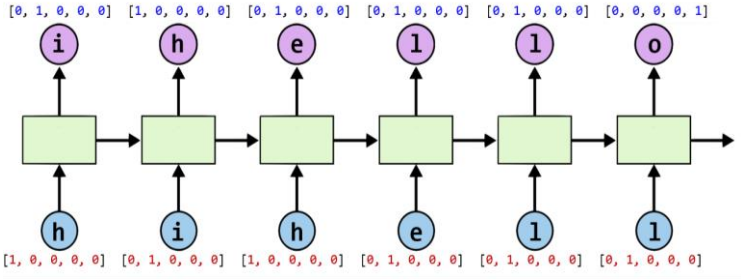
unique chars (vocabulary, voc):
h, i, e, l, o

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

```
x_data = [[0, 1, 0, 2, 3, 3]]   # hihell
```

```
y_data = [[1, 0, 2, 3, 3, 4]]
```

```
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
```



Axis

```
prediction

      [[1 0 1 1 1 4]]
```

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

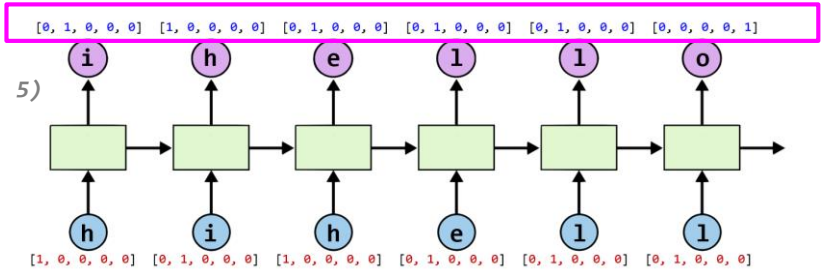
axis의 개수는 rank의 값과 동일합니다.
axis를 카운트하는 방식은 배열에서 가장 바깥쪽 덩이를 시작으로 0부터 카운트 합니다.

```
[
    [
        [1,2,3],[4,5,6]
    ],
    [
        [7,8,9],[10,11,12]
    ]
]
```

와 같은 형태를 가질텐데, 이때 빨간색 부분이 axis = 0 이고, 파란색 부분이 axis = 1, 검은색 부분이 axis = 2 입니다.
또한 가장 안쪽에 있는 axis 는 -1로 표현하기도 합니다.

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

```
outputs →
# outputs shape : (1, 6, 5)
```



3 RNN with long sequences

Better data creation

숫자 알파벳

```
sample = " if you want you"
idx2char = list(set(sample))  # set type : unique data creation
char2idx = {c: i for i, c in enumerate(idx2char)}  # dictonary type : {Key : value}
```

{알파벳:숫자 } # {i:0, f:1, y:2 . . . }

```
# 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
```

Better data creation

숫자 알파벳

```
sample = " if you want you"
idx2char = list(set(sample))  # set type : unique data creation
char2idx = {c: i for i, c in enumerate(idx2char)}  # dictonary type : {Key : value}
```

{알파벳:숫자 } # {i:0, f:1, y:2, ... }

```
sample_idx = [char2idx[c] for c in sample]  # char to index
x_data = [sample_idx[:-1]]  # X data sample (0 ~ n-1) → e.g.) ifyouwantyou: if you want yo
y_data = [sample_idx[1:]]   # Y label sample (1 ~ n) → e.g.)ifyouwantyou: f you want you
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
                                        # idx2char의 size와 같다
                                        # final output size (RNN or softmax, etc.)
```

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.AdamOptimizer(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: nnuffuunnuuuyuy
1 loss: 2.29675 Prediction: nnuffuunnuuuyuy
...

1418 loss: 1.37351 Prediction: if you want you
1419 loss: 1.37331 Prediction: if you want you
```

ex12_8.ipynb (1/2)

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

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)
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
learning_rate = 0.1

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
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_one_hot, initial_state=initial_state, dtype=tf.float32)
```

ex12_8.ipynb (2/2)

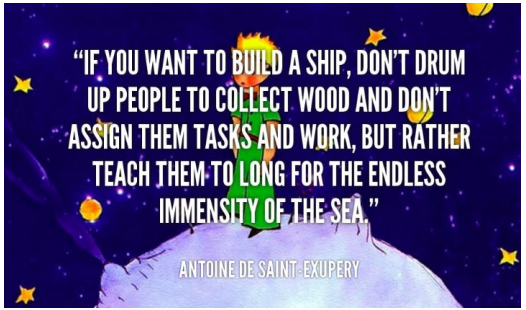
```
# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])
outputs = tf.contrib.layers.fully_connected(X_for_fc, num_classes, activation_fn=None)

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

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=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(50):
        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))
```

ex12_8.ipynb (Notes)

```
0 loss: 2.295272 Prediction: u
1 loss: 2.1581483 Prediction: y    u        u
2 loss: 1.9606304 Prediction: y   uu u    uuu
3 loss: 1.7423404 Prediction: y   ou   n  you
4 loss: 1.459033 Prediction: y  you  ann you
5 loss: 1.1581826 Prediction: yf you uann you
6 loss: 0.89342517 Prediction: yf you want you
7 loss: 0.64734656 Prediction: yf you want you
8 loss: 0.45212293 Prediction: yf you want you
9 loss: 0.31170127 Prediction: if you want you
10 loss: 0.2121459 Prediction: if you want you
.
.
.

43 loss: 0.00072751974 Prediction: if you want you
44 loss: 0.00069658685 Prediction: if you want you
45 loss: 0.00066906714 Prediction: if you want you
46 loss: 0.00064458756 Prediction: if you want you
47 loss: 0.00062259275 Prediction: if you want you
48 loss: 0.00060281286 Prediction: if you want you
49 loss: 0.0005849543 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.

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 for the parsing

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.
```

dataX dataY

169

next slide

Making dataset

```
char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}
           {알파벳:숫자 } # {i:0, f:1, y:2 . . . }

dataX = []
dataY = []
```

Window 크기

```
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)
```

```
# 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.
```

dataX dataY

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=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.AdamOptimizer(learning_rate=0.1).minimize(loss)

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

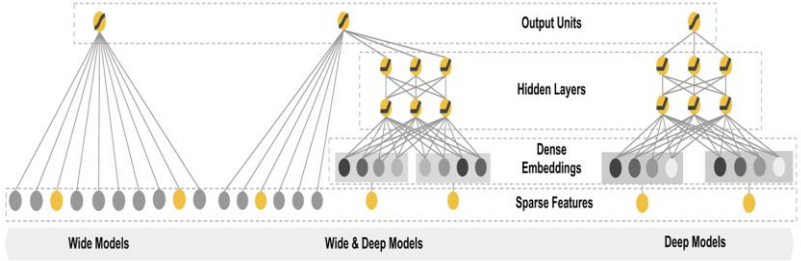
Why it does not work?

- Logit ?
- Deep RNN ?

4

RNN with long sequences: Stacked RNN + Softmax layer

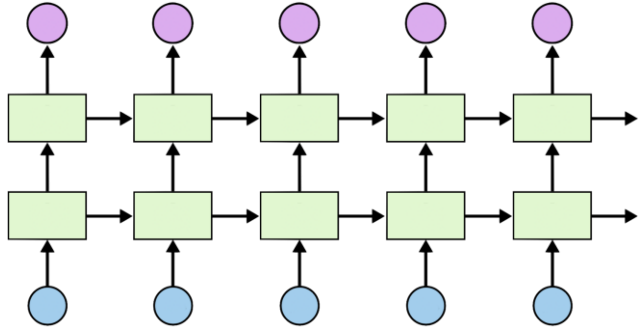
Wide & Deep




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

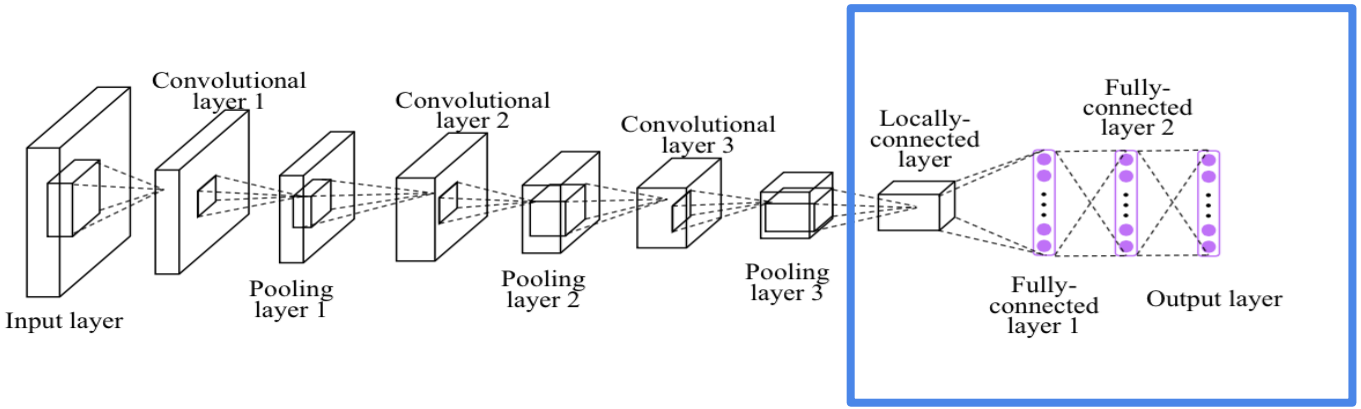
```
data_dim = len(char_set)
hidden_size = len(char_set)
num_classes = len(char_set)
seq_length = 10   # Any arbitrary number for the parsing

batch_size = len(dataX)
```

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



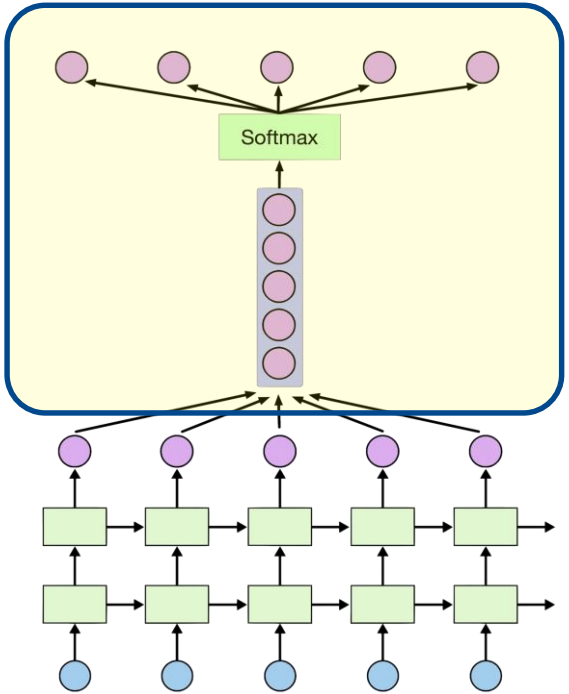
Softmax

RNN에서 나온 outputs

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

   `new_outputs = X_for_softmax * W + b`

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



Softmax

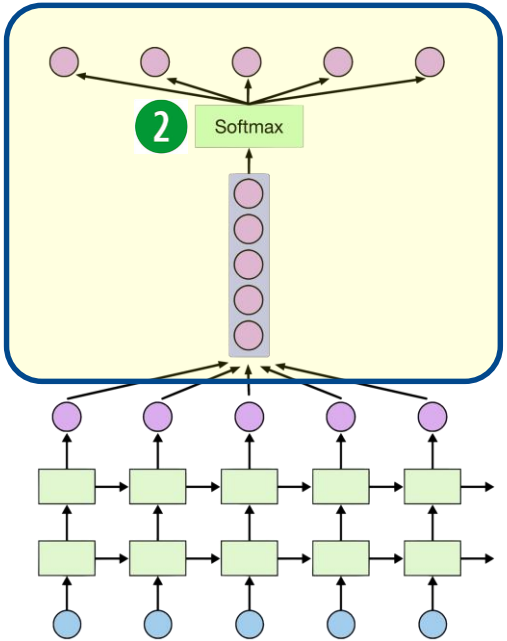
```
# (optional) softmax layer
X_for_softmax = tf.reshape(outputs, [-1, hidden_size])
softmax_w = tf.get_variable("softmax_w", [hidden_size, num_classes]
softmax_b = tf.get_variable("softmax_b",[num_classes])
outputs = tf.matmul(X_for_softmax,softmax_w) + softmax_b
outputs = tf.reshape(outputs,[batch_size, seq_length, num_classes])
```

RNN에서 나온 outpus

1

Wx + b

2



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): # iteration → 문장이 길어서 500번 정도는 반복해야 함
    _, meloss, 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]), meloss)
```

```
0 0 yyyyspppppp 3.216124
0 1 sssspppppt 3.216124
0 2 gettppttttt 3.216124
0 3 nn.yptttte 3.216124
0 4 nhpppttttet 3.216124
0 5 yypppppeeee 3.216124
0 6 ggtttteeee 3.216124.
.
.
0 165 yyyysssssss 3.216124
0 166 yyasssssss 3.216124
0 167 natnssssss 3.216124
0 168 guuuusspppp 3.216124
0 169 niussnssss 3.216124
.
.
.

499 0 g you want 0.22875121
499 1 oyou want  0.22875121
499 2 tou want t 0.22875121
499 3 ou want to 0.22875121
499 4  want to  0.22875121
499 5 mwant to b 0.22875121
.
.
499 165 gy of the  0.22875121
499 166 h of the s 0.22875121
499 167 oof the se 0.22875121
499 168 tf the sea 0.22875121
499 169  the sea. 0.22875121
```

Training and print results

```
data_dim = len(char_set)
hidden_size = len(char_set)
num_classes = len(char_set)
seq_length = 10  # Any arbitrary number for the parsing

batch_size = len(dataX)
```

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

```
# 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):   # result → results가 seq_length단위 (10행)로 분리됨
    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='') #index에는 10개씩
    else:
        print(char_set[index[-1]], end='')
```

f 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.

ex12_9.ipynb (1/3)

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
tf.set_random_seed(777)  # reproducibility
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)
```

ex12_9.ipynb (2/3)

```
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)

# 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])
```

ex12_9.ipynb (3/3)

```
# 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='')
      # print("★",j,''.join([char_set[t] for t in index]), end='')
    else:
        print(char_set[index[-1]], end='')
      # print("●",j, char_set[index[-1]], end='')
```

```
★ 0 p you want● 1  ● 2 t● 3 o● 4  ● 5 b● 6 u● 7 i● 8 l● 9 d● 10  ● 11 a● 12  ●
13 s● 14 h● 15 i● 16 p● 17 ,● 18  ● 19 d● 20 o● 21 n● 22 '● 23 t● 24  ● 25 d● 26
r● 27 u● 28 m● 29  ● 30 u● 31 p● 32  ● 33 p● 34 e● 35 o● 36 p● 37 l● 38 e● 39  ●
40 t● 41 o● 42 g● 43 e● 44 t● 45 h● 46 e● 47 r● 48  ● 49 t● 50 o● 51  ● 52 c● 53
o● 54 l● 55 l● 56 e● 57 c● 58 t● 59  ● 60 w● 61 o● 62 o● 63 d● 64  ● 65 a● 66 n●
67 d● 68  ● 69 d● 70 o● 71 n● 72 '● 73 t● 74  ● 75 a● 76 s● 77 s● 78 i● 79 g● 80
n● 81  ● 82 t● 83 h● 84 e● 85 m● 86  ● 87 t● 88 a● 89 s● 90 k● 91 s● 92  ● 93 a●
94 n● 95 d● 96  ● 97 w● 98 o● 99 r● 100 k● 101 ,● 102  ● 103 b● 104 u● 105 t● 106
● 107 r● 108 a● 109 t● 110 h● 111 e● 112 r● 113  ● 114 t● 115 e● 116 a● 117 c●
118 h● 119  ● 120 t● 121 h● 122 e● 123 m● 124  ● 125 t● 126 o● 127  ● 128 l● 129
o● 130 n● 131 g● 132  ● 133 f● 134 o● 135 r● 136  ● 137 t● 138 h● 139 e● 140  ●
141 e● 142 n● 143 d● 144 l● 145 e● 146 s● 147 s● 148  ● 149 i● 150 m● 151 m● 152
e● 153 n● 154 s● 155 i● 156 t● 157 y● 158  ● 159 o● 160 f● 161  ● 162 t● 163 h●
164 e● 165  ● 166 s● 167 e● 168 a● 169 .
```

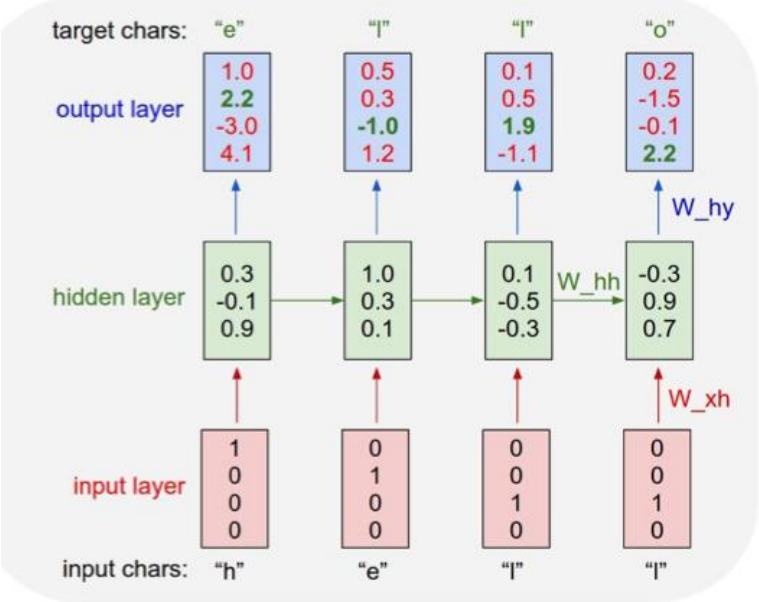
ex12_9.ipynb (Notes)

```
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
4 ou want to -> u want to
5 u want to  ->  want to b
6  want to b -> want to bu
7 want to bu -> ant to bui
8 ant to bui -> nt to buil
.
.
.
163 nsity of t -> sity of th
164 sity of th -> ity of the
165 ity of the -> ty of the
166 ty of the  -> y of the s
167 y of the s ->  of the se
168  of the se -> of the sea
169 of the sea -> f the sea.

Tensor("one_hot:0", shape=(?,
10, 25), dtype=float32)
```

```
0 0 yyyyspppppp 3.216124
0 1 sssspppptt 3.216124
0 2 gettpptttt 3.216124
0 3 nn.ypttttе 3.216124
0 4 nhpppttttet 3.216124
0 5 yypppppeeе 3.216124
0 6 ggtttteeee 3.216124.
.
.
0 165 yyyysssssss 3.216124
0 166 yyasssssss 3.216124
0 167 natnssssss 3.216124
0 168 guuuusspppp 3.216124
0 169 niussnssss 3.216124
.
.
.

499 0 g you want 0.22875121
499 1 oyou want  0.22875121
499 2 tou want t 0.22875121
499 3 ou want to 0.22875121
499 4   want to  0.22875121
499 5 mwant to b 0.22875121
.
.
499 165 gy of the  0.22875121
499 166 h of the s 0.22875121
499 167 oof the se 0.22875121
499 168 tf the sea 0.22875121
499 169   the sea. 0.22875121
```

```
m 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.
```

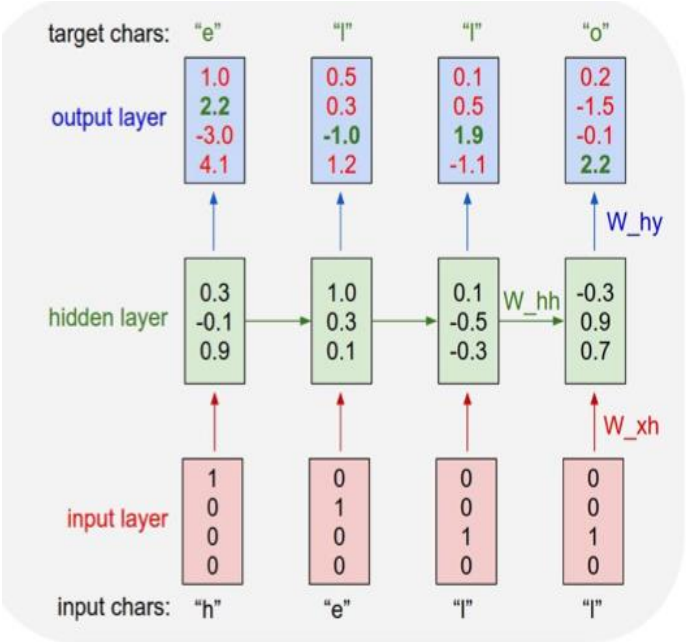
char-rnn



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 srain 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 nues 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;
}
```

5 RNN with time series data (stock)

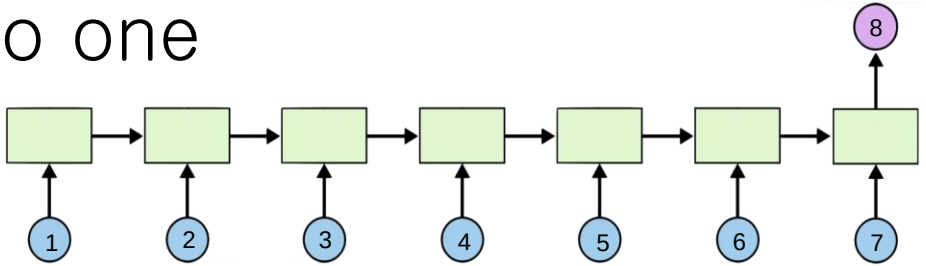
Time series data

June

day	Open (price)	High (price)	Low (price)	Volume (count)	Close (price)
1	828.659973	833.450012	828.349976	1247700	**831.659973**
2	823.02002	828.070007	821.655029	1597800	**828.070007**
3	819.929993	824.400024	818.97998	1281700	**824.159973**
4	819.359985	823	818.469971	1304000	**818.97998**
5	819	823	816	1053600	**820.450012**
6	816	820.958984	815.48999	1198100	**819.23999**
7	811.700012	815.25	809.780029	1129100	**813.669983**
8	809.51001	810.659973	804.539978	989700	**809.559998**
9	807	811.840027	803.190002	1155300	**808.380005**

'data-02-stock_daily.csv'

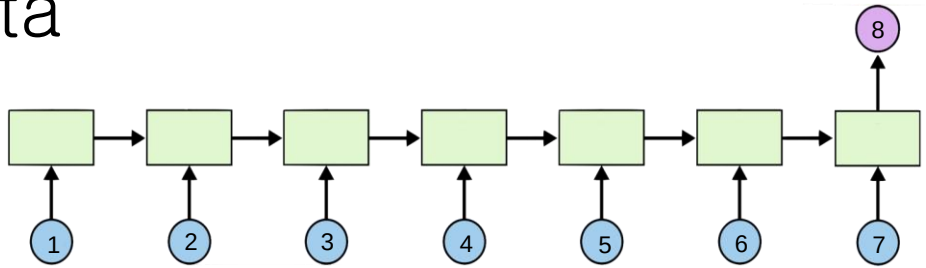
Many to one



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	?

```
input_dim = 5
output_dim (hidden_size) = 1
seq_length = 7
```

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) # normalization
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)
```

seq_length = 7

data_dim = 5

```
[ 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]
```

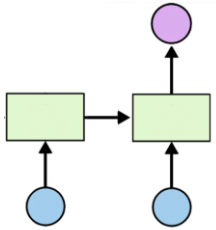
Training and test datasets

```
# split to train and testing
train_size = int(len(dataY) * 0.7) # 70% train
test_size = len(dataY) - train_size # 30% test

trainX = np.array(dataX[0:train_size])
trainY = np.array(dataY[0:train_size])

testX = np.array(dataX[train_size:len(dataX)])
testY = 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])
```

seq_length = 7

data_dim = 5

LSTM and Loss

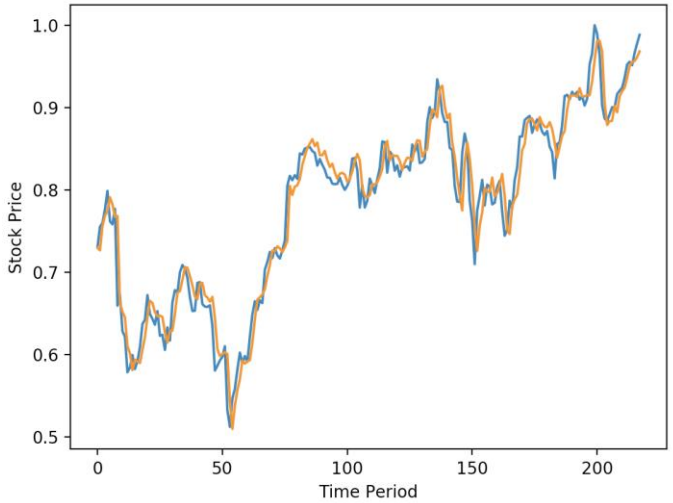


```
# input placeholders
X = tf.placeholder(tf.float32, [None, seq_length, data_dim])
Y = tf.placeholder(tf.float32, [None, 1])

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)
```

1

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()          # true value
```



ex12_10(stock).ipynb (1/3)

```
import tensorflow as tf
import numpy as np
import matplotlib
import os
tf.set_random_seed(777)  # reproducibility

def MinMaxScaler(data):
    numerator = data - np.min(data, 0)
    denominator = np.max(data, 0) - np.min(data, 0)
    # noise term prevents the zero division
    return numerator / (denominator + 1e-7)

# train Parameters
seq_length = 7
data_dim = 5
hidden_dim = 10
output_dim = 1
learning_rate = 0.01
iterations = 500

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

ex12_10(stock).ipynb (2/3)

```
# build a dataset
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)
# train/test split
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 place holders
X = tf.placeholder(tf.float32, [None, seq_length, data_dim])
Y = tf.placeholder(tf.float32, [None, 1])

# build a LSTM network
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_dim, state_is_tuple=True, activation=tf.tanh)
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(learning_rate)
train = optimizer.minimize(loss)
```

ex12_10(stock).ipynb (3/3)

```
targets = tf.placeholder(tf.float32, [None, 1])
predictions = tf.placeholder(tf.float32, [None, 1])
rmse = tf.sqrt(tf.reduce_mean(tf.square(targets - predictions)))

with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)

    # Training step
    for i in range(iterations):
        _, step_loss = sess.run([train, loss], feed_dict={X: trainX, Y: trainY})
        print("[step: {}] loss: {}".format(i, step_loss))

    # Test step
    test_predict = sess.run(Y_pred, feed_dict={X: testX})
    rmse_val = sess.run(rmse, feed_dict={targets: testY, predictions: test_predict})
    print("RMSE: {}".format(rmse_val))
```

ex12_10(stock).ipynb (Notes)

```
[[2.13751054e-01 2.08179810e-01 1.91791832e-01 4.66075110e-04
  1.92092403e-01]
 [1.93935034e-01 2.03641926e-01 2.08664571e-01 2.98467330e-03
  1.96551555e-01]
 [2.10516454e-01 2.05289413e-01 2.03558748e-01 2.59926504e-04
  1.87749731e-01]
 [1.86678765e-01 2.09480567e-01 2.08781843e-01 0.00000000e+00
  2.17448151e-01]
 [3.06973882e-01 3.14634137e-01 2.18993665e-01 1.24764722e-02
  2.16981885e-01]
 [2.19142110e-01 2.63907214e-01 2.24686396e-01 4.56323384e-01
  2.24967473e-01]
 [2.33129931e-01 2.36419163e-01 1.62682724e-01 5.70171193e-01
  1.47442742e-01]] -> [0.13289962]
.
.
.

[[0.91021623 0.91296982 0.92617114 0.10284127 0.92046468]
 [0.91753068 0.90955899 0.93013248 0.08799857 0.92390372]
 [0.92391259 0.92282604 0.94550876 0.10049296 0.93588207]
 [0.93644323 0.93932734 0.96226394 0.10667742 0.95211558]
 [0.94518557 0.94522671 0.96376051 0.09372591 0.95564213]
 [0.9462346  0.94522671 0.97100833 0.11616922 0.9513578 ]
 [0.94789567 0.94927335 0.97250489 0.11417048 0.96645463]] ->
[0.97785024]
[[0.91753068 0.90955899 0.93013248 0.08799857 0.92390372]
 [0.92391259 0.92282604 0.94550876 0.10049296 0.93588207]
 [0.93644323 0.93932734 0.96226394 0.10667742 0.95211558]
 [0.94518557 0.94522671 0.96376051 0.09372591 0.95564213]
 [0.9462346  0.94522671 0.97100833 0.11616922 0.9513578 ]
 [0.94789567 0.94927335 0.97250489 0.11417048 0.96645463]
 [0.95690035 0.95988111 0.9803545  0.14250246 0.97785024]] ->
[0.98831302]
```

```
[step: 0] loss: 161.7401580810547
[step: 1] loss: 102.94868469238281
[step: 2] loss: 60.83835220336914
[step: 3] loss: 32.31366729736328
[step: 4] loss: 15.422471046447754
[step: 5] loss: 8.562556266784668
.
.
.

[step: 489] loss: 0.4726698398590088
[step: 490] loss: 0.4721939265727997
[step: 491] loss: 0.4717211425304413
[step: 492] loss: 0.4712508022785187
[step: 493] loss: 0.4707837402820587
[step: 494] loss: 0.4703194499015808
[step: 495] loss: 0.4698580503463745
[step: 496] loss: 0.46939945220947266
[step: 497] loss: 0.4689437448978424
[step: 498] loss: 0.46849092841148376
[step: 499] loss: 0.4680408537387848
RMSE: 0.02595146745443344
```