메디치소프트 기술연구소

딥러닝 7일

Recurrent Neural Network

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

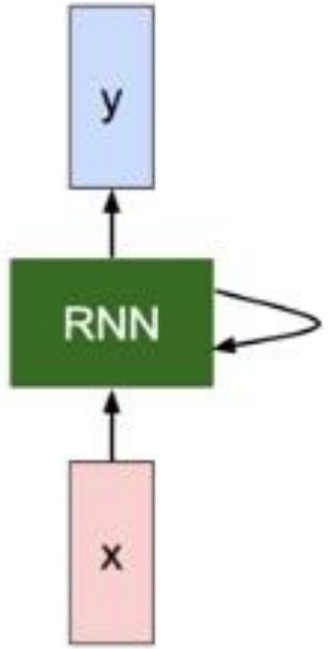
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

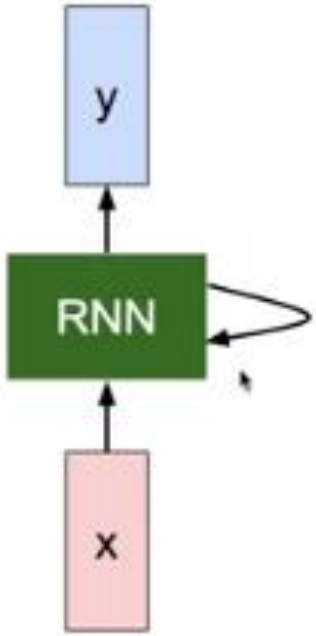
old state

input vector at some time step



(Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



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

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

$$y_t = W_{hy}h_t$$

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 17 8 Feb 2016

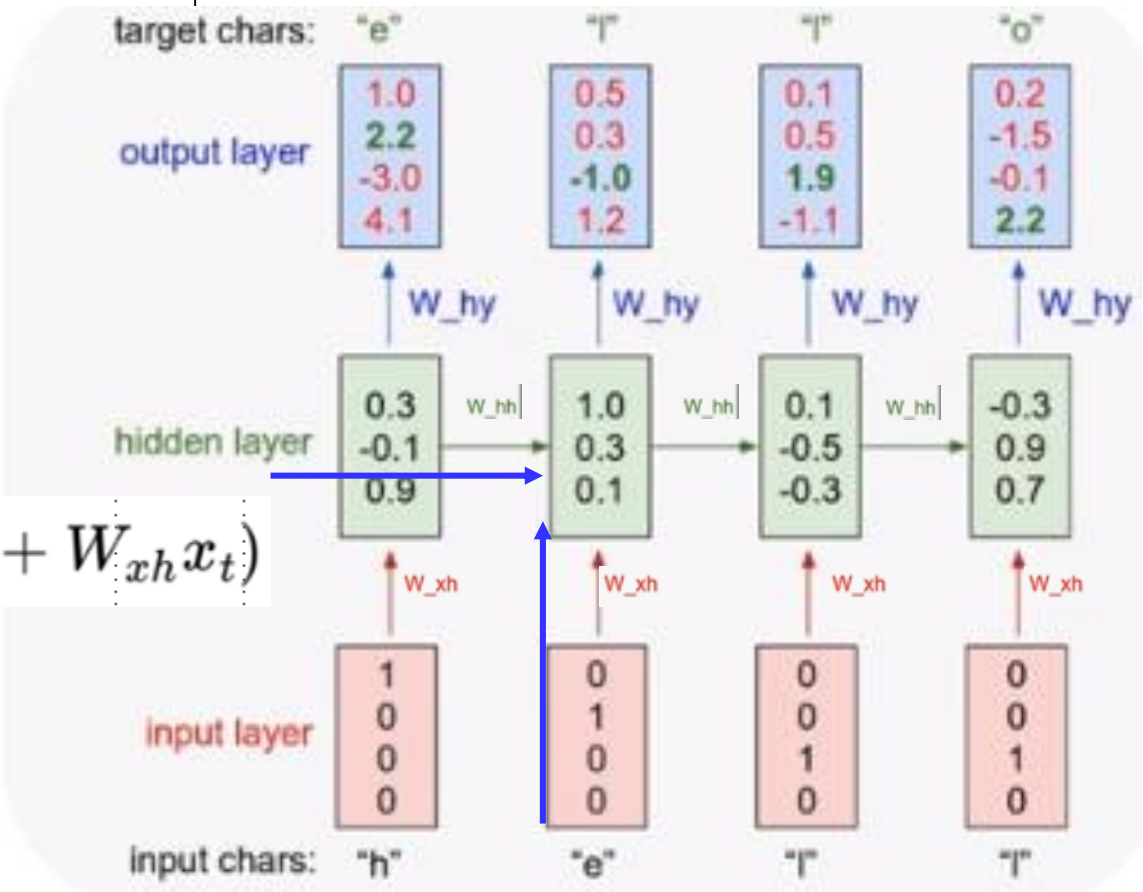
Character-level language model example

$$y_t = W_{hy} h_t$$

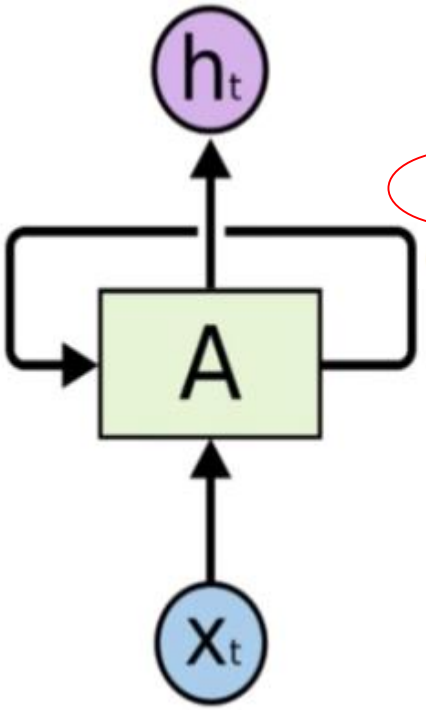
Vocabulary:
[h,e,l,o]

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

Example training sequence:
“hello”



RNN – 예제




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

출력사이즈

교체가능

Output이 다음의 셀과 연결

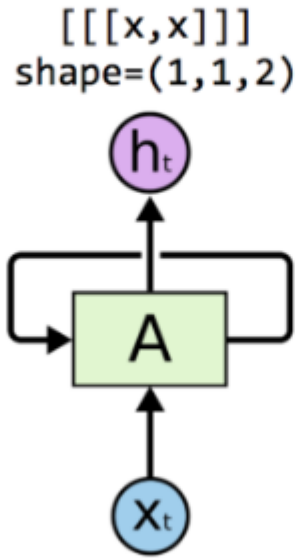
. Dynamix_rnn()
적절한 타임 스텝에 걸쳐 셀을 실행하기 위해 while_loop() 연산을 사용

. 출력사이즈 정함: hidden_size

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

hidden_size=2

입력데이터상관없이 출력의 Dimension은 2이다.



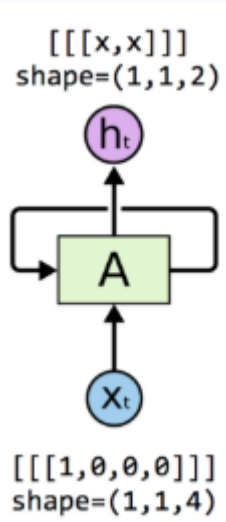

```
# One hot encoding
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

RNN – 예제Input 4 dim->2 hidden_size

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
import pprint
pp = pprint.PrettyPrinter(indent=4)
sess = tf.InteractiveSession()
# One cell RNN input_dim(4) -> output_dim (2)
hidden_size = 2
cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)
print(cell.output_size, cell.state_size)
x_data = np.array([[[1,0,0,0]]], dtype=np.float32) # x_data = [[[1,0,0,0]]]
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())
```

출력

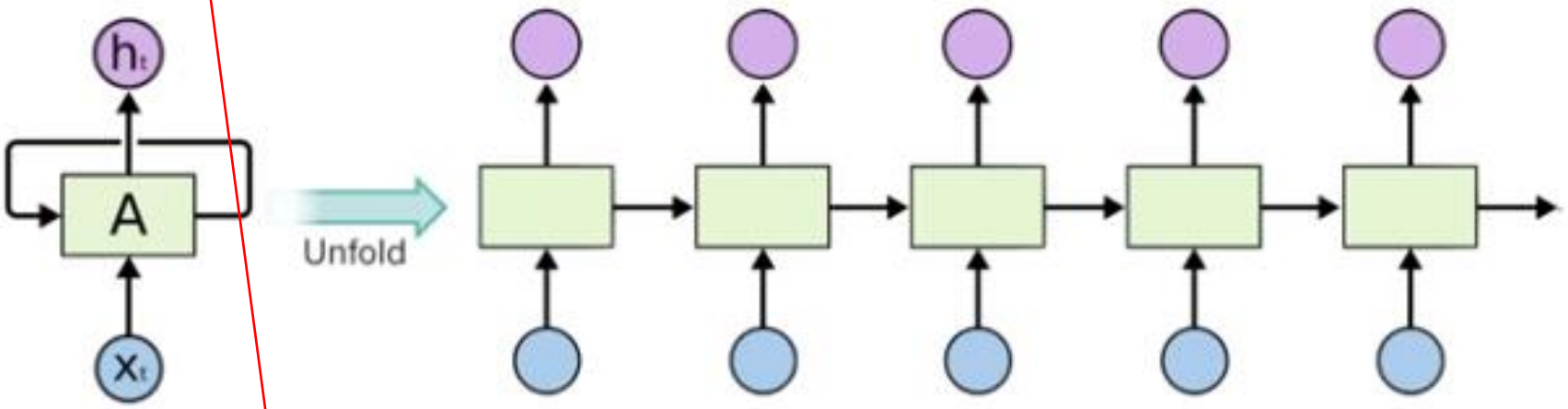
출력값



Unfolding to n sequences

```
Hidden_size=2
sequence_length=5
```

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



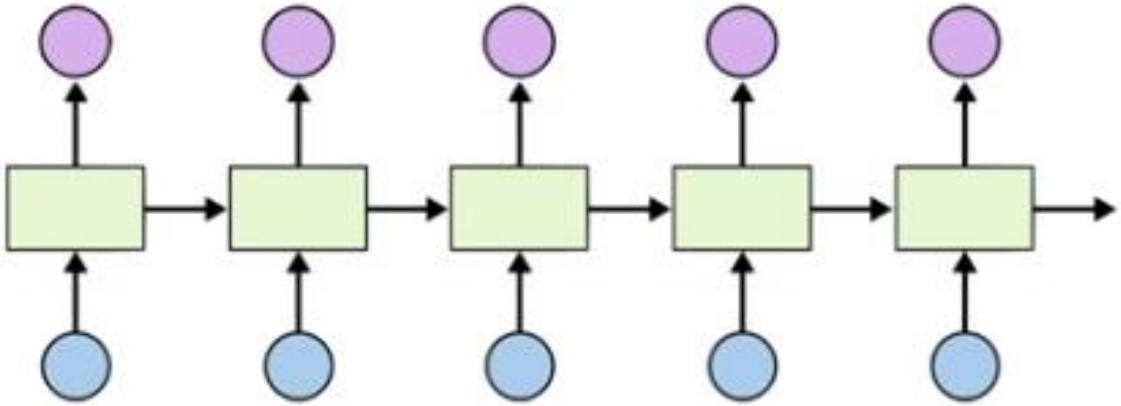
shape=(1,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]
h e l l o

RNN – HELLO

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

Batching input

```
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]]]
```



```
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
```

RNN – Hihello예제

```python
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
import pprint

pp = pprint.PrettyPrinter(indent=4)
sess = tf.InteractiveSession()
with tf.variable_scope('two_sequances') as scope:
    # One cell RNN input_dim (4) -> output_dim (2). sequence: 5
    hidden_size = 2
    cell = tf.contrib.rnn.BasicRNNCell(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)
    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())
```

입력과 출력의 shape확인

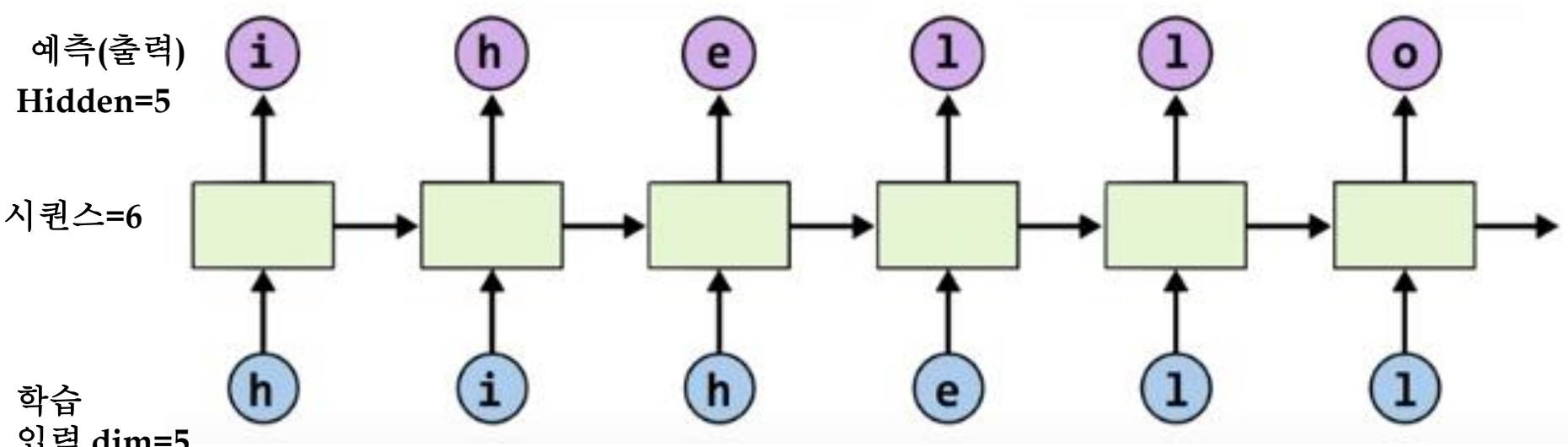
```
array([[[-0.27701476, -0.47703248],
        [-0.17500615,  0.2320899 ],
        [ 0.6444088 ,  0.15394928],
        [ 0.11884318,  0.10508373],
        [ 0.55162483,  0.43363634]]], dtype=float32)
```

RNN – HIHELLO예제

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

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

RNN – HIHELLO예제



배치 =1(문자열)

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

.[1, 0, 0, 0, 0] #h
.[0, 1, 0, 0, 0] #i
.[0, 0, 1, 0, 0] #e
.[0, 0, 0, 1, 0] #l
.[1, 0, 0, 0, 1] #o

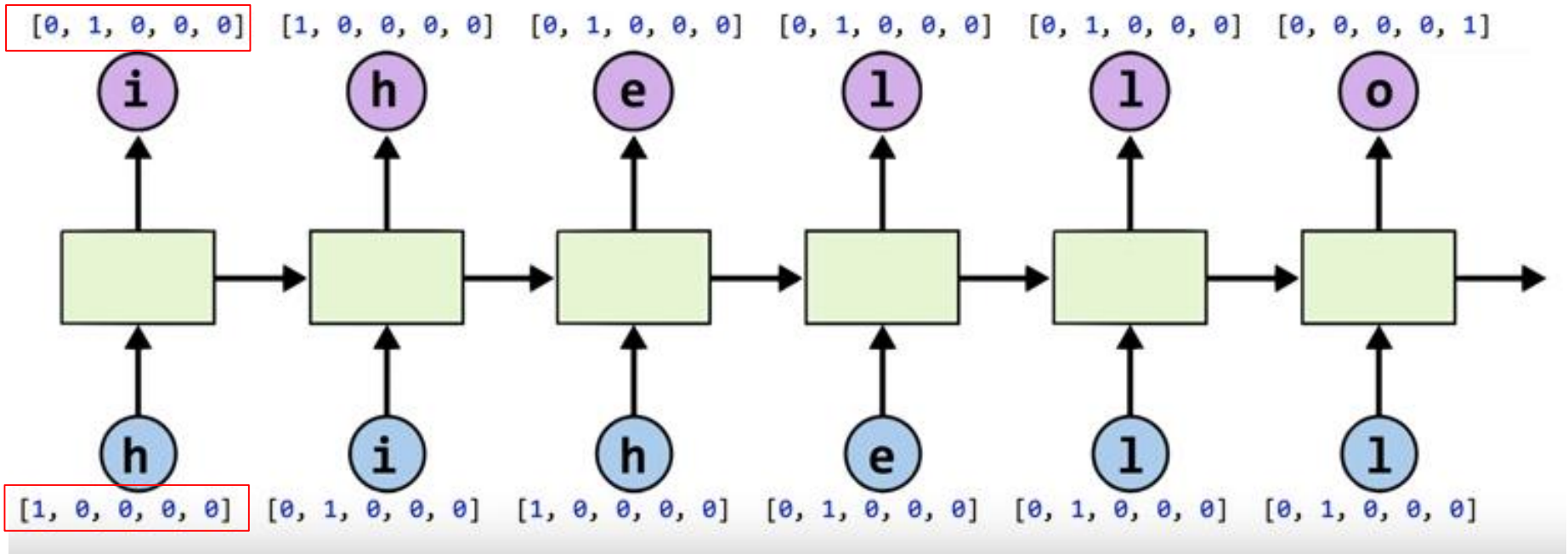
Hihello
h=>예측
I =>예측

RNN – HIHELLO예제

Teach RNN 'hihello'

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

Hidden_size =5



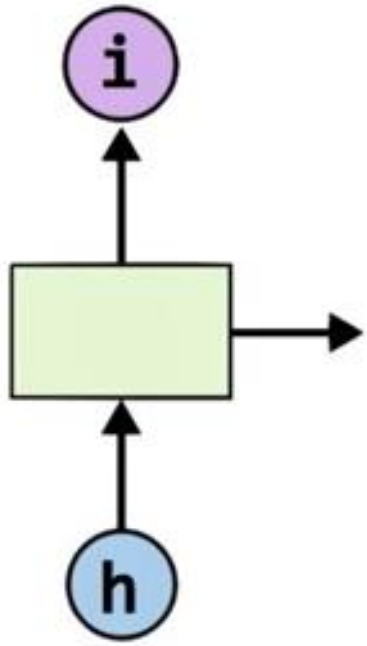
Execute RNN

hidden_rnn_size

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


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

1

2



RNN – HIHELLO예제

```
import tensorflow as tf
import numpy as np
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
input_dim = 5  # one-hot size
hidden_size = 5  # output from the LSTM. 5 to directly predict one-hot
batch_size = 1   # one sentence
sequence_length = 6  # |ihello| == 6
learning_rate = 0.1
```

문자를 index로

인덱스

one-hot

학습시키고자하는 문자열

#Hihello=> on-hotcoding
#index h:0,i:1,e:2,l:3,0:4 (dictionary)

#출력하고자 하는 라벨

RNN – HIHELLO예제

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

cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size, state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32) 출력:5
outputs, _states = tf.nn.dynamic_rnn(
    cell, X, initial_state=initial_state, dtype=tf.float32)

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

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

#예측 #1

RNN – HIHELLO예제

```
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)

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

```
1997 loss: 1.5854686e-05 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
   Prediction str:  ihello
1998 loss: 1.5834818e-05 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
   Prediction str:  ihello
1999 loss: 1.5795082e-05 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
   Prediction str:  ihello
```

숫자=>문자변환

RNN – Long Sequence Rnn

```
import tensorflow as tf
import numpy as np
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 (unit #)
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
```

#각각의 unique한 값

#i:숫자, C:문자

#전체길이

#sample data index

#문자의갯수

RNN – HIHELLO예제 –Data creation

```
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)

# 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)
```

#모두 1

#weights은 1

RNN – HIHELLO예제 –Data creation

```
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))
```

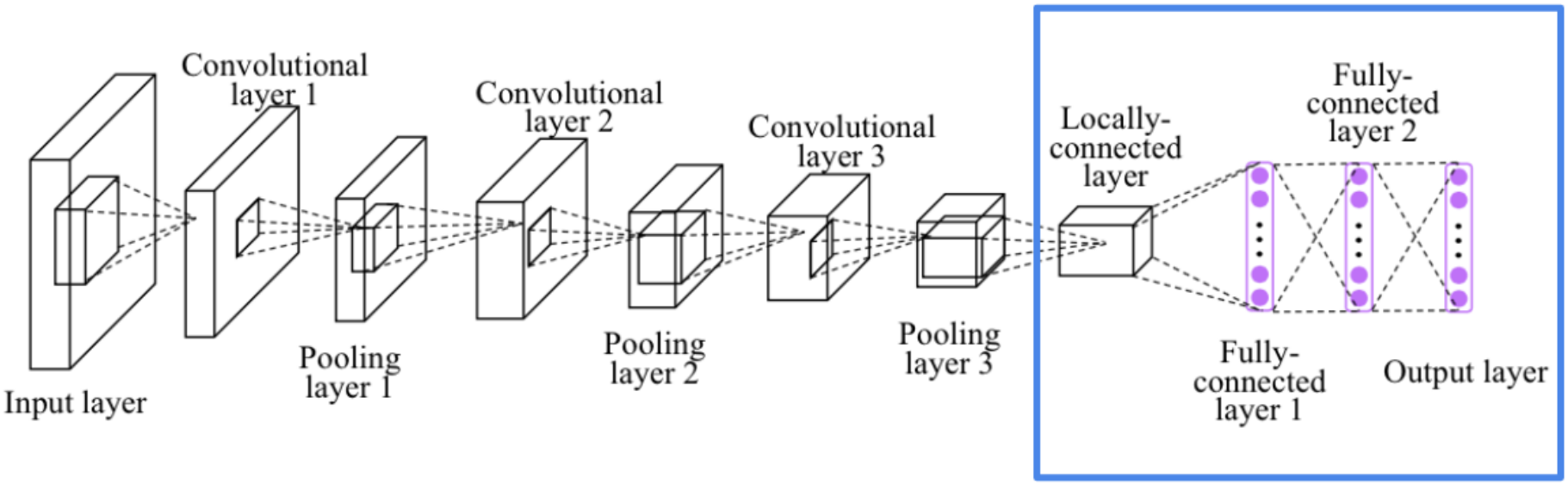
```
9 loss: 0.64819854 Prediction: yf you want you
10 loss: 0.48782146 Prediction: yf you want you
11 loss: 0.35372406 Prediction: if you want you
12 loss: 0.25655434 Prediction: if you want you
13 loss: 0.1855688 Prediction: if you want you
14 loss: 0.12913868 Prediction: if you want you
15 loss: 0.088434435 Prediction: if you want you
16 loss: 0.061431967 Prediction: if you want you
```

RNN – 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 – Really long sentence



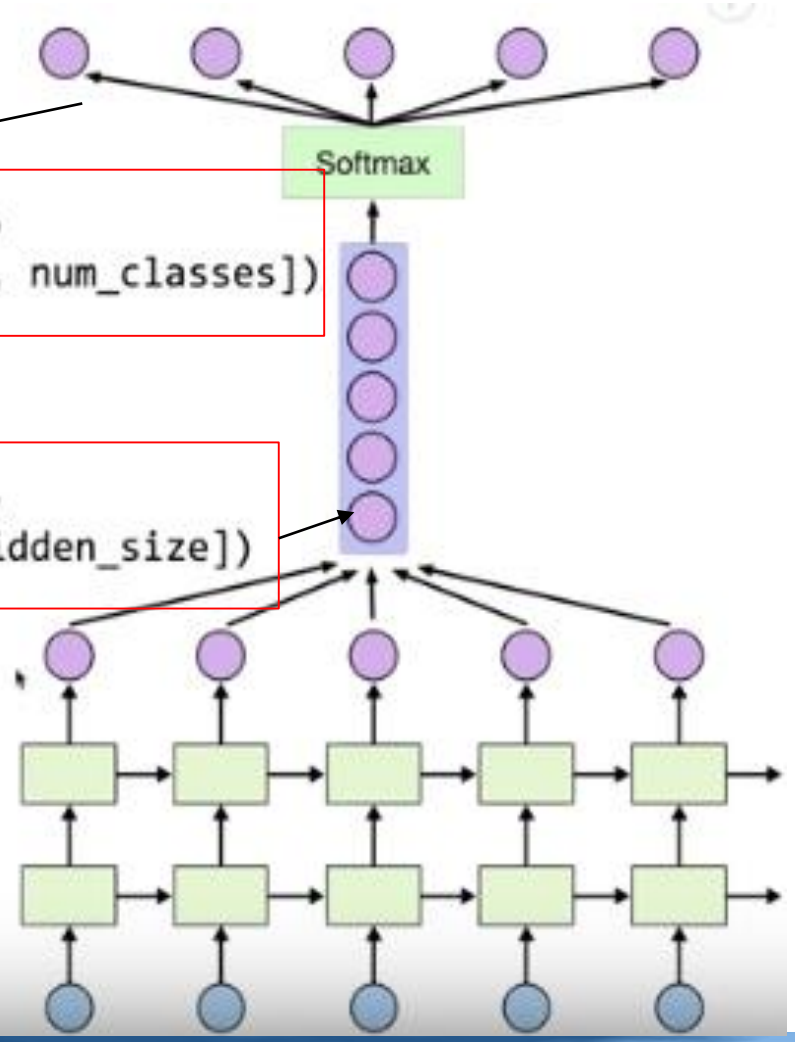
RNN – Really long sentence

Softmax

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

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

소프트맥스 입력



RNN – Really long sentence

Softmax

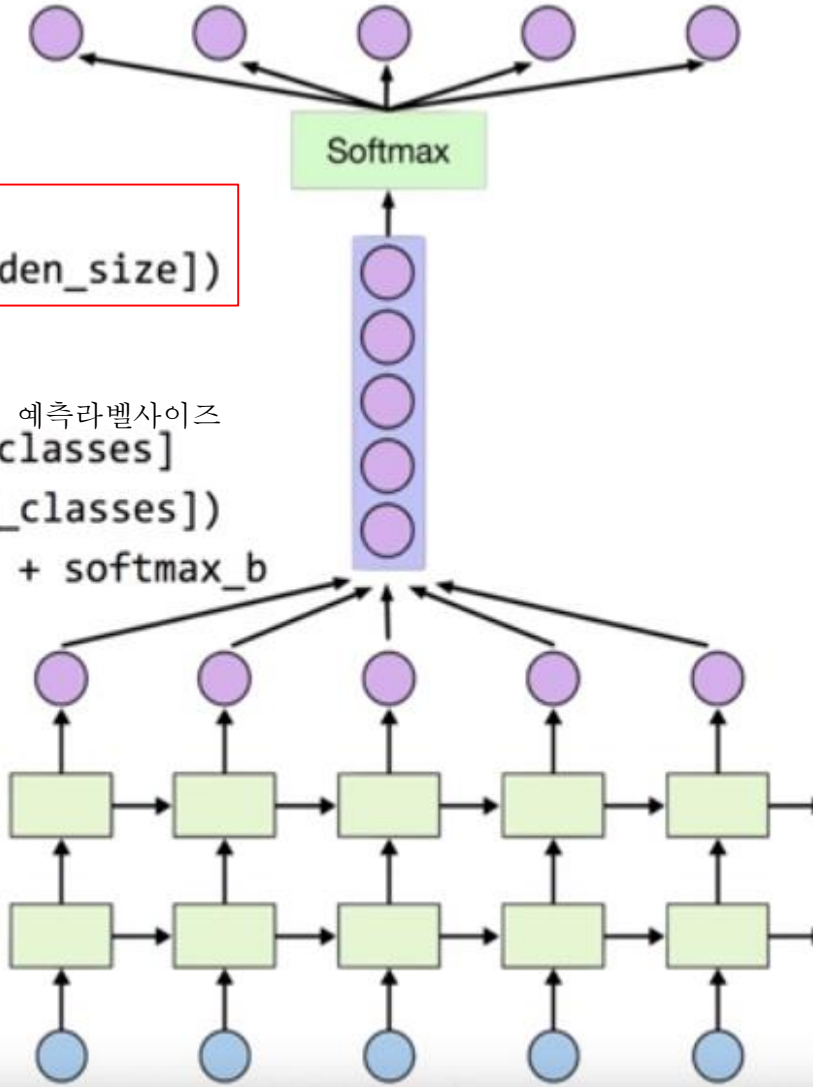
RNN->Softmax입력

```
# (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])
```

입력사이즈

예측라벨사이즈

최종output



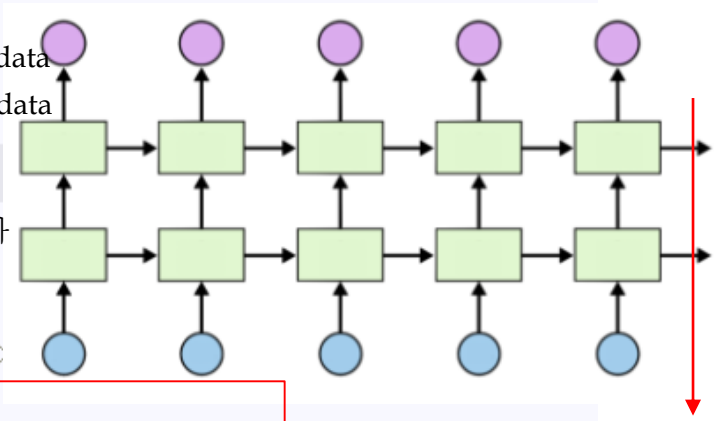
RNN – Really long sentence

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn
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))   #중복없는 알파벳 집합 만들고 리스트변환 print(char_set)
char_dic = {w: i for i, w in enumerate(char_set)} #알파벳을 key, 인텍스를 value로 하는 딕셔너리 생성
data_dim = len(char_set)
hidden_size = len(char_set)                    #각 셀의 출력크기
num_classes = len(char_set)                    #분류 총수
sequence_length = 10  # Any arbitrary number   #1개 시퀀스의 길이(시계열데이터의 입력갯수)
learning_rate = 0.1                            #학습률
dataX = []  #입력 시퀀스를 저장하기 위한 배열
dataY = []  #출력 시퀀스를 저장하기 위한 배열
for i in range(0, len(sentence) - sequence_length):
    x_str = sentence[i:i + sequence_length]                  #시퀀스 길이만큼을 문자열
    y_str = sentence[i + 1: i + sequence_length + 1]         #입력보다 1칸 오른쪽에 시작하는 시퀀스길이만큼의 문자열
    print(i, x_str, '->', y_str)                             #print("==➔",x,"--➔"y)

    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)  #array만듬
    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 – Really long sentence

```
batch_size = len(dataX) #전체 169
X = tf.placeholder(tf.int32, [None, sequence_length]) #Xdata
Y = tf.placeholder(tf.int32, [None, sequence_length]) #Ydata
# One-hot encoding
X_one_hot = tf.one_hot(X, num_classes)  #전체분류개수1개차원추가
print(X_one_hot)  # check out the shape
# Make a lstm cell with hidden_size (each unit output vec
def lstm_cell():
    cell = rnn.BasicLSTMCell(hidden_size, state_is_tuple=True)
    return cell

#2층의 stacked RNN생성
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  #전체분류개수(Fully Connected)
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])
```



#RNN Cell들을 연결
#hidden size는 각 RNN Cell출력크기

RNN – Really long sentence

```
# All weights are 1 (equal weights)
weights = tf.ones([batch_size, sequence_length])   #sequence loss를 구할때 모든 sequence의 가중치 동일하게 1
sequence_loss = tf.contrib.seq2seq.sequence_loss(
    logits=outputs, targets=Y, weights=weights)
mean_loss = tf.reduce_mean(sequence_loss)          #sequence loss 구한다
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):                          #마지막result sum
    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='')  #첫줄은 sequence_lenth만큼 출력
    else:
        print(char_set[index[-1]], end='')                    #두번째 줄부터는 마지막1줄만 출력
```

```
499 169 n the sea. 0.22882889
t you want to build a ship, don't drum up people together to collect wood and don't
```

. 구성비가 다른 불균형 데이터

. 원소가 9개인 numpy 배열 생성
. Y값은 0이 3개, 1은 6개 => 비율 1:2(불균형데이터)
. Stratified가 붙은 클래스를 이용하거나 stratify옵션을 켜야 한다.

```
import numpy as np
seed = 0
np.random.seed(seed)
# 원소가 9개인 numpy 배열을 생성한다
# Y값은 0이 3개, 1은 6개로 비율은 1:2이다 (불균형데이터)
X = np.array([-5, -3, -1, 1, 3, 5, 7, 9, 11])
Y = np.array([0, 0, 0, 1, 1, 1, 1, 1, 1])
splits = 3
```

. 구성비가 다른 불균형 데이터

. StratifiledKFold
- Stratified하게 트레이닝셋과 테스트셋으로 나눈다
- KFold기법에 의해 test에 선택된 인덱스는 겹치지 않도록 한다

```
from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=splits, shuffle=True, random_state=seed)

print(kfold)
print("=" * 100)
for train_index, test_index in kfold.split(X, Y):
    print("train Index:", train_index)
    print("test index:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]
    print("-" * 100)
```

. 구성비가 다른 불균형 데이터

. StratifiedShuffleSplit

- Stratified하게 트레이닝셋과 테스트셋으로 나눈다
- test에 선택된 인덱스는 겹쳐도 되며 splits개수만큼 추출한다

```
# StratifiedShuffleSplit
# Stratified하게 트레이닝셋과 테스트셋으로 나눈다
# test에 선택된 인덱스는 겹쳐도 되며 splits개수만큼 추출한다
from sklearn.model_selection import StratifiedShuffleSplit

shufflesplit = StratifiedShuffleSplit(n_splits=splits, random_state=seed, test_size=0.3)

print(shufflesplit)
print("=" * 100)
for train_index, test_index in shufflesplit.split(X, Y):
    print("train Index:", train_index)
    print("test index:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]
    print("-" * 100)
```

. Train_test_split
- Stratified하게 트레이닝셋과 테스트셋으로 나누다(1회만실시)

```
from sklearn.model_selection import train_test_split

train_index, test_index = train_test_split(np.array(range(X.shape[0])), shuffle=True, stratify=Y,
                                           test_size=0.3, random_state=seed)
print("train Index:", train_index)
print("test index:", test_index)
X_train, X_test = X[train_index], X[test_index]
Y_train, Y_test = Y[train_index], Y[test_index]
print("-" * 100)
```

. 구성비가 다른 불균형 데이터

```
StratifiedKFold(n_splits=3, random_state=0, shuffle=True)
================================================================
train Index: [0 1 3 4 6 7]
test index: [2 5 8]
----------------------------------------------------------------
train Index: [0 2 3 5 7 8]
test index: [1 4 6]
----------------------------------------------------------------
train Index: [1 2 4 5 6 8]
test index: [0 3 7]
----------------------------------------------------------------
StratifiedShuffleSplit(n_splits=3, random_state=0, test_size=0.3,
            train_size=None)
================================================================
train Index: [1 5 4 2 7 6]
test index: [3 8 0]
----------------------------------------------------------------
train Index: [8 6 5 7 1 0]
test index: [3 2 4]
----------------------------------------------------------------
train Index: [8 7 4 2 1 5]
test index: [6 3 0]
----------------------------------------------------------------
train Index: [1 5 4 2 7 6]
test index: [3 8 0]
```