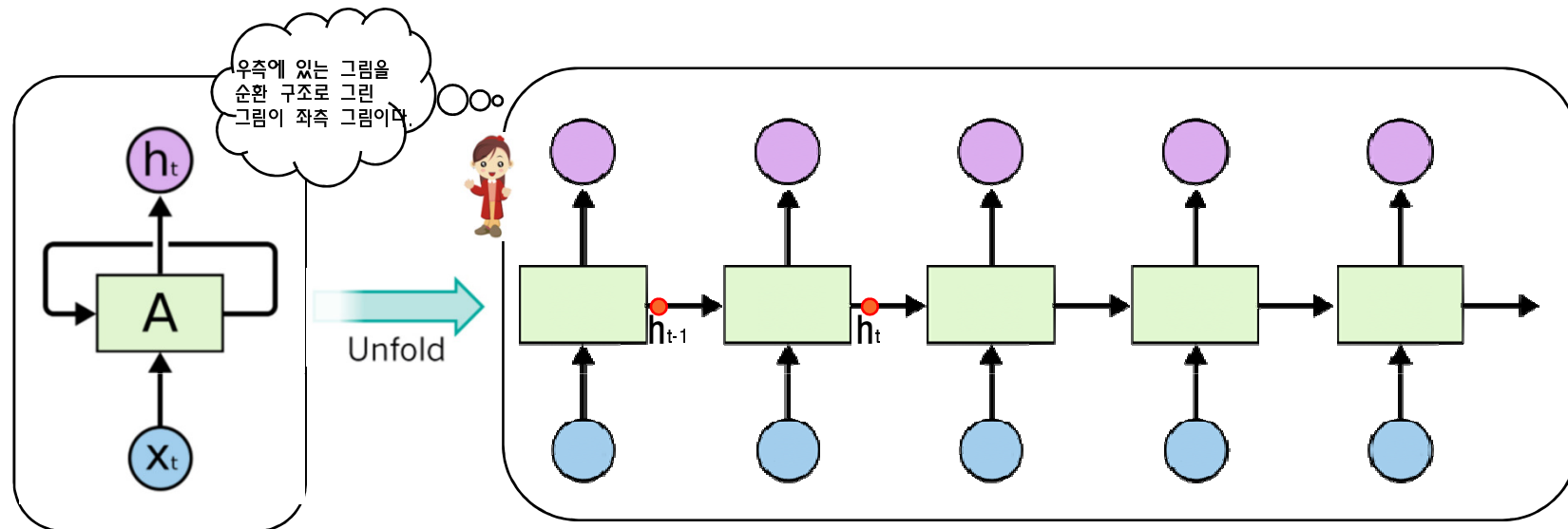


RNN(Recurrent Neural Network)

- **Sequence data**란 연속적인 데이터를 말한다.
- 예를 들어서 음성 인식, 자연어 처리 등이 이에 해당한다.
- Sequence data의 특징
 - We don't understand one word only
 - 우리는 단어 1개만 가지고 이해할 수 없다.
 - 예를 들어 "야구"라고 하면 야구를 한다는 건지, 좋아한다는 건지, 관람하러 가는 건지 알 수 없다.
 - "나는 지금 야구 시청을 한다."라고 한다면 "텔레비전을 보고 있구나"라고 생각할 수 있다.
 - We understand based on the previous words + this word. (**time series**)
 - 이전 단어를 이해하고 있어야 지금 이 단어도 이해가 된다.
 - NN/CNN cannot do this
 - 기존의 인공 신경망과 CNN에서는 불가능했다.

RNN(순환 신경망)

- Recurrent(되풀이되는, 반복되는) Neural Network
- 이전 상태(old state)가 다음 상태(new state)에 영향을 미친다.



t_t : 시간(time)

h_{t-1} : 이전 상태(old state)

X_t : input vector(입력하는 데이터)

h_t : 새로운 상태(new state)

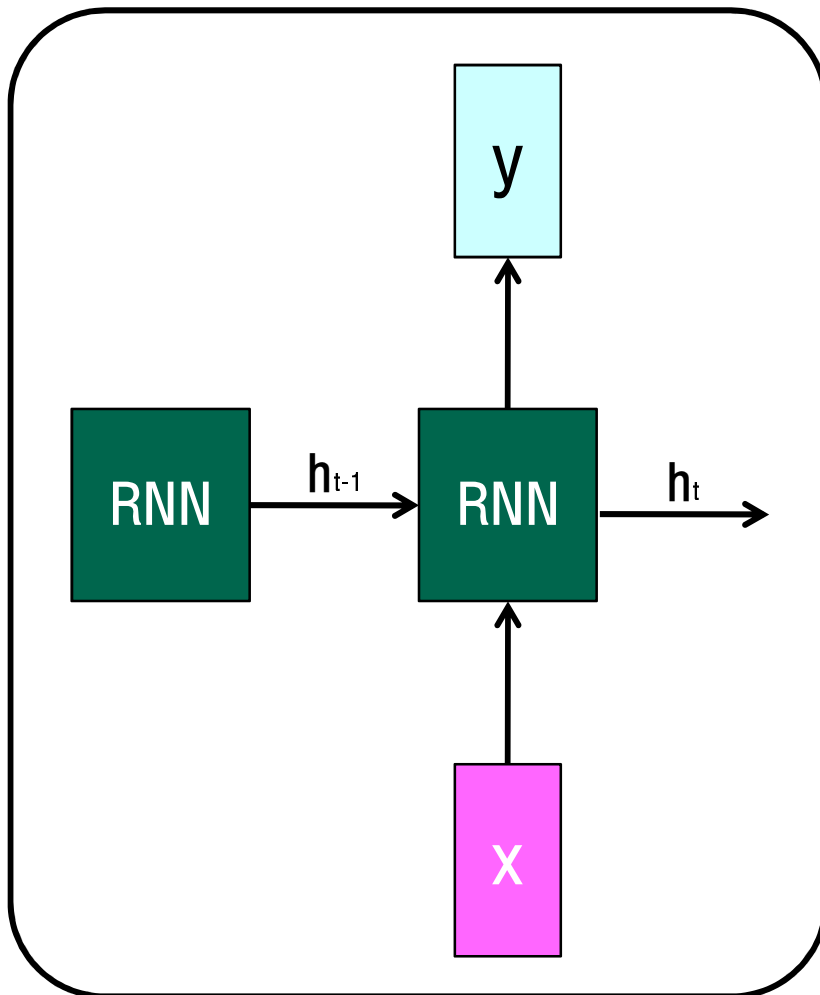
어제의 마감 증시 가격

어제 가격을 입력

오늘의 가격을 예측

RNN

- Recurrent Neural Network를 계산하는 방법



$$h_t = fw^*(h_{t-1}, x_t)$$

h_{t-1} : 이전 상태(old state)

x_t : input vector(입력하는 데이터)

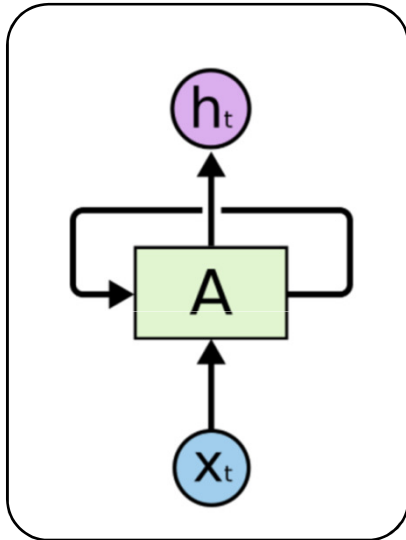
h_t : 새로운 상태(new state)

fw : 여러 단계를 거치더라도 fw 는 모두 동일한 값을 가진다.

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

(Vinila) RNN

- Recurrent Neural Network를 계산하는 방법



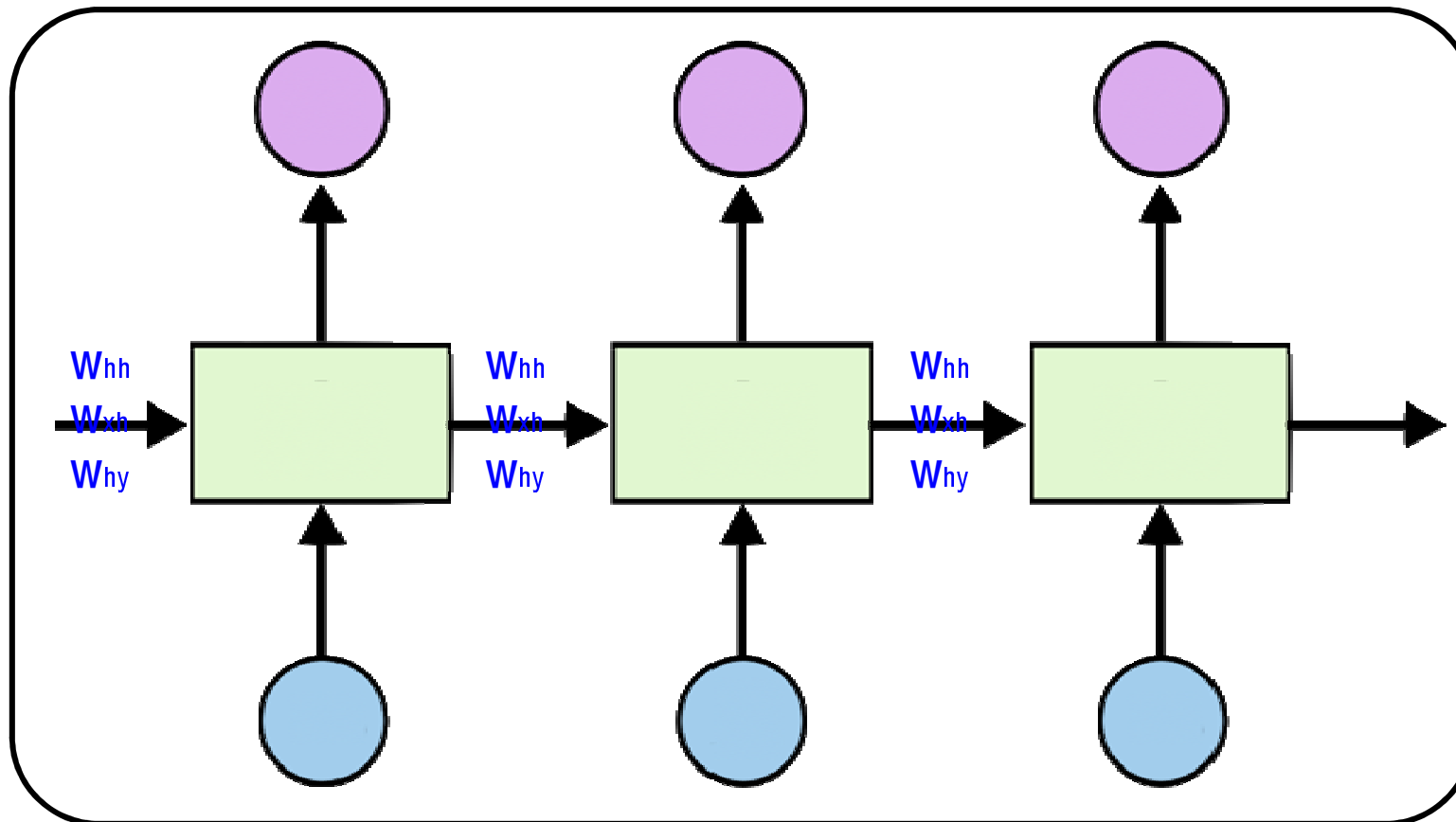
$$h_t = fw^*(h_{t-1}, x_t)$$

$$h_t = \tanh^*(w_{hh} * h_{t-1}, w_{xh} * x_t)$$

$$y_t = w_{hy} * h_t$$

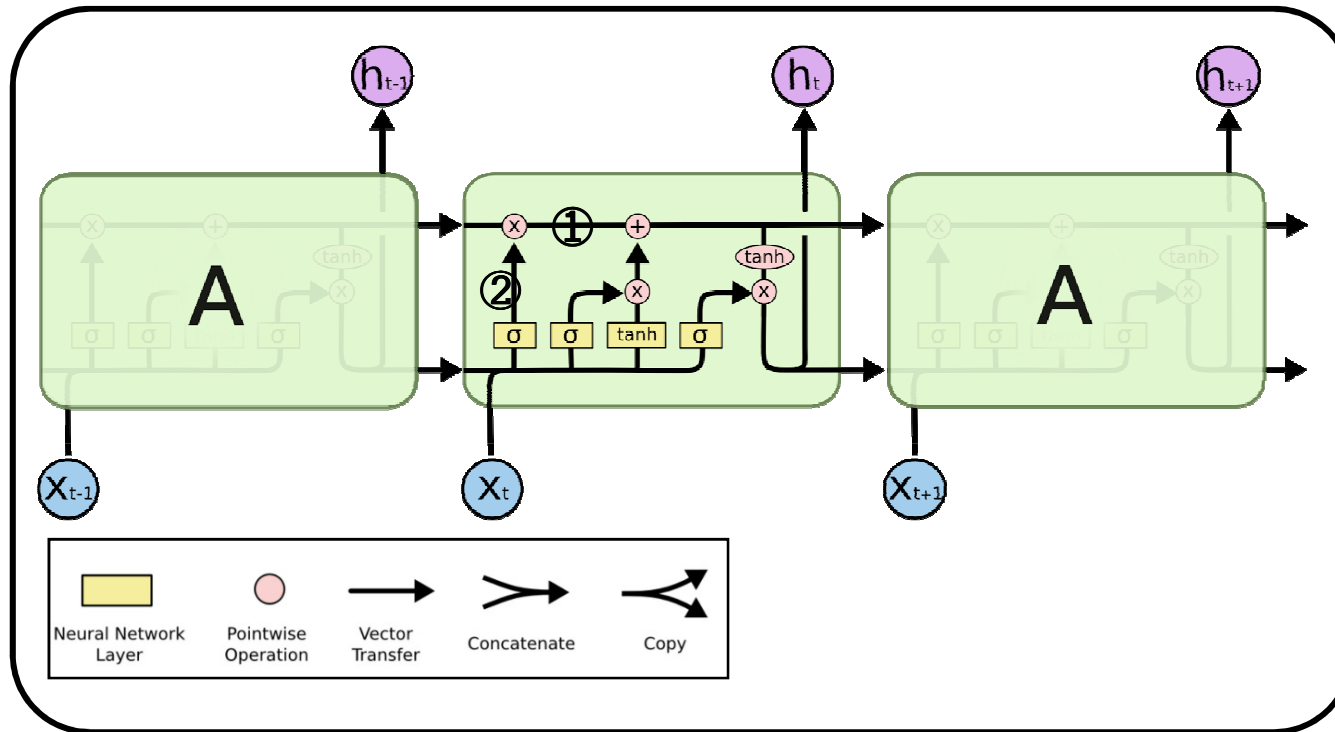
RNN

- Recurrent Neural Network



LSTM

- LSTM(Long Short Term Memory)



- ① cell state : 상단부를 가로 지르는 horizontal line
- ② "forget gate layer"라는 sigmoid layer가 결정한다.
0(버림), 1(정보 취함)

정해야 할 값들

- 코딩을 하면서 거의 동일한 패턴으로 사용하는 변수들을 정리해본다.

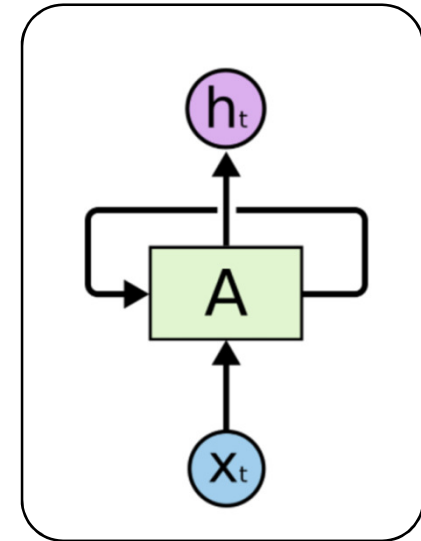
변수 이름	설명
input_dimension	입력의 차원 크기(dimension)
hidden_size	출력의 차원 크기(dimension) one hot 형식으로 출력된다.
sequence_length	시퀀스의 길이(그림의 네모 박스 갯수)
batch_size	입력될 문자열의 개수
idx2char	유니크한 문자열을 저장하고 있는 리스트
char2idx	{문자:인덱스} 형식을 저장하고 있는 사전
sample_idx	각 글자들의 색인을 저장하고 있는 리스트
num_classes	분류될 클래스의 개수 idx2char와 동일한 값을 가진다.
dictionary_size	사전의 크기
stacked_number	Stacked RNN에 몇 층으로 할 것인가를 지정한다.

RNN

- RNN을 구현하는 일반적인 절차
- 1. cell 객체를 구한다.
 - 1) 출력의 사이즈(hidden_size)를 결정한다.
- 2. 구동한다.
 - 1) cell 객체
 - 2) 입력 벡터(input vector)
 - 3) 구동 후 반환되는 항목
 - (1) output : 출력 결과
 - (2) states : 마지막 상태

셀을 생성하는 단계

셀을 가지고
학습 구동하는 단계



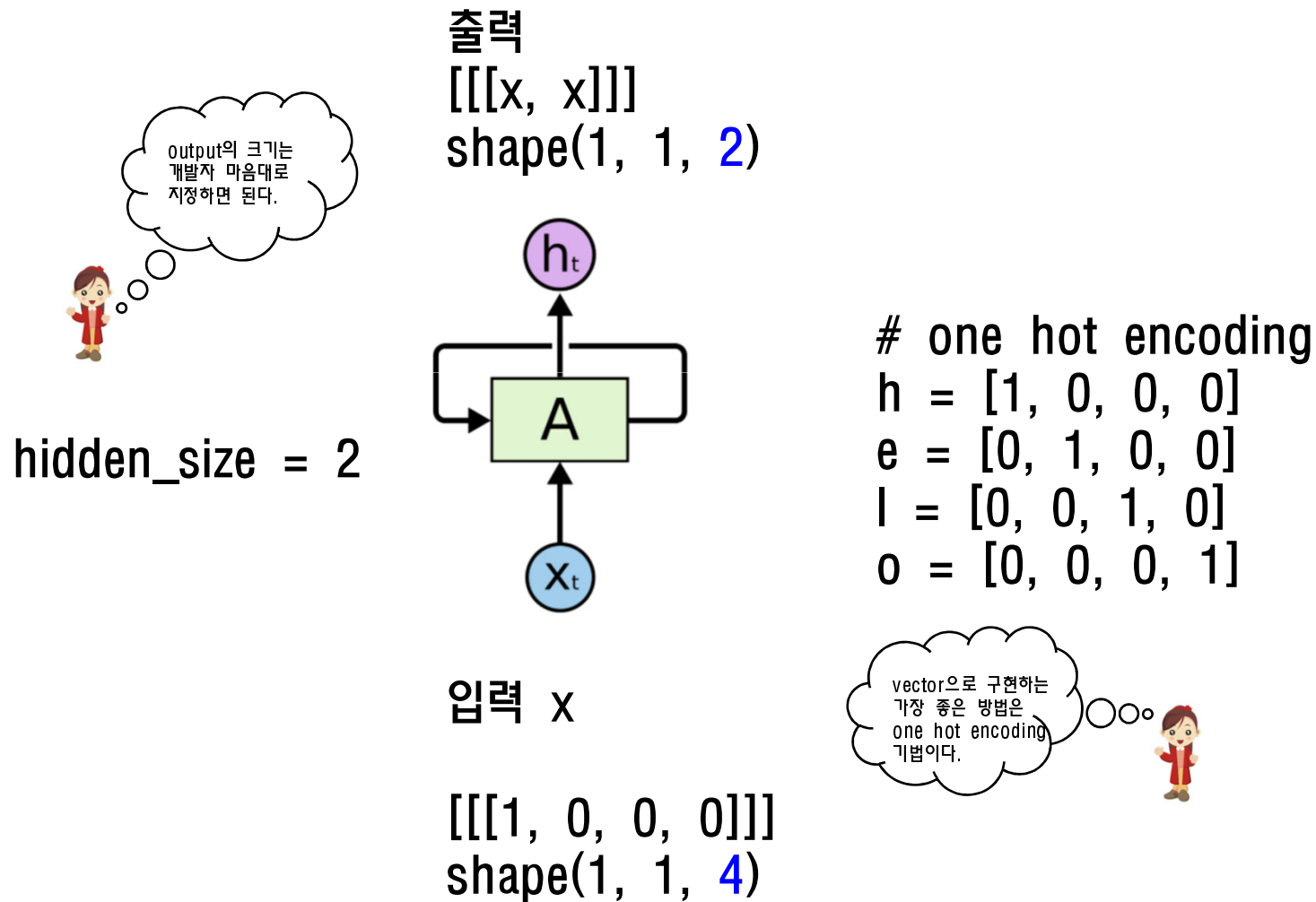
```
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size)
```

...

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


One node

- 예시는 4개의 입력(input-dim)이 들어가서 2(hidden_size)개의 출력이 나오는 예시이다.



One node

- 예시는 4개의 입력(input-dim)이 들어가서 2(hidden_size)개의 출력이 나오는 예시이다.
- 파일 이름 : rnn_01_basic_01.py

```
import tensorflow as tf
import numpy as np
```

```
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 = tf.Session()
```

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

```
print(sess.run(outputs))
```

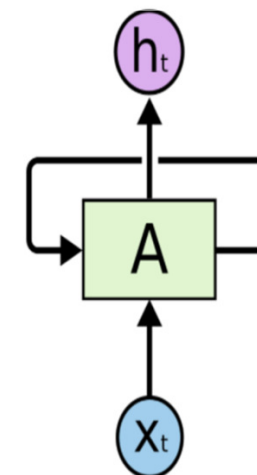
hidden_size의
수만큼 결과가 나왔다.



출력 예시

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

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



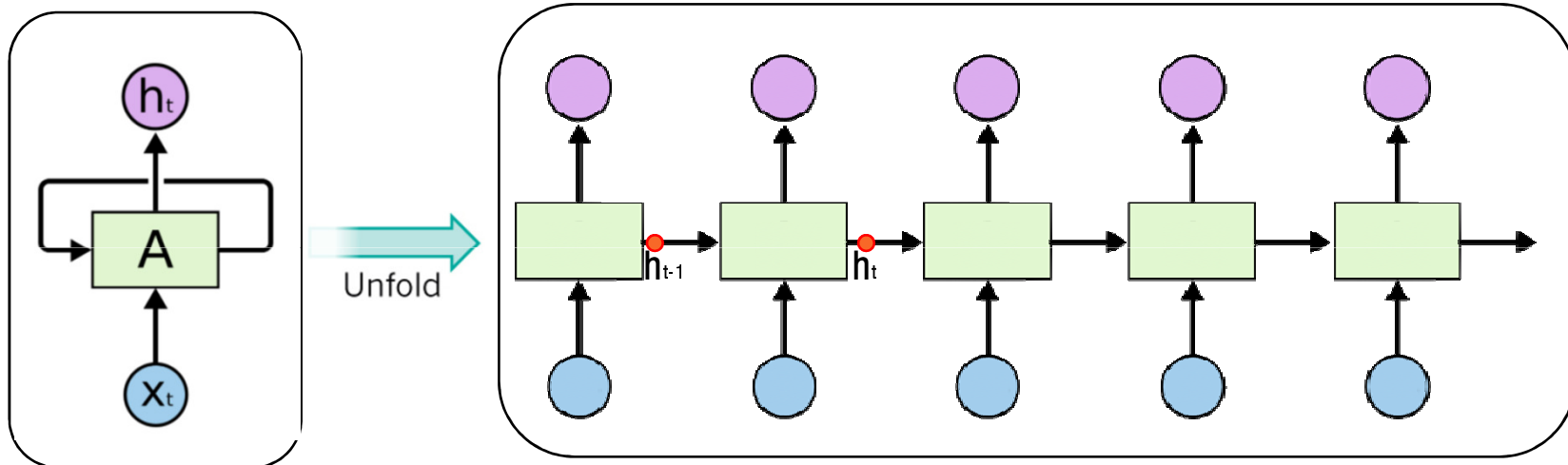
[[[1,0,0,0]]]
shape=(1,1,4)

Unfolding to n sequences

- 알파벳 5개(sequence_length=5)를 입력해주겠다.

shape(1, 5, 2)

[[[x, x], [x, x], [x, x], [x, x], [x, x]]]



입력 x

[[[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

shape(1, 5, 4)

Hidden_size=2

Input dimension = 4

sequence_length=5

Unfolding to n sequences

- 알파벳 5개(sequence_length=5)를 입력해주겠다.
- 파일 이름 : rnn_01_basic_02.py

```
import tensorflow as tf
import numpy as np

hidden_size = 2

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

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

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

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

print(sess.run(outputs))
```

입력의 Shape
(1, 5, 4)
'hello'

```
[[[ 0.00199991  0.05925646]
   [ 0.19543312  0.02881216]
   [ 0.36373657 -0.04851383]
   [ 0.46824604 -0.08120006]
   [ 0.32149205 -0.0405251 ]]]
```

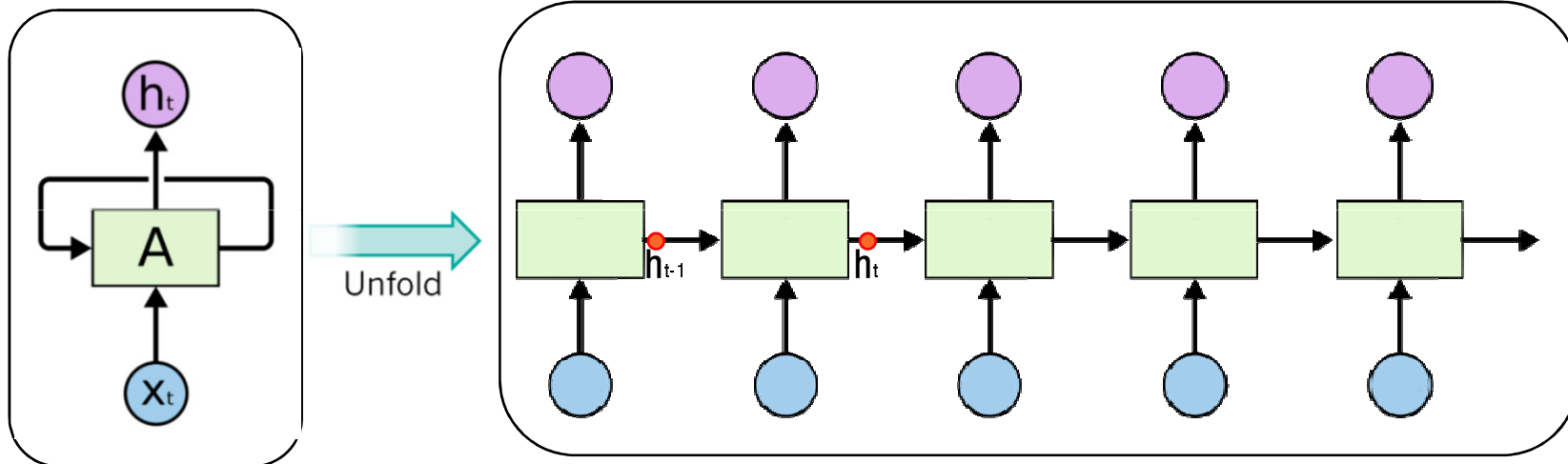
출력의 Shape
(1, 5, 2)

Batching input(배치 처리)

- 문자열을 여러 셋트(예시에서는 3 세트) 넣어서 일괄 처리해준다.

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]]]
```



```
[[[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]], # eoll  
 [[0, 0, 1, 0], [0, 0, 1, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0]]] # lleel
```

shape(3, 5, 4)

batch_size = 3 (3개를 일괄 입력)

Batching input(배치 처리)

- 문자열을 여러 셋트(예시에서는 3 셋트) 넣어서 일괄 처리해준다.
- 파일 이름 : rnn_01_basic_03.py

```
hidden_size = 2
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size)
x_data = np.array([[[ 1., 0., 0., 0.],
                    [0., 1., 0., 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., 1., 0., 0.],
                    [0., 1., 0., 0.],
                    [0., 0., 1., 0.] ]], dtype=np.float32)
outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
sess = tf.Session()
sess.run(tf.global_variables_initializer())
print(sess.run(outputs))
```

입력의
shape(3, 5, 4)
'hello', 'eolll', 'lleel'

```
[[[ 0.16913295  0.09676598]
 [ 0.23080564  0.03097105]
 [ 0.04161567 -0.02242604]
 [-0.08938518 -0.02789405]
 [-0.07891441 -0.06986038]]

 [[ 0.1771442  -0.01965534]
 [ 0.04581871 -0.1079137 ]
 [-0.08907796 -0.08347935]
 [-0.1689633  -0.05801167]
 [-0.21024437 -0.03112213]]

 [[-0.11138839 -0.0070499 ]
 [-0.17391098  0.00049861]
 [ 0.10956379  0.00397476]
 [ 0.21601819 -0.03345909]
 [ 0.02442703 -0.04979977]]]
```

출력의 Shape
(3, 5, 2)

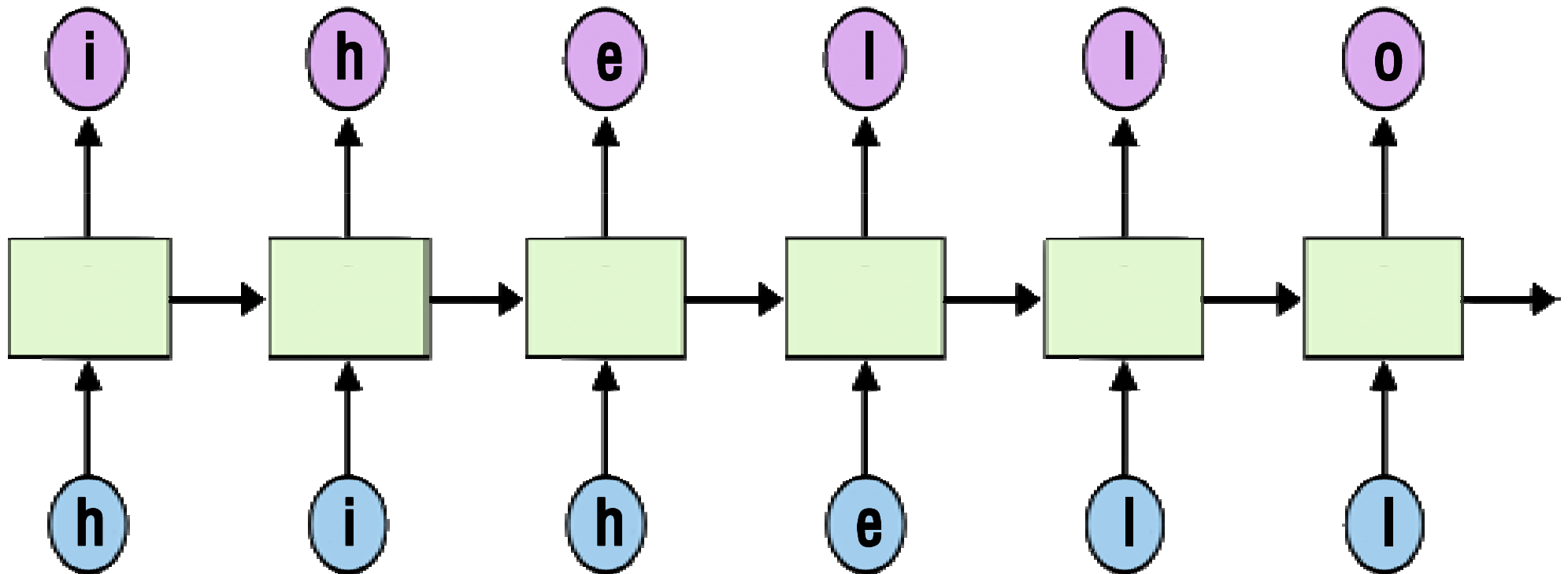
tensorflow library

Hi Hello RNN



Teach RNN 'hihello'

- h를 입력하여 순차적으로 다음과 같은 문자열이 출력되게 만들겠다.
- 파일 이름 : rnn_02_hello_rnn.py
- 출력할 문자열 : **hihello**



One-hot encoding

- h를 입력하여 순차적으로 문자열이 출력되게 만들겠다.

text: 'hihello' # 각각의 문자를 one hot 인코딩 시켜야 한다.

unique chars (vocabulary, voc):

h, i, e, l, o # **유일한 문자열**로 만들어야 한다.

voc index: # 인덱스를 딕셔너리라고 부른다.

h:0, i:1, e:2, l:3, o:4

컴퓨터가 0이면 'h'로 해석하겠다.
컴퓨터가 1이면 'i'로 해석하겠다.

[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'

iii 에이콘아카데미 강남

정해야 할 값들

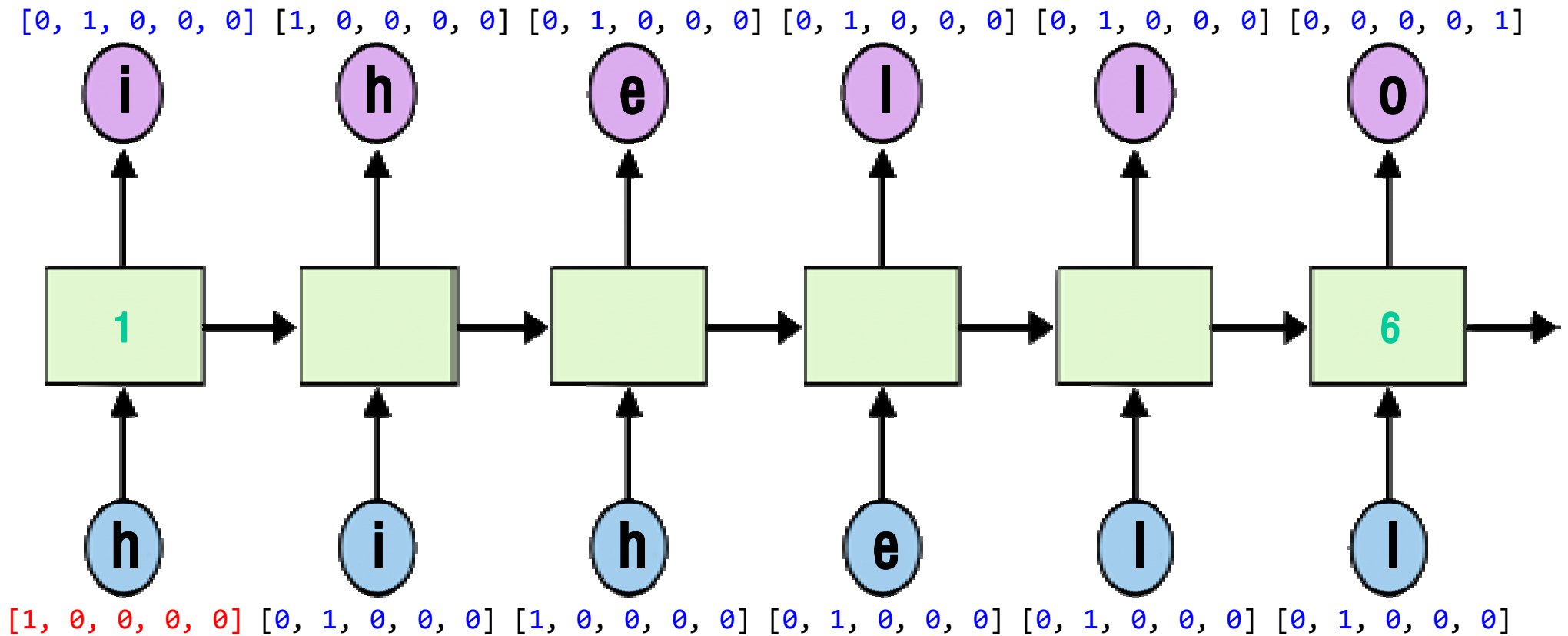
입력 dimension : 5

시퀀스 길이 : 6 (녹색 1, 6)

출력 dimension : 5 (one hot으로 나온다)

batch size : 1(문자열 1개이므로...)

[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

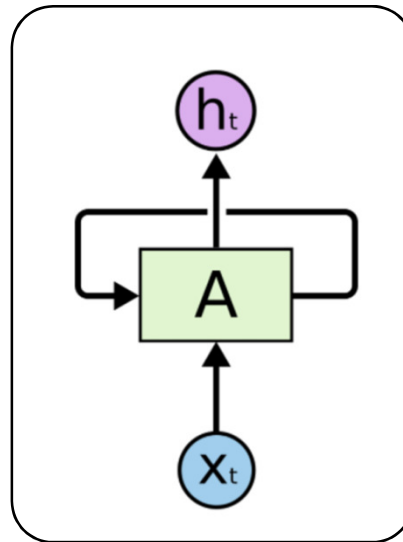


Execute RNN

- RNN을 실행하기

```
# RNN model  
hidden_size : cell의 출력 사이즈이다.  
  
rnn_cell = rnn_cell.BasicLSTMCell( hidden_size )  
  
outputs, _states = tf.nn.dynamic_rnn(  
    rnn_cell, x, initial_state=initial_state, dtype=tf.float32 )
```

입력 dimension : 5
시퀀스 길이 : 6



Data creation

- 입출력 데이터 생성

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

```
# 입력 값 : x_data
```

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

```
# x_data를 one hot encoding 하여 x_one_hot을 생성한다.
```

```
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
```

```
# 결과 값(y_data) : ihello
```

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

Feed to RNN

- RNN에 데이터 넣어 주기

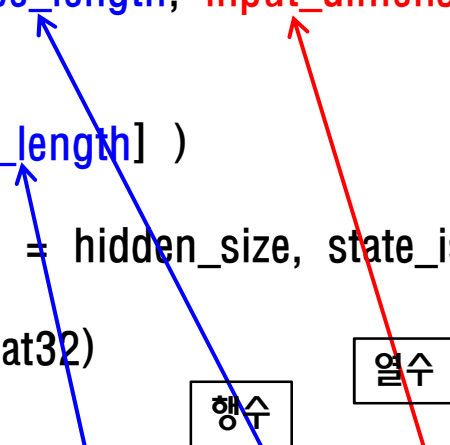
```
# x one-hot
x = tf.placeholder( tf.float32, [None, sequence_length, input_dimension] )

# y label
y = tf.placeholder( tf.int32, [None, sequence_length] )

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)
```



행수

열수

```
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
```

Cost : sequence_loss

- output이 얼마나 좋은가를 나타내는 지표로 loss를 계산해야 한다.
- 텐서 플로우에는 sequence_loss 함수를 제공하고 있다.
- sequence_loss 함수
 - logits : 우리의 예측(one hot 형식)
 - targets : y_data
 - weights : 얼마나 중요한가를 나타내는 지표(모두 1이라고 보면 된다.)

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

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)

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 # 즉, 0.9가 0.7보다는 훨씬 정답에 가까우므로 손실이 적다고 나온다.
```

Cost : sequence_loss

```
outputs, _states = tf.nn.dynamic_rnn(  
    cell, x, initial_state=initial_state, dtype=tf.float32)  
  
# weights : batch_size 수 만큼 모두 1을 넣어 주면 된다.  
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& Results

```
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))
```

```
0 loss: 1.55474 prediction: [[3 3 3 3 4 4]] true y: [[1, 0, 2, 3, 3, 4]] Prediction str: IIIloo
1 loss: 1.55081 prediction: [[3 3 3 3 4 4]] true y: [[1, 0, 2, 3, 3, 4]] Prediction str: IIIloo
2 loss: 1.54704 prediction: [[3 3 3 3 4 4]] true y: [[1, 0, 2, 3, 3, 4]] Prediction str: IIIloo
3 loss: 1.54342 prediction: [[3 3 3 3 4 4]] true y: [[1, 0, 2, 3, 3, 4]] Prediction str: IIIloo
...
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
```


tensorflow library

RNN with long sequences



Better data creation

- 문자열이 짧으면 하드 코딩하면 되지만 문자열이 길어 지면 힘들어진다.
- **보편적인 코딩**을 해보도록 한다.
- 파일 이름 : rnn_03_char_sequence.py

```
sample = "if you want you"
```

```
# 중복 배제하고, 유니크한 단어 모음집 만들기  
idx2char = list(set(sample)) # index -> char
```

```
# 문자열을 넣어 주면 인덱스를 알려 주는 사전  
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
```

```
# num_classes는 idx2char의 크기와 동일하다.  
x_one_hot = tf.one_hot( x, num_classes ) # one hot: 1 -> 0 1 0 0 0 0 0 0 0 0
```

				y_data
if	you	want	you	
				x_data

Hyper parameters

- 문자열이 정해지면 나머지 파라미터들은 자동으로 다음과 같이 구할 수 있다.

```
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.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: nnuffuunnnuuuyuy
1 loss: 2.29675 Prediction: nnuffuunnnuuuyuy
...

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

Really long sentence?

- 이번에는 제법 길다고? 생각하는 문자열을 이용하여 테스트 해본다.
- 파일 이름 : rnn_04_long_char_non_stack.py

```
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.")
```

```
# 문자열이 너무 길어서 sequence_length=10 만큼 잘라서 계속 비교한다.  
# 아래 예시는 batch_size = 170인 경우이다.
```

```
# 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.
```

Making dataset

- 이번에는 제법 길다고? 생각하는 문자열을 이용하여 테스트 해본다.

```
idx2char = list(set(sample))
char2idx = {w: i for i, w in enumerate(idx2char)}

dataX = []
dataY = []

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

    x = [char2idx[c] for c in x_str] # x str to index
    y = [char2idx[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.
```

RNN parameters

```
idx2char = list(set(sample))  
char2idx = {w: i for i, w in enumerate(idx2char)}  
dictionary_size = len(char2idx)  
hidden_size = len(char2idx)  
num_classes = len(char2idx)  
sequence_length = 10 # 임의의 숫자로 지정했다.  
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 = 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)
```

Run long sequence RNN
Why it does not work?

tensorflow library

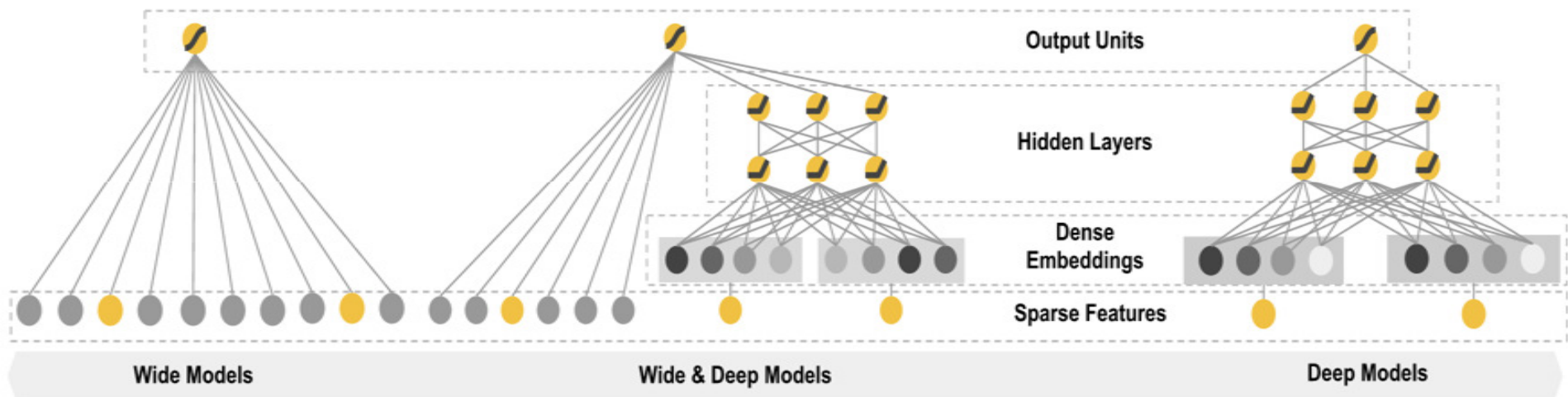
RNN with long sequences:

Stacked RNN + Softmax layer



Wide & Deep

- 제대로 실행되지 않았다.
- 데이터가 많아지다 보면 1층만으로 부족하다.
- 깊고 넓게(Wide & Deep)층을 많이 쌓아 보자.
- Stacked RNN의 필요성



Stacked RNN

- Stacked RNN 구현하기
- 파일 이름 : rnn_05_long_char_stacked.py

```
x = tf.placeholder(tf.int32, [None, sequence_length])
y = tf.placeholder(tf.int32, [None, sequence_length])
```

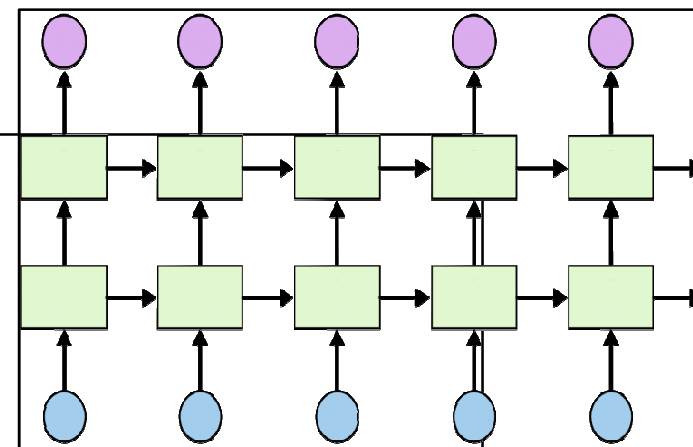
```
# One-hot encoding
x_one_hot = tf.one_hot(X, num_classes)
print(x_one_hot) # check out the shape
```

```
# 1. 일단 cell 객체를 구한다.
cell = rnn.BasicLSTMCell(hidden_size, state_is_tuple=True)
```

```
stacked_number = 2
# MultiRNNCell 함수에 cell 객체를 넣으면서 산술 연산 곱셈을 하면 층 수를 늘릴 수 있다.
multi_cells = rnn.MultiRNNCell([cell] * stacked_number, state_is_tuple=True) # 2층
```

```
# outputs: unfolding size x hidden size, state = hidden size
outputs, _states = tf.nn.dynamic_rnn(cell, x_one_hot, dtype=tf.float32)
```

```
outputs, _states = tf.nn.dynamic_rnn(multi_cells, x_one_hot, dtype=tf.float32)
```



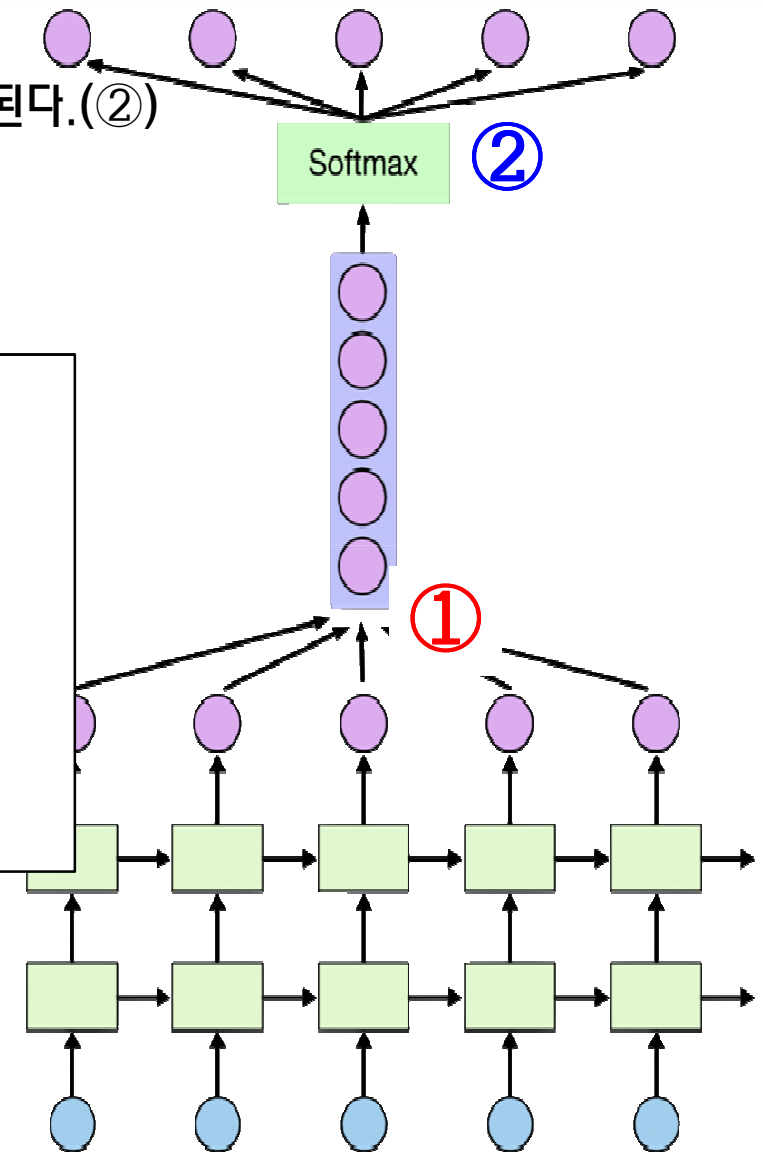
Softmax

- RNN에서 나온 결과에 Softmax를 붙여 주자.
- Softmax에 한 줄로 넣어 주고(①), 나중에 다시 펼쳐 주면 된다.(②)
- Softmax에 맞는 w만 reshape 해주면 된다.

①

```
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])  
  
# sm_outputs는 softmax의 output을 의미한다.  
sm_outputs = tf.matmul(x_for_softmax, softmax_w) + softmax_b  
  
outputs = tf.reshape(sm_outputs, [batch_size, sequence_length,  
num_classes])
```

②



Loss

```
# 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=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 tttttttt 3.23111
0 168 tttttttt 3.23111
0 169 tttttttt 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})

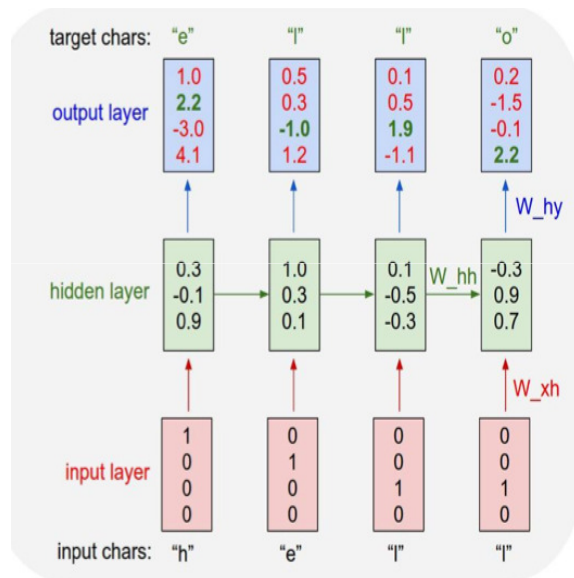
for j, result in enumerate(results):
    index = np.argmax(result, axis=1)

    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

- RNN 학습을 통하여 컴퓨터가 작성한 문장들이다.
- <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>



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:

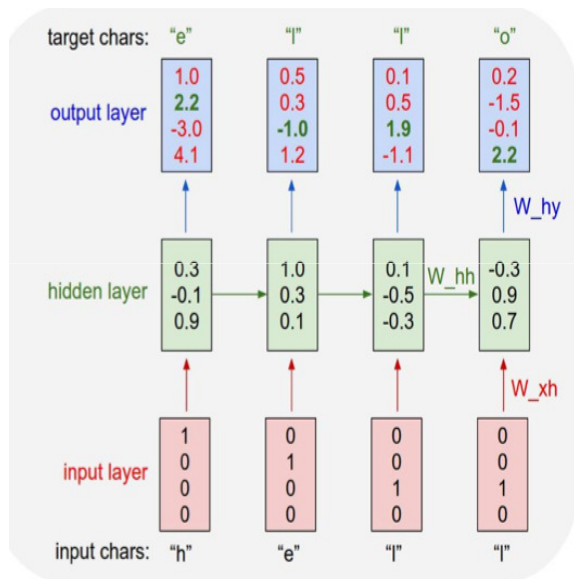
I'll drink it.

source code

Linux Source Code

강남

- RNN이 만든 소스 코드 내용.
- <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

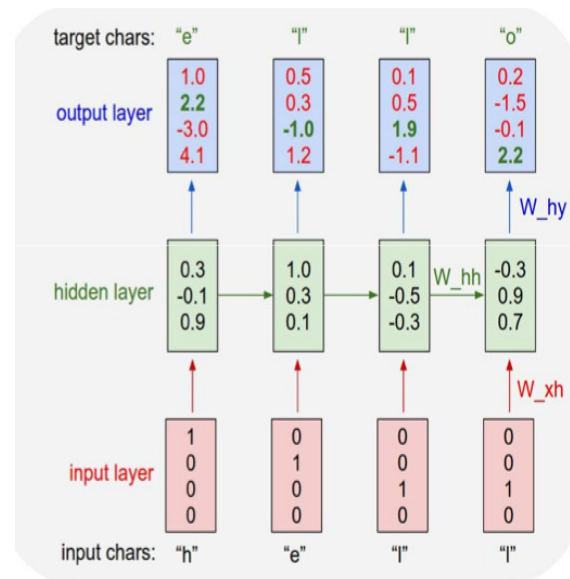


I wanted to push structured data to its limit, so for the final challenge I decided to use code. In particular, I took all the source and header files found in the [Linux repo on Github](#), concatenated all of them in a single giant file (474MB of C code) (I was originally going to train only on the kernel but that by itself is only ~16MB). Then I trained several deep neural nets 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;
}
```

char/word rnn

- char/word level n to n model
- <https://github.com/sherjilozair/char-rnn-tensorflow>
- <https://github.com/hunkim/word-rnn-tensorflow>



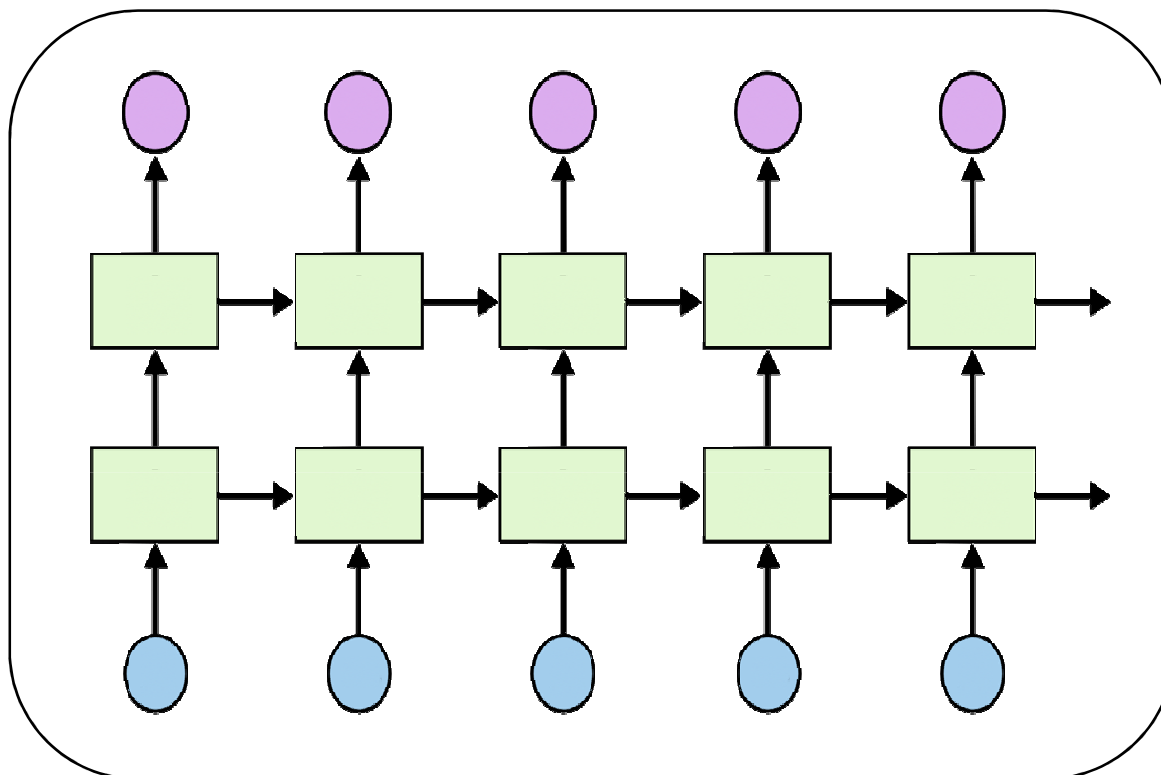
tensorflow library

Dynamic RNN



Different sequence length

- 들어 오는 데이터가 가변적이면 시퀀스의 길이를 동적으로 다르게 설정해야 한다.



sequence_length=[5,
2,
3]

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

이전에는 내부에서 패딩을
자동으로 넣었다.
하지만, 이것 때문에
비용 값이 이상하게
나왔다.



Dynamic RNN

```
# 3 batches 'hello', 'eoll', 'leel'
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.          ]])
```

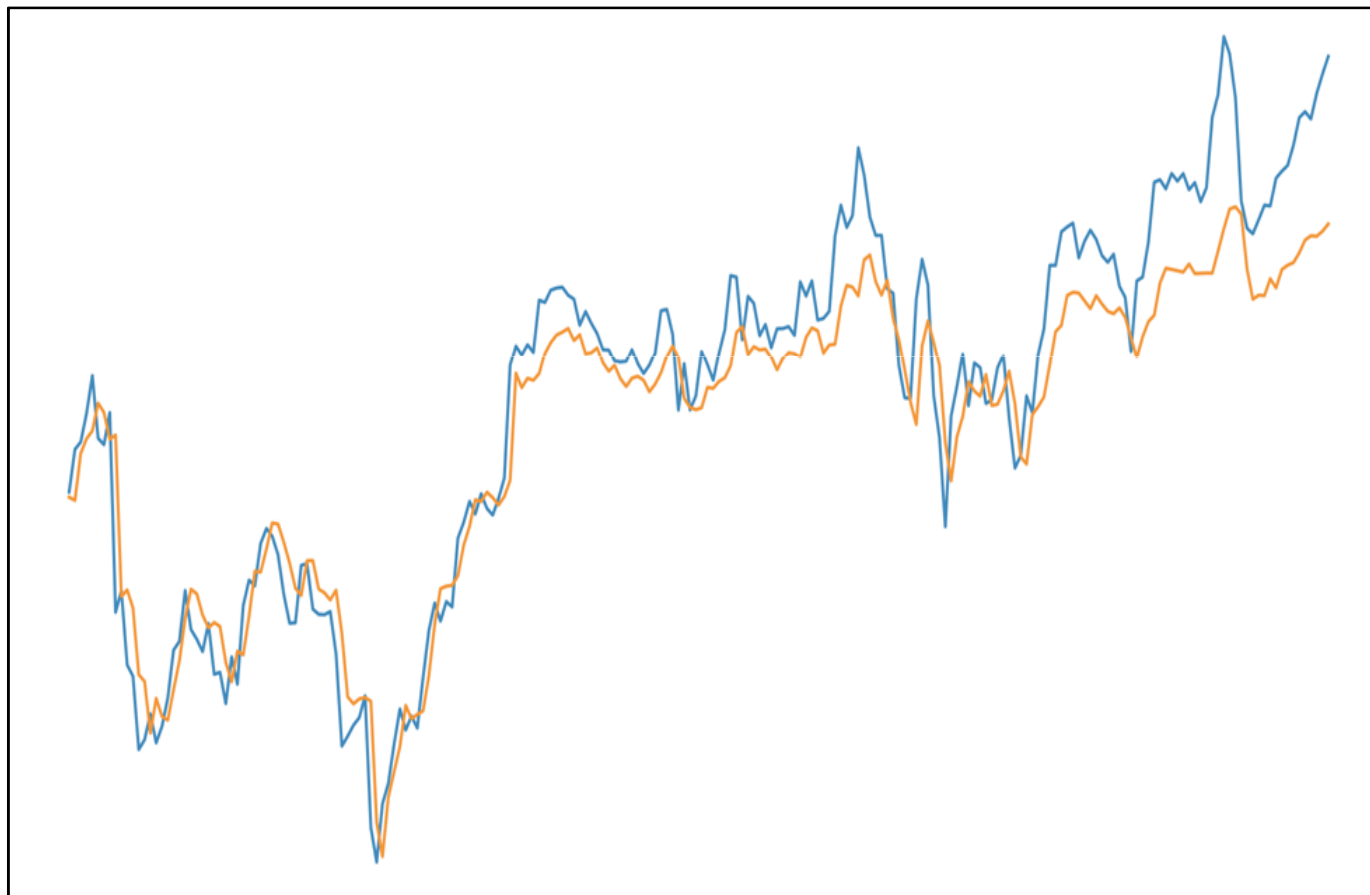
tensorflow library

RNN with time series data (stock)



Time series data

- 타임 시리즈 데이터는 시간에 따라서 값이 변하는 데이터를 의미한다.
- 이러한 것들을 예측해보자.



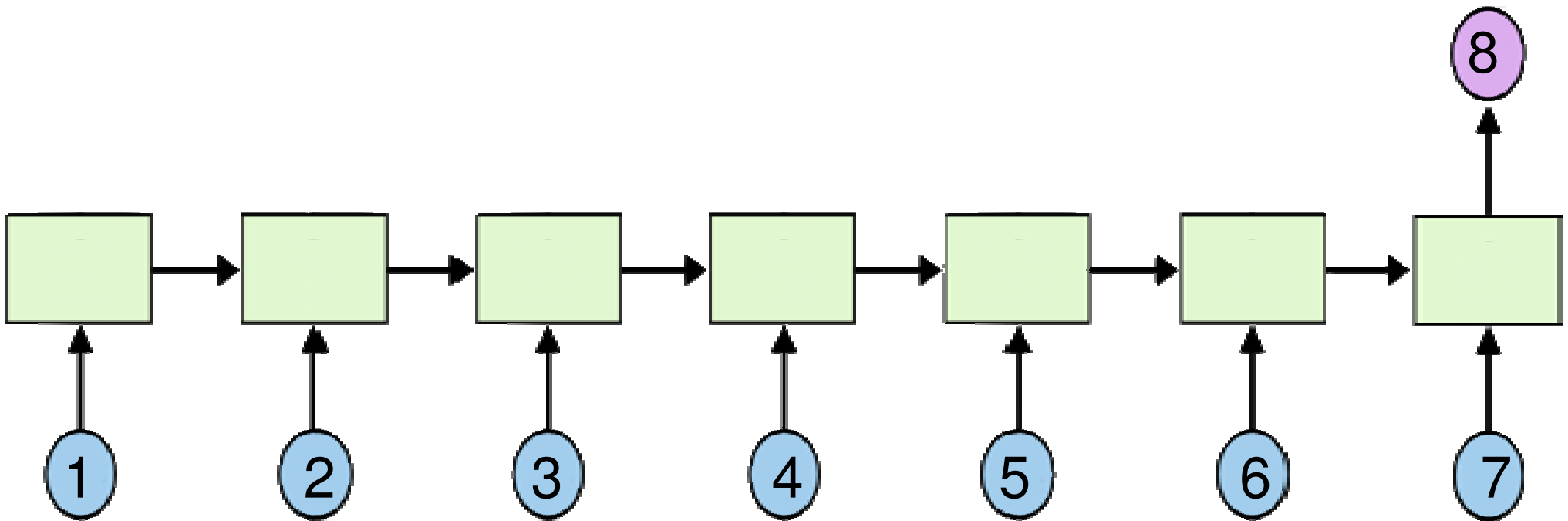
Time series data

- 날마다 연속적으로 이어져 있으므로 Time Series 데이터라고 한다.
- 파일 이름 : rnn_06_stock_prediction.py, data-02-stock_daily.csv

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

Many to one

- 7일간의 데이터를 가지고 8일째 데이터를 분석하자.
- 이전에 데이터를 연속적으로 연결해서 데이터를 예측한다.



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	?
807	811.840027	803.190002	1155300	?

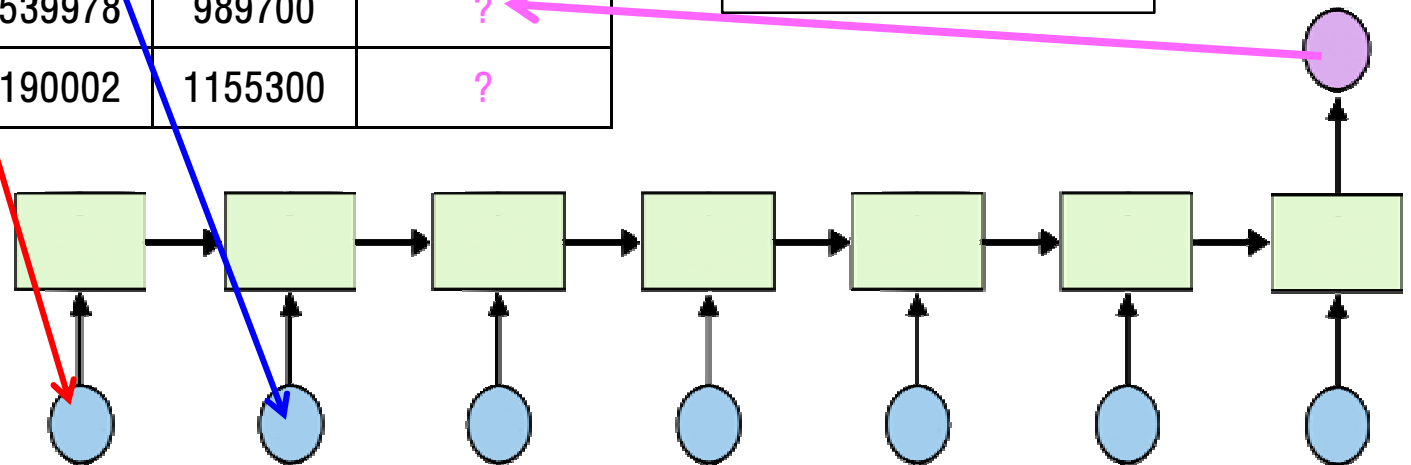
7일간의 데이터를 가지고
8일째 데이터를 분석하자.

정해야 할 값들

입력 dimension : 5

시퀀스 길이 : 7(7일)

output dimension : 1



Reading data

- 데이터 읽어 오기

```
timesteps = sequence_length = 7
input_dimension = 5
output_dimension = 1
```

```
# Open, High, Low, Close, Volume
xy = np.loadtxt('data-02-stock_daily.csv', delimiter=',')
```

```
# 시간순으로 만들기 위해서 꺾꾸로 뒤집기
xy = xy[::-1] # reverse order (chronically ordered)
```

```
xy = MinMaxScaler(xy) # 정규화
```

```
x = xy
y = xy[:, [-1]] # Close as label
```

```
dataX = []
dataY = []
for i in range(0, len(y) - sequence_length):
    _x = x[i:i + sequence_length]
    _y = y[i + sequence_length] # Next close price

    print(_x, "->", _y)
    dataX.append(_x)
    dataY.append(_y)
```

정해야 할 값들
입력 dimension : 5
시퀀스 길이 : 7(7일)
output : 1

```
# x 값 : 7일치 매일 5개씩
[[ 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 ]]
```

```
# y 값
--> [ 0.14106001]
```

Training and test datasets

- 훈련용과 테스트용 데이터 셋 준비하기

```
# 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, 가장 앞의 None은 batch_size
x = tf.placeholder(tf.float32, [None, sequence_length, input_dimension])

# 가장 앞의 None은 batch_size
y = tf.placeholder(tf.float32, [None, 1])
```

LSTM and Loss

```
# input placeholders
x = tf.placeholder(tf.float32, [None, sequence_length, input_dimension])

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

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

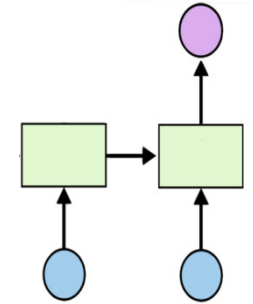
outputs, _states = tf.nn.dynamic_rnn(cell, x, dtype=tf.float32)

y_pred = tf.contrib.layers.fully_connected(
    outputs[:, -1], output_dimension, 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)
```



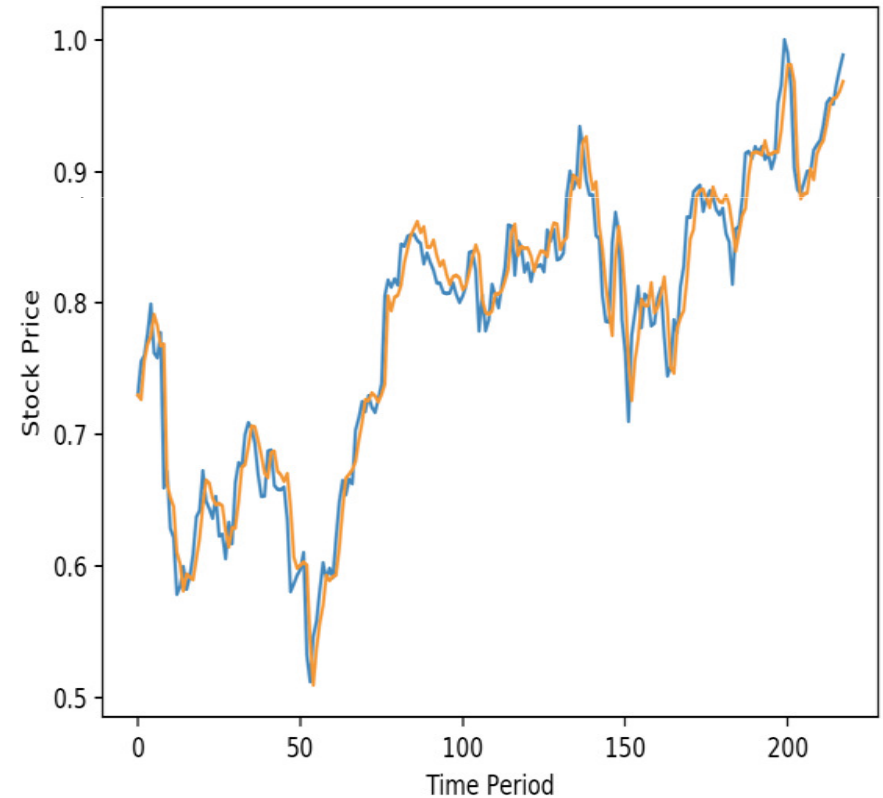
Training and Results

```
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
 - <http://tv.naver.com/v/2302940>
- Conversation Modeling/Question Answering
- Image/Video Captioning
 - <https://petapixel.com/2016/09/23/googles-image-captioning-ai-can-describe-photos-94-accuracy/>
 - <http://links.laughingsquid.com/post/146997649985/dense-captioning-of-video-demonstrating-the>
- Image/Music/Dance Generation