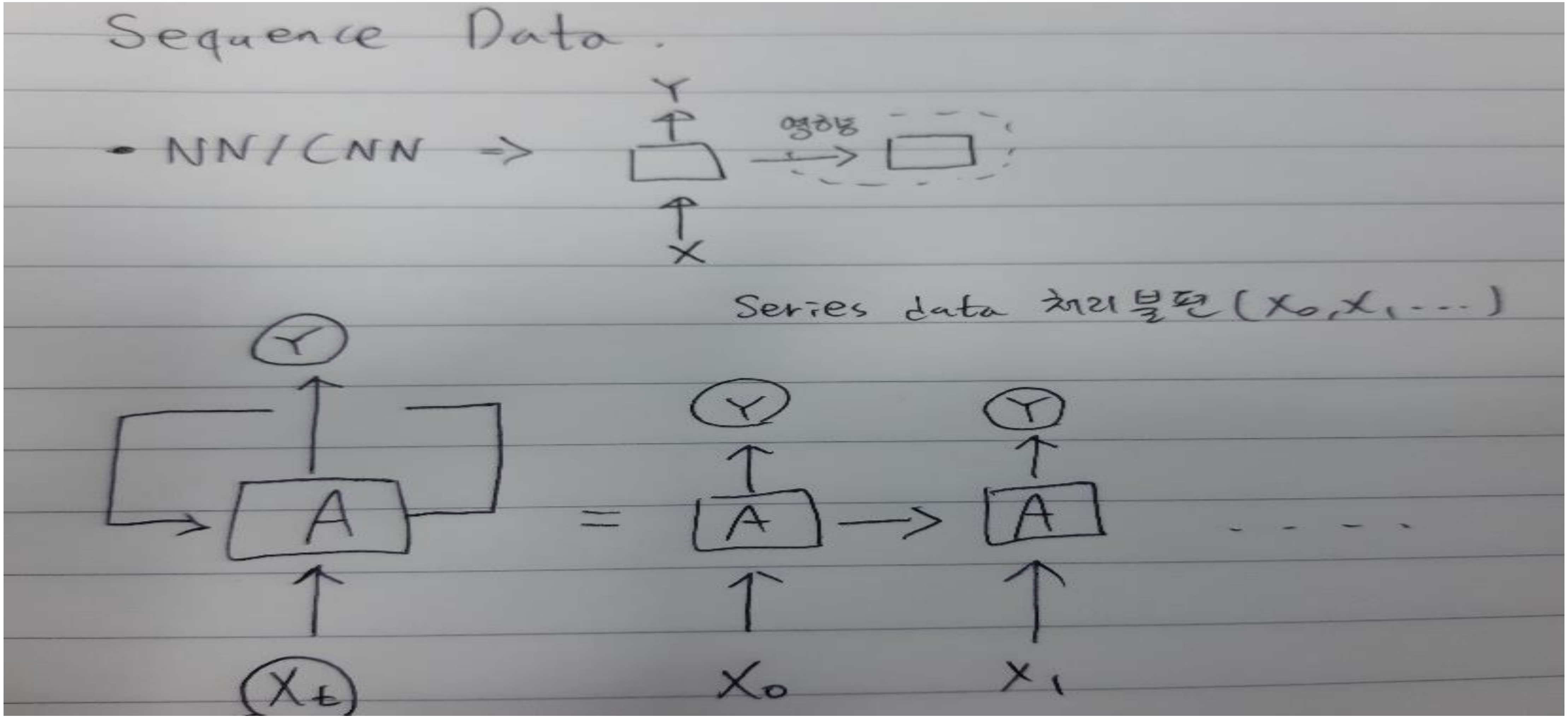
Recurrent Neural Networks

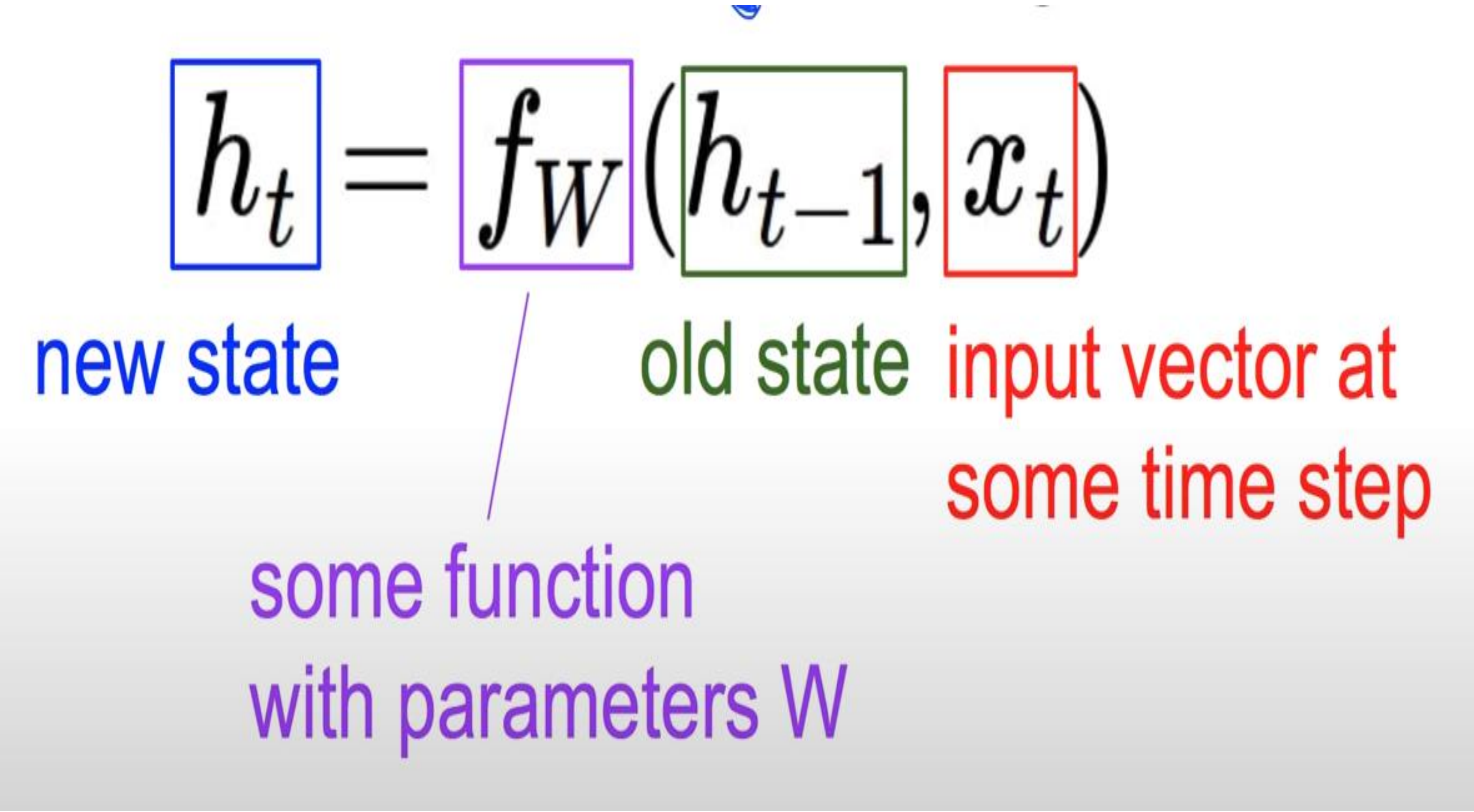
Sequence Data



Sequence Data.

- NN/CNN ->

Series data 처리불편 (x_0, x_1, ...)



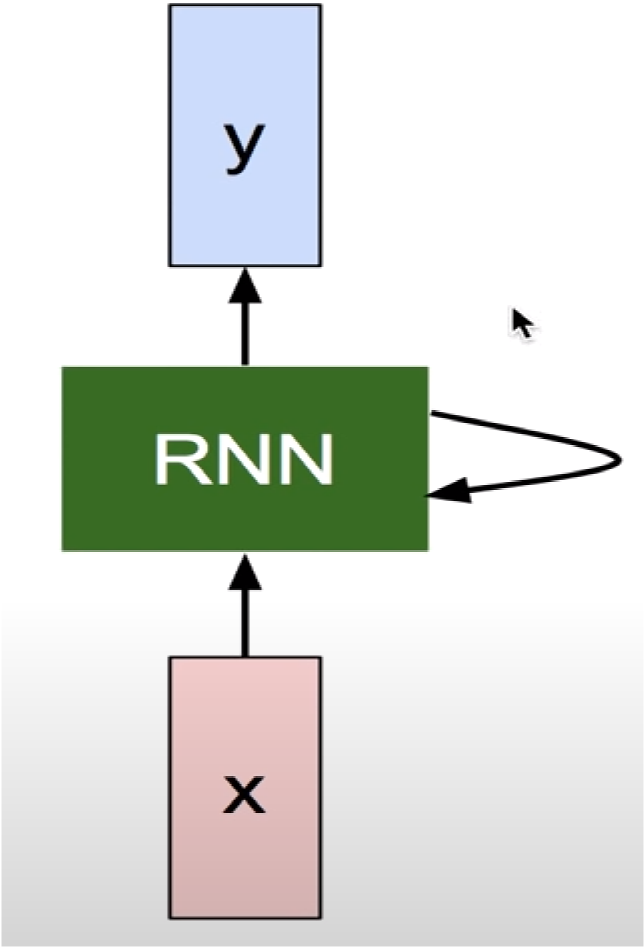
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

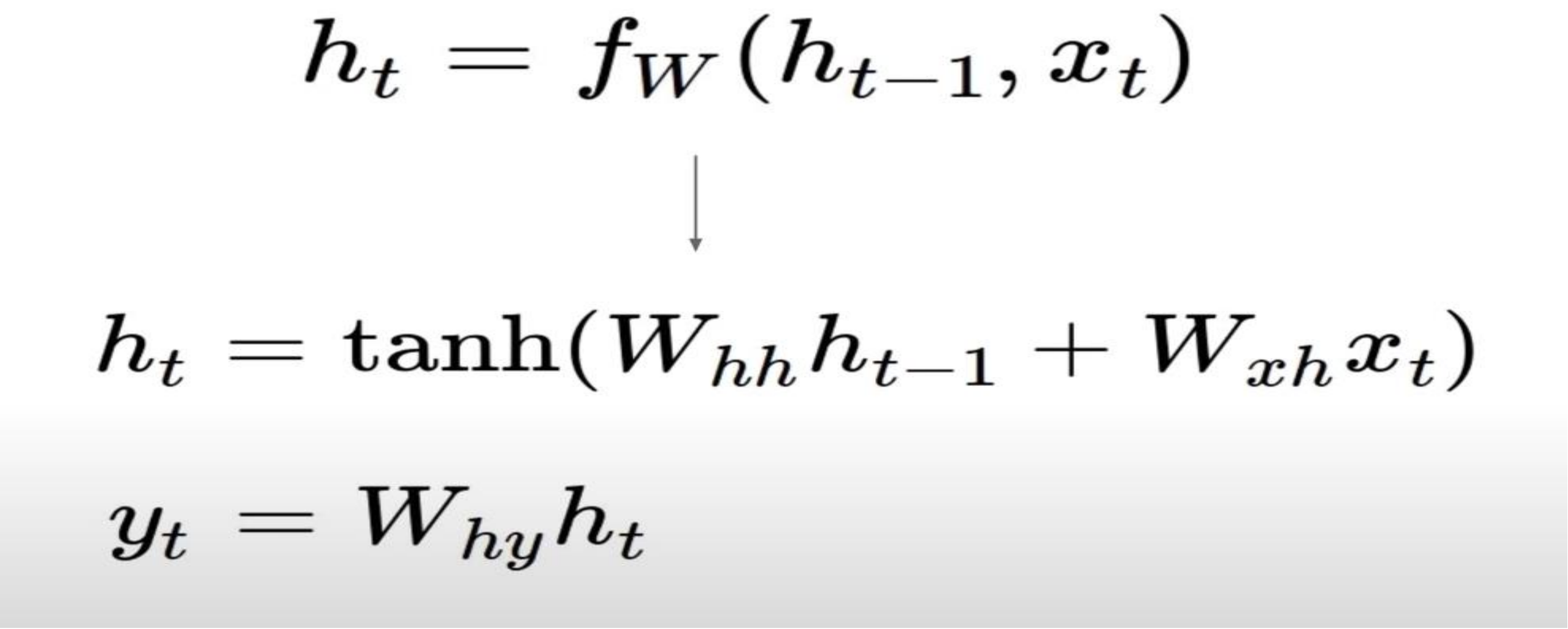
input vector at some time step

some function with parameters W



RNN에 적용되는 fw가 모두 동일

(Vanilla) RNN

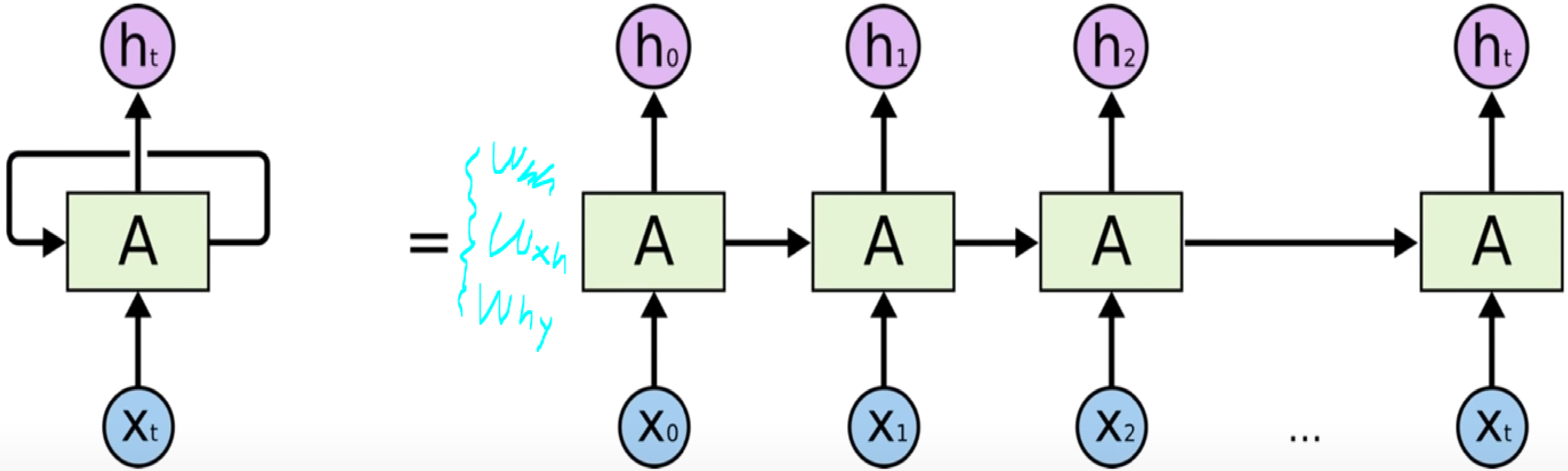


$$h_t = f_W(h_{t-1}, x_t)$$

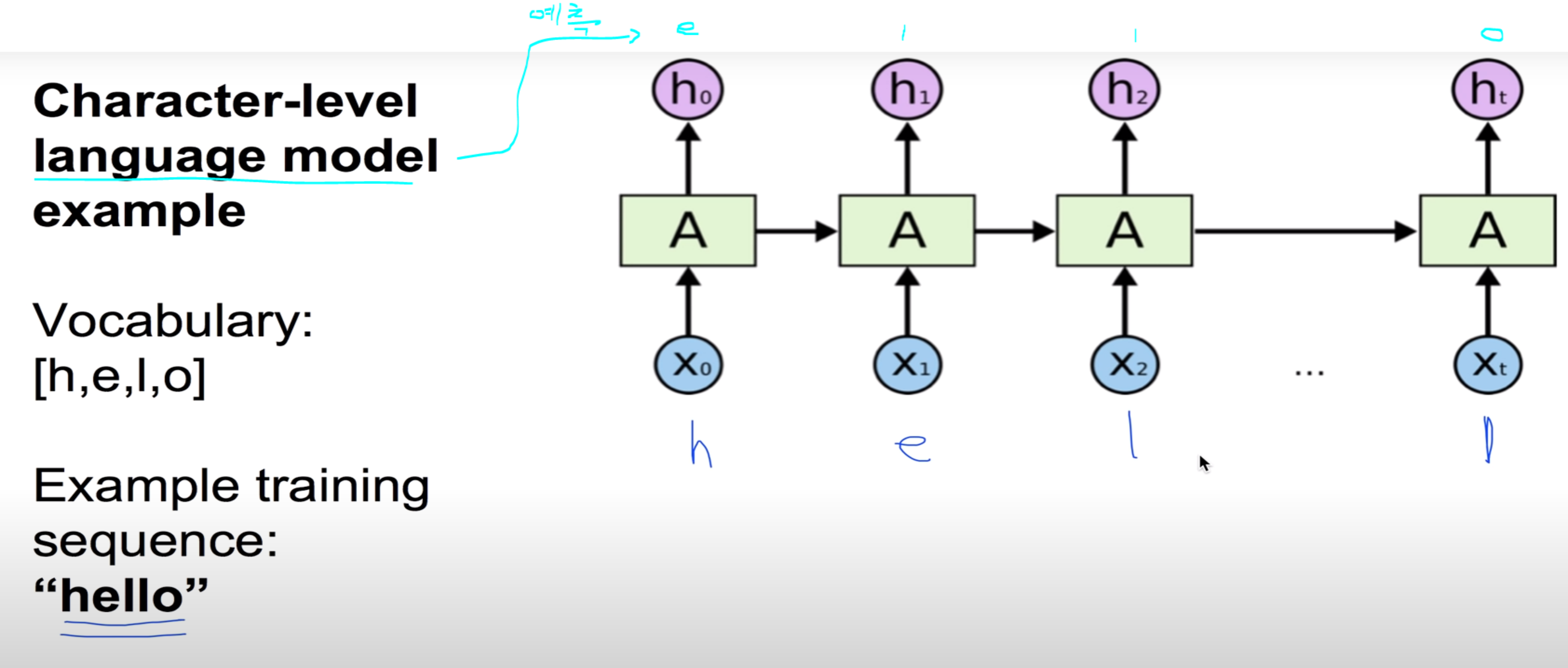
$$\downarrow$$

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

$$y_t = W_{hy}h_t$$



동일한 가중치를 갖고 학습



Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**

이전 단어를 보고 다음 단어를 예측하기

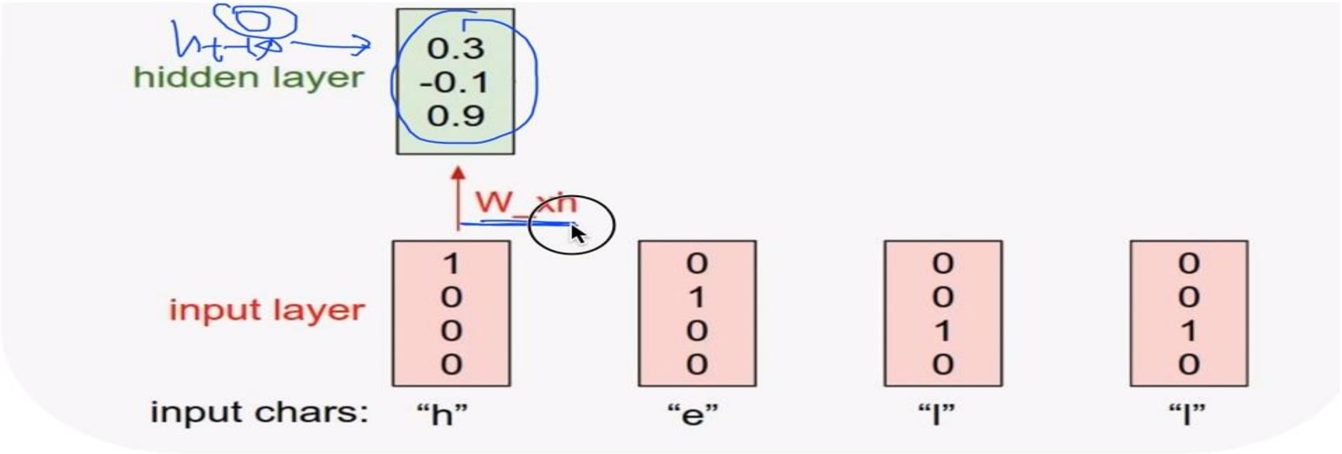
1Stage

Character-level language model example

Vocabulary:
[h,e,l,o]

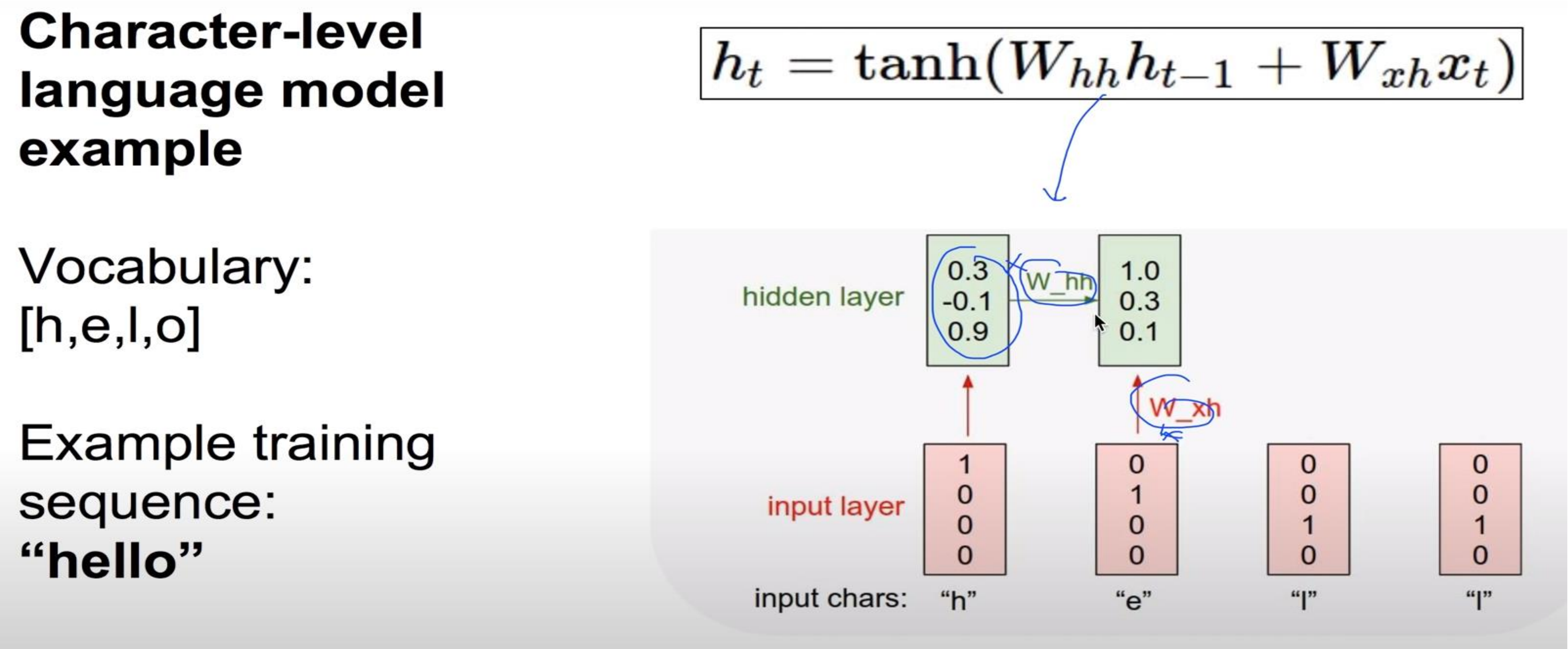
Example training sequence:
“hello”

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



Ht-1은 값이 없으므로 0으로 초기화한다.

2Stage



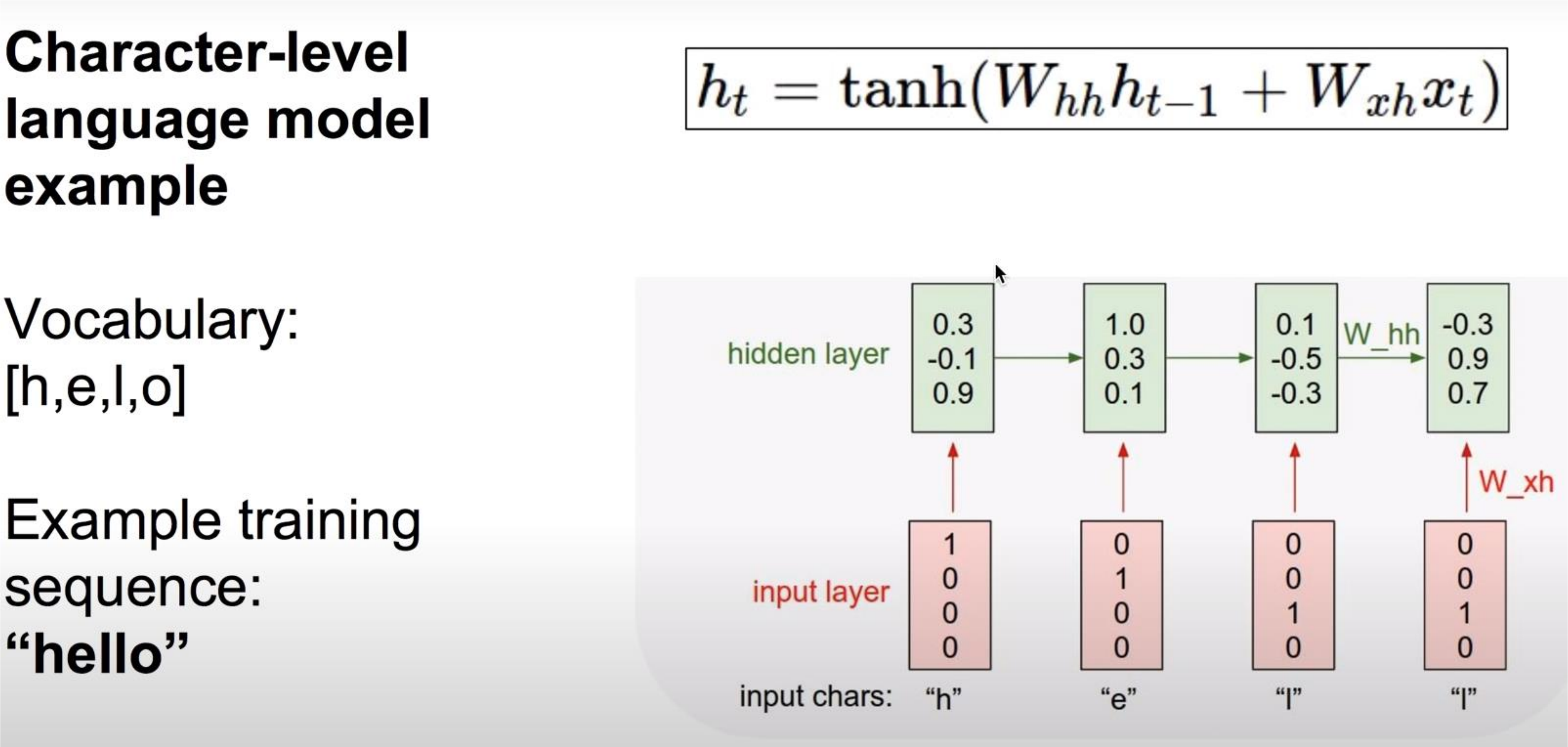
Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

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

4Stage



Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

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

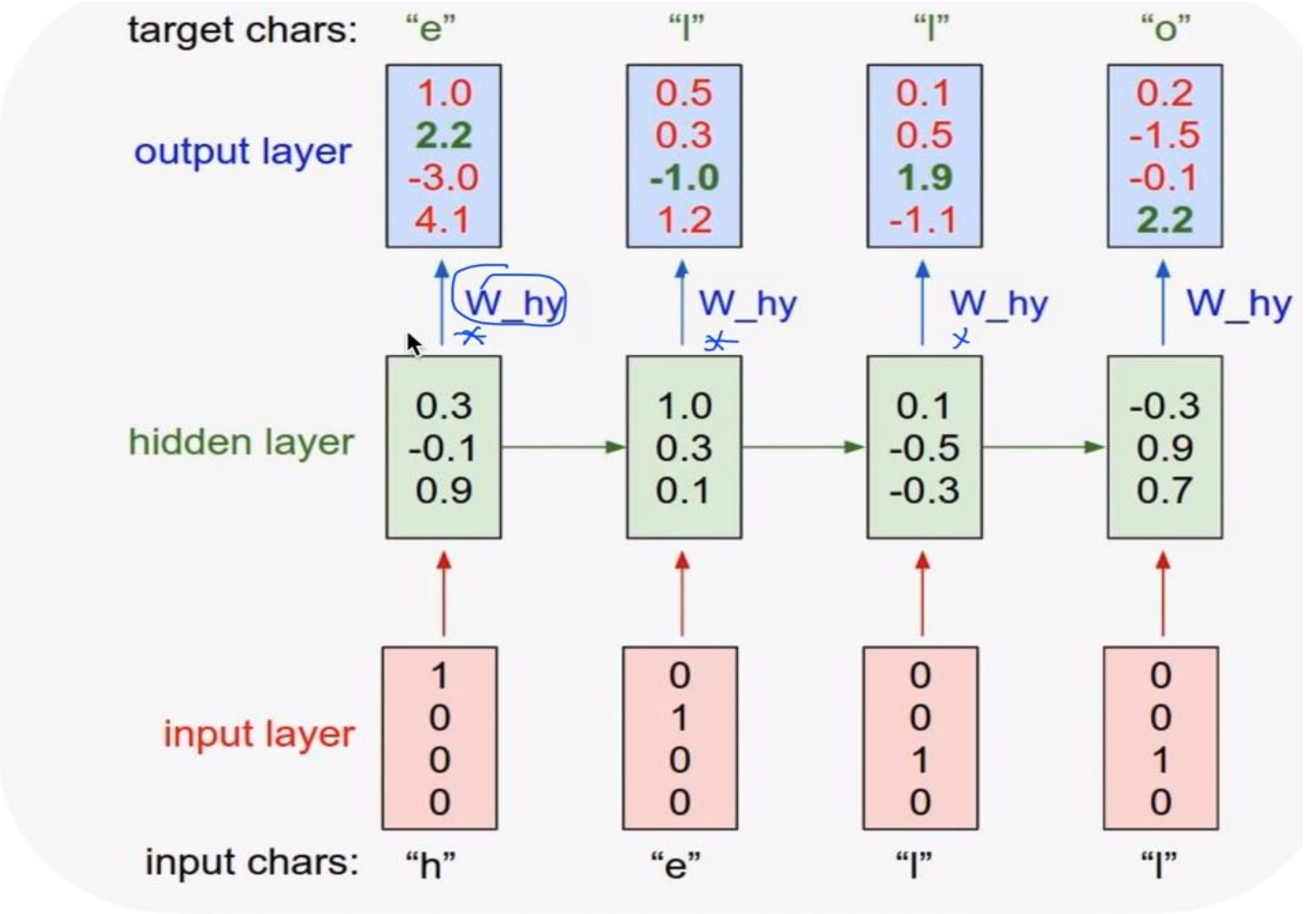
Output(Y)

Character-level language model example

$$y_t = \underline{W_{hy} h_t}$$

Vocabulary: [h,e,l,o]

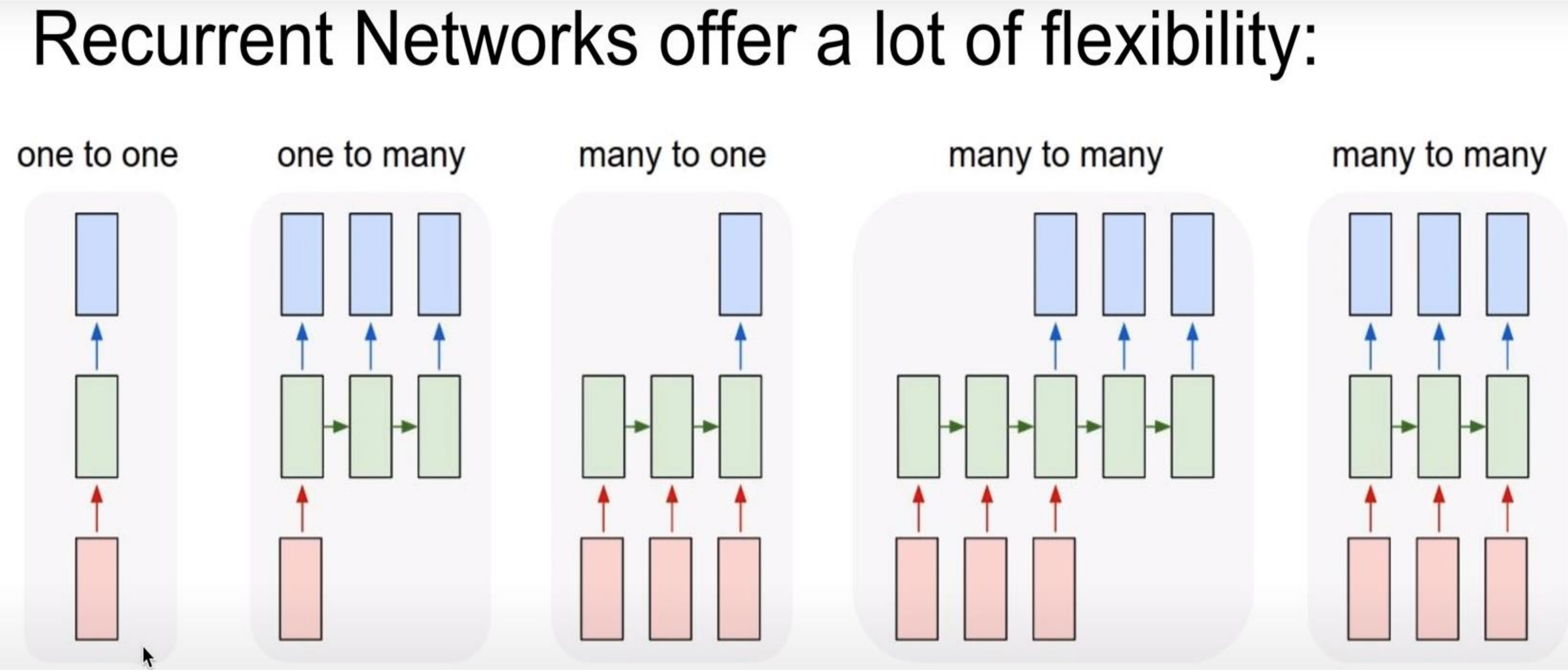
Example training sequence: **“hello”**



Outlayer : Softmax 함수를 통해 학습시키기

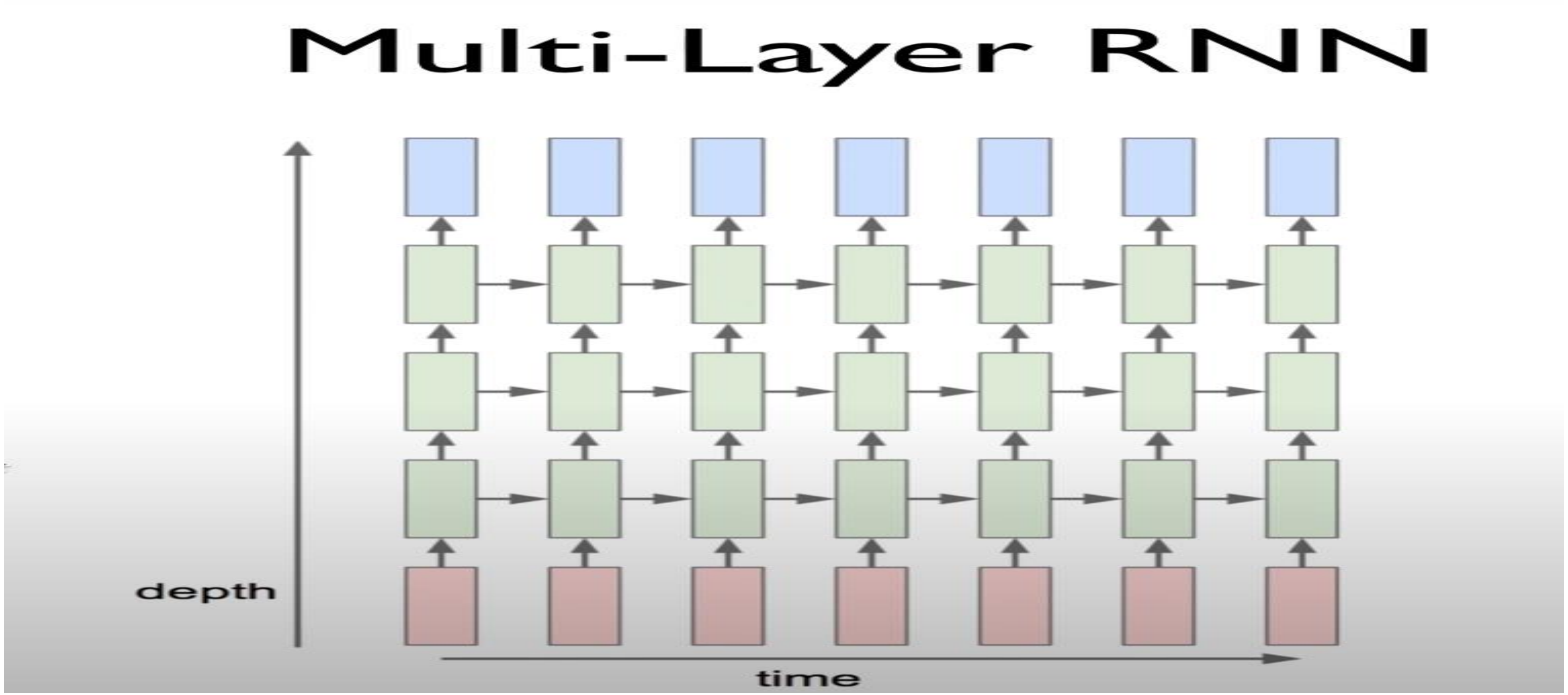
RNN applications

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



Recurrent Networks offer a lot of flexibility:

다양한 형태로 구성 가능

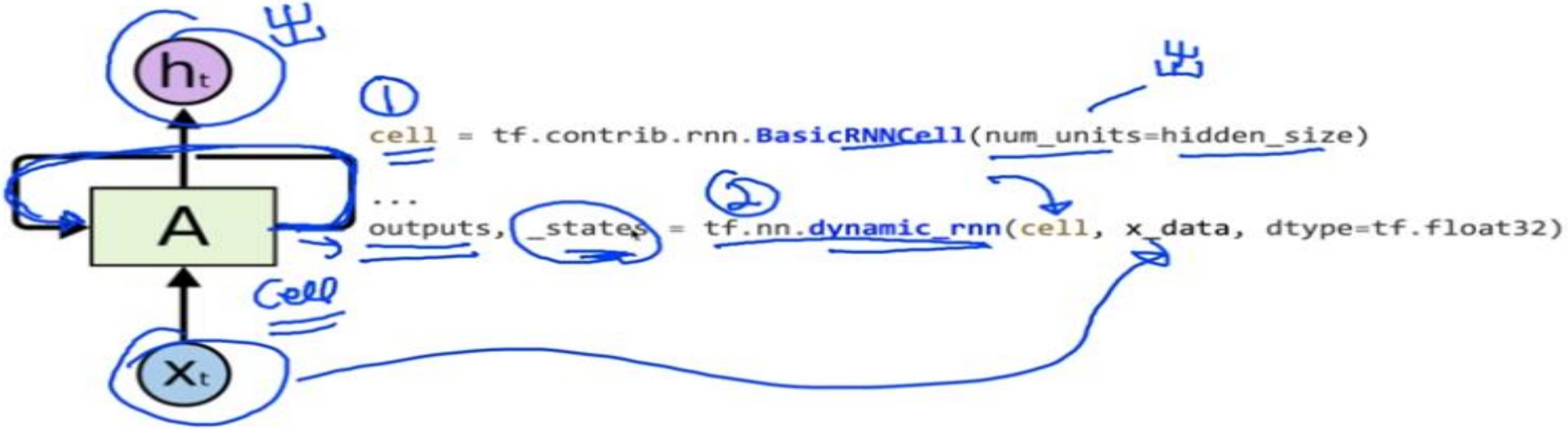


Multi-Layer RNN

다양한 형태로 구성 가능

기타 모델 : LSTM, GRU 등

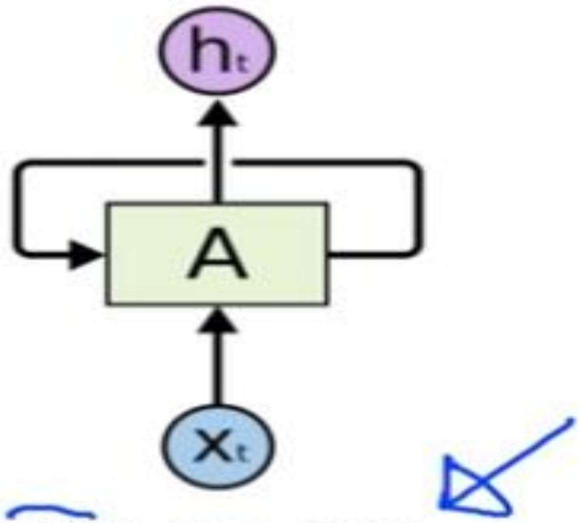
RNN - Basic



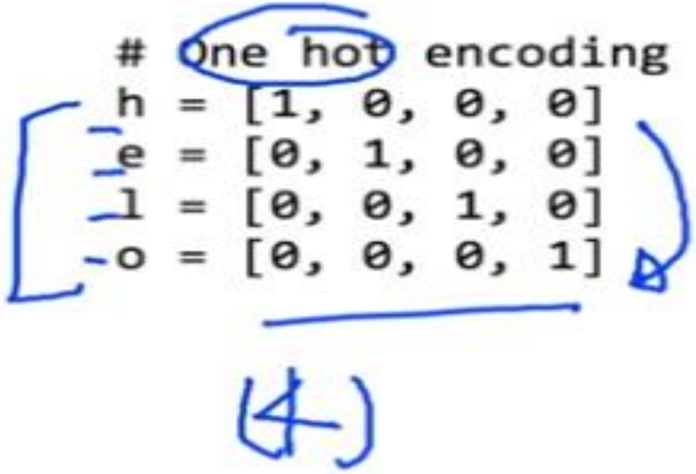
```
cell = tf.contrib.rnn.BasicRNNCell(num_units=hidden_size)
...
outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32)
```

1. Cell을 생성한다. RNN, LSTM 등
   1. 출력 size 결정한다.
2. Cell을 만든 것에 입력을 주고 출력을 얻어낸다.

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



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

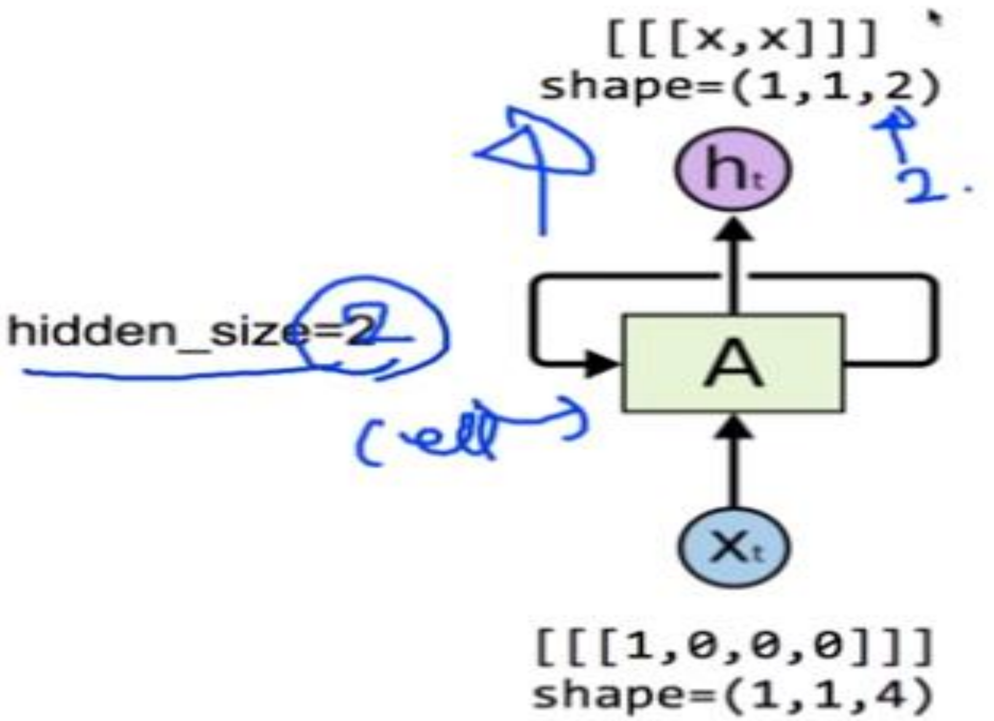


```
# One hot encoding
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
```

(4)

입력은 각 단어를 구분하기 위해 One hot encoding을 사용하였고,
4개이므로 shape의 마지막인자는 4가 된다.

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



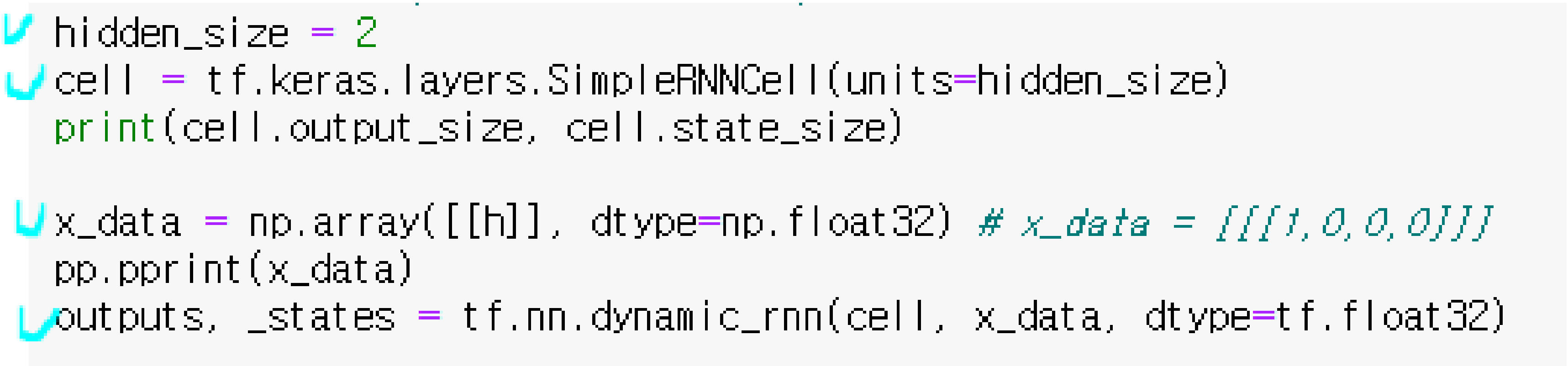
[[[x,x]]]
shape=(1,1,2)

hidden_size=2

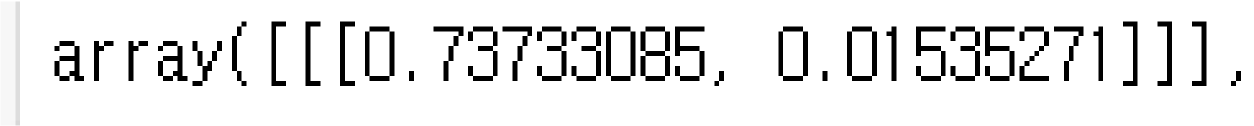
[[[1,0,0,0]]]
shape=(1,1,4)

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

출력은 hidden_size의 개수와 같다. -> One hot encoding 개수

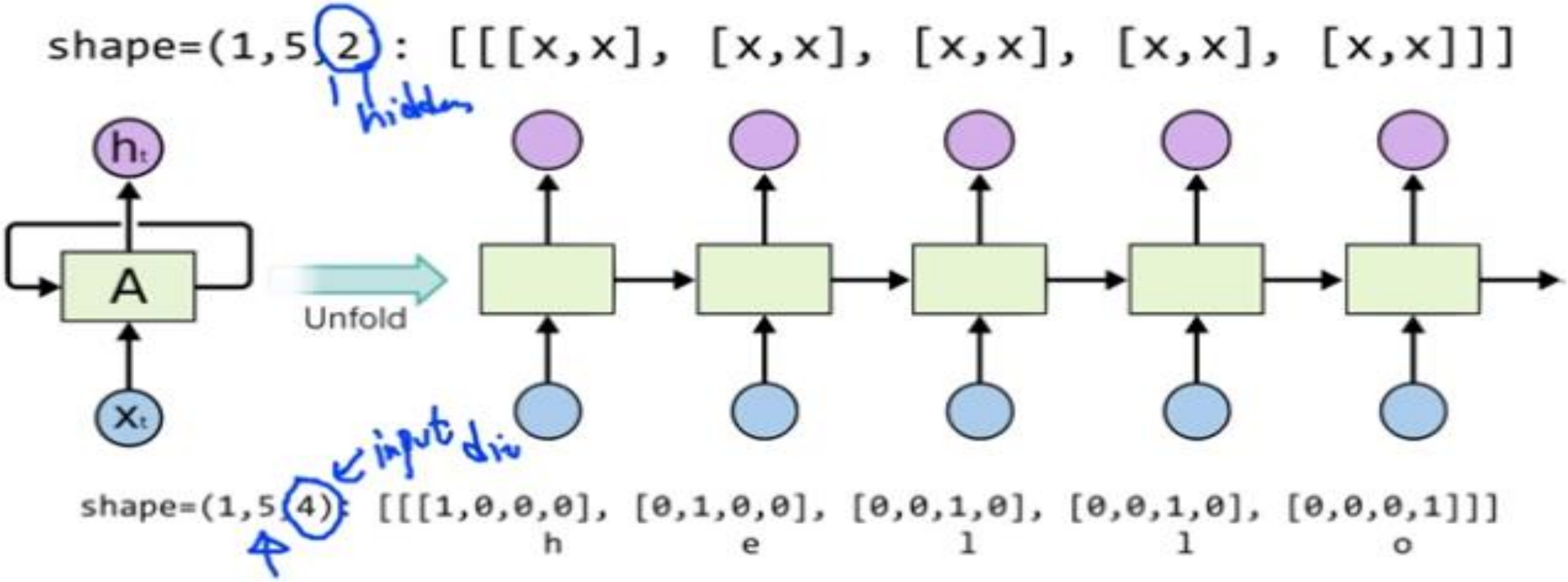


```
hidden_size = 2
cell = tf.keras.layers.SimpleRNNCell(units=hidden_size)
print(cell.output_size, cell.state_size)

x_data = np.array([[h]], 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)
```



```
array([[[0.73733085, 0.01535271]]],
```

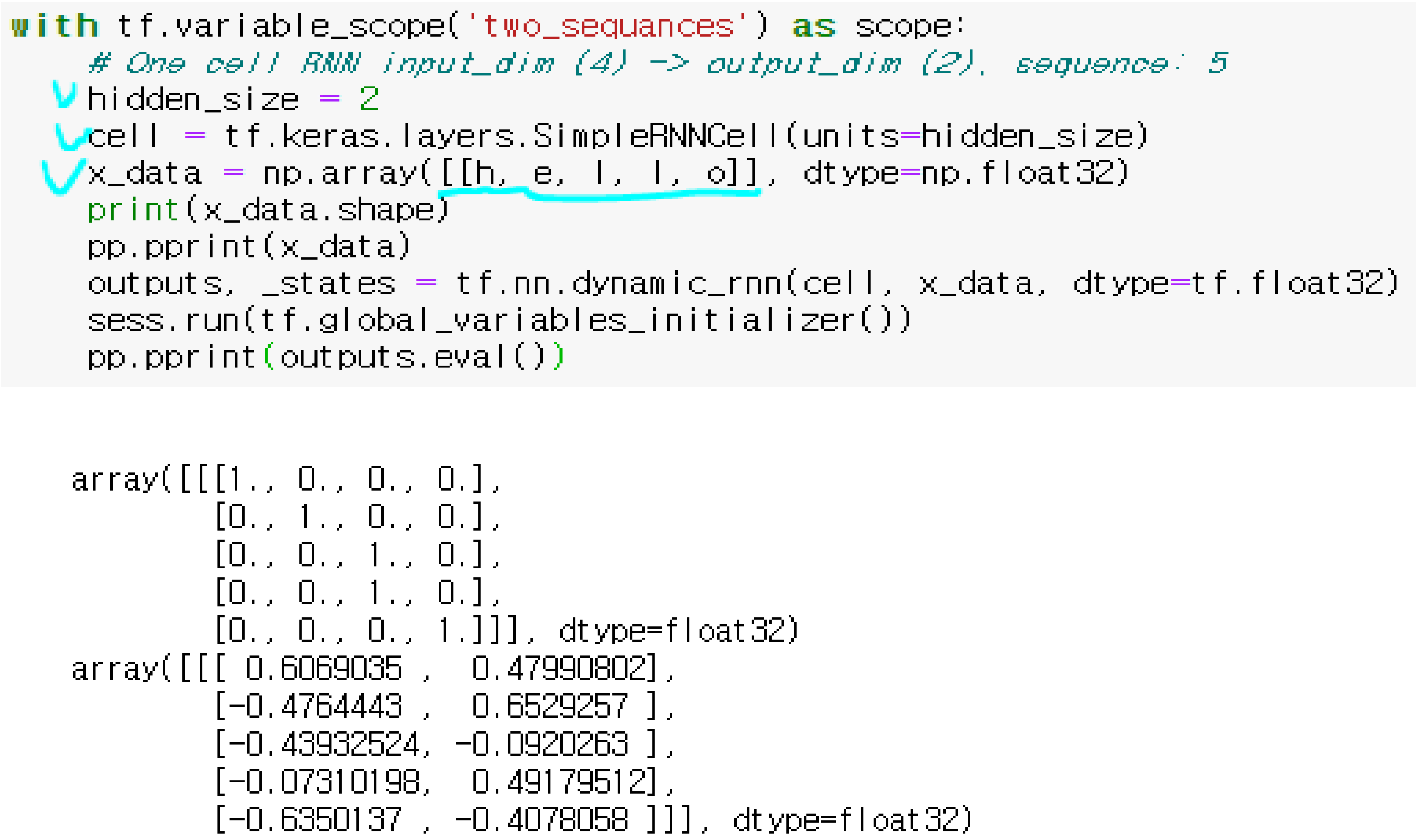
Hidden_size, cell 생성한 뒤 input data를 주고 output 확인
결과가 hidden_size 개수와 동일함을 확인



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

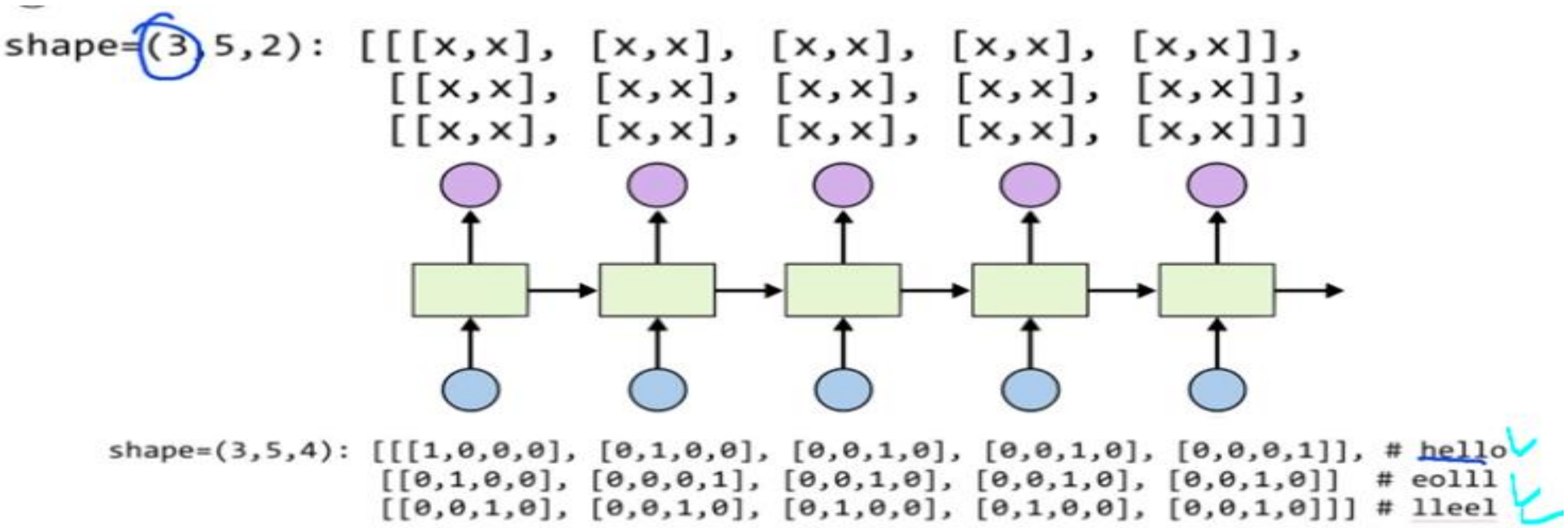
```
shape=(1,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]
                   h           e           l           l           o
```

두번째 인자 : 몇 개의 sequence data를 생성할 것인지, output으로 나오는 sequence data개수와 동일함



```
with tf.variable_scope('two_sequances') as scope:
    # One cell RNN input_dim (4) -> output_dim (2). sequence: 5
    hidden_size = 2
    cell = tf.keras.layers.SimpleRNNCell(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())
```

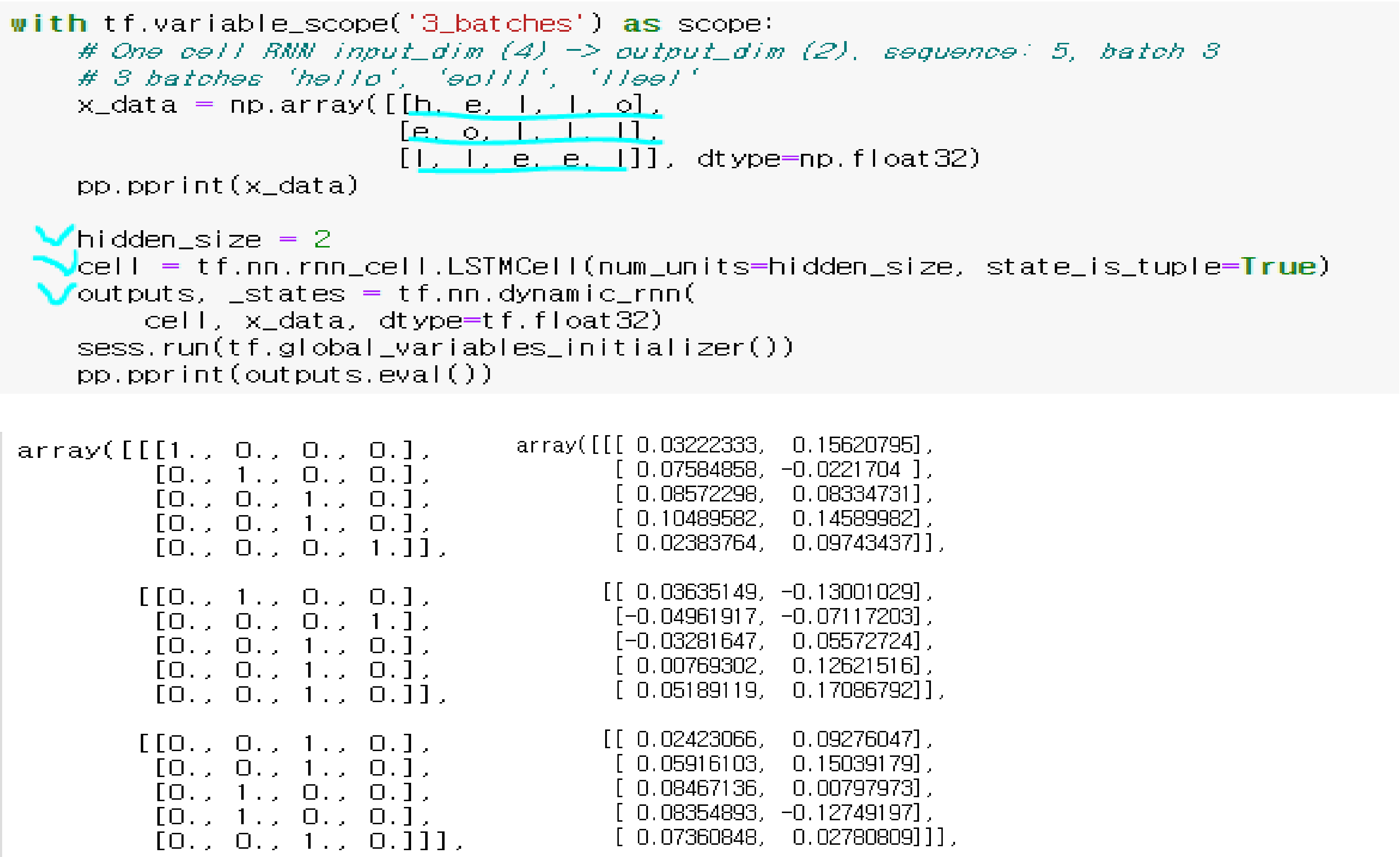
```
array([[[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]]], dtype=float32)
array([[[ 0.6069035 ,  0.47990802],
        [-0.4764443 ,  0.6529257 ],
        [-0.43932524, -0.0920263 ],
        [-0.07310198,  0.49179512],
        [-0.6350137 , -0.4078058 ]]], dtype=float32)
```



```
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
```

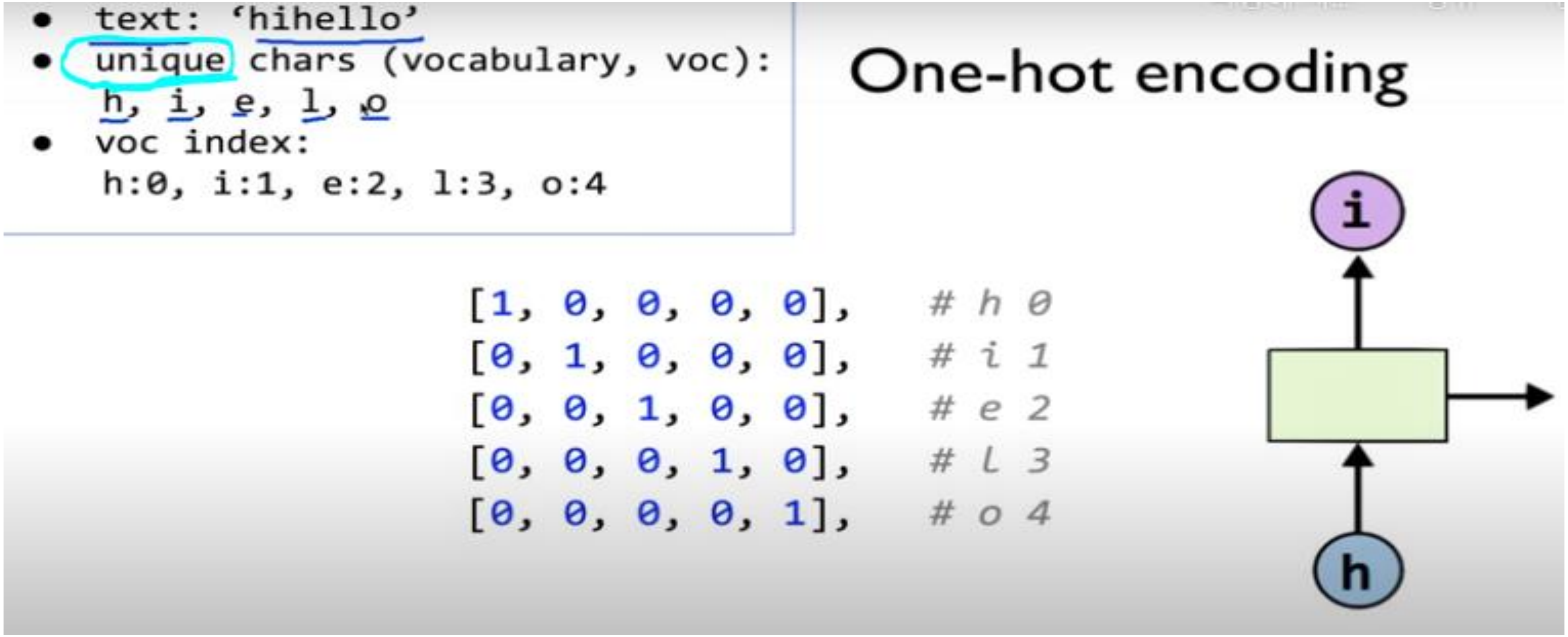
첫번째 인자 : 몇 개의 data가 들어오는지(=batch), output으로 나오는 batch 개수와 동일함



```
with tf.variable_scope('3_batches') as scope:
    # 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)

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

```
array([[[1., 0., 0., 0.],          array([[[ 0.03222333,  0.15620795],
        [0., 1., 0., 0.],                 [ 0.07584858, -0.0221704 ],
        [0., 0., 1., 0.],                 [ 0.08572298,  0.08334731],
        [0., 0., 1., 0.],                 [ 0.10489582,  0.14589982],
        [0., 0., 0., 1.]],                [ 0.02383764,  0.09743437]],

       [[0., 1., 0., 0.],                [[ 0.03635149, -0.13001029],
        [0., 0., 0., 1.],                 [-0.04961917, -0.07117203],
        [0., 0., 1., 0.],                 [-0.03281647,  0.05572724],
        [0., 0., 1., 0.],                 [ 0.00769302,  0.12621516],
        [0., 0., 1., 0.]],                [ 0.05189119,  0.17086792]],

       [[0., 0., 1., 0.],                [[ 0.02423066,  0.09276047],
        [0., 0., 1., 0.],                 [ 0.05916103,  0.15039179],
        [0., 1., 0., 0.],                 [ 0.08467136,  0.00797973],
        [0., 1., 0., 0.],                 [ 0.08354893, -0.12749197],
        [0., 0., 1., 0.]]],               [ 0.07360848,  0.02780809]]],
```

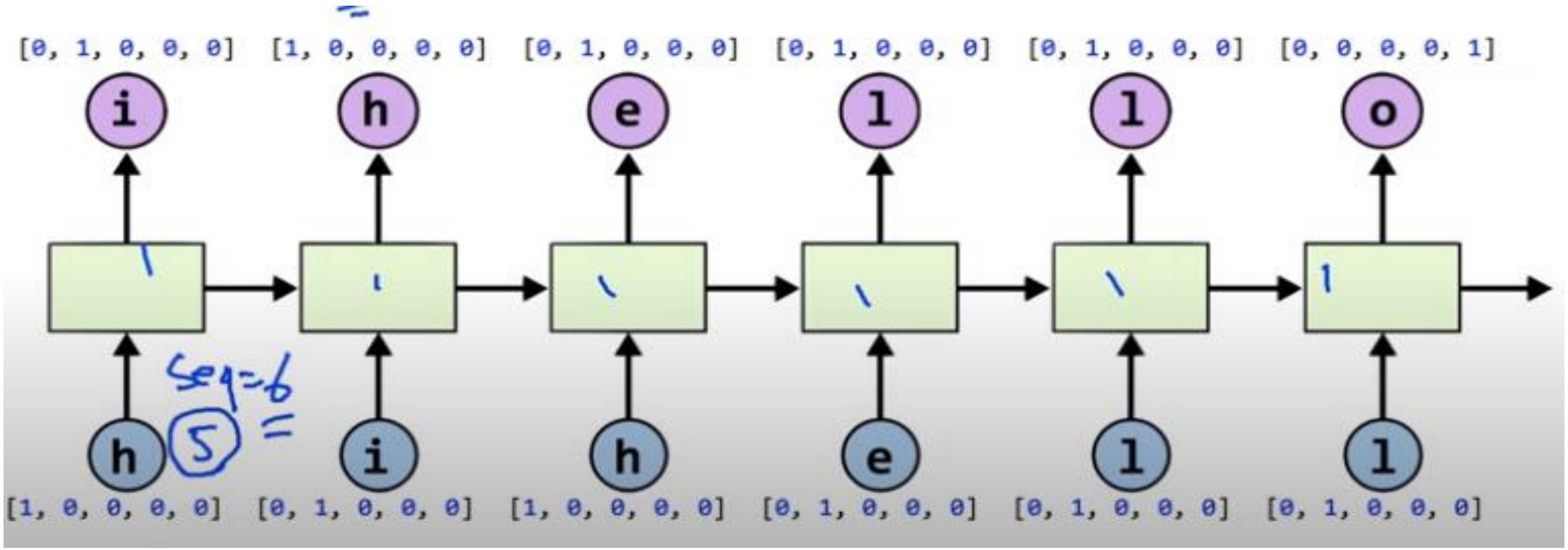
RNN – Hi Hello Training



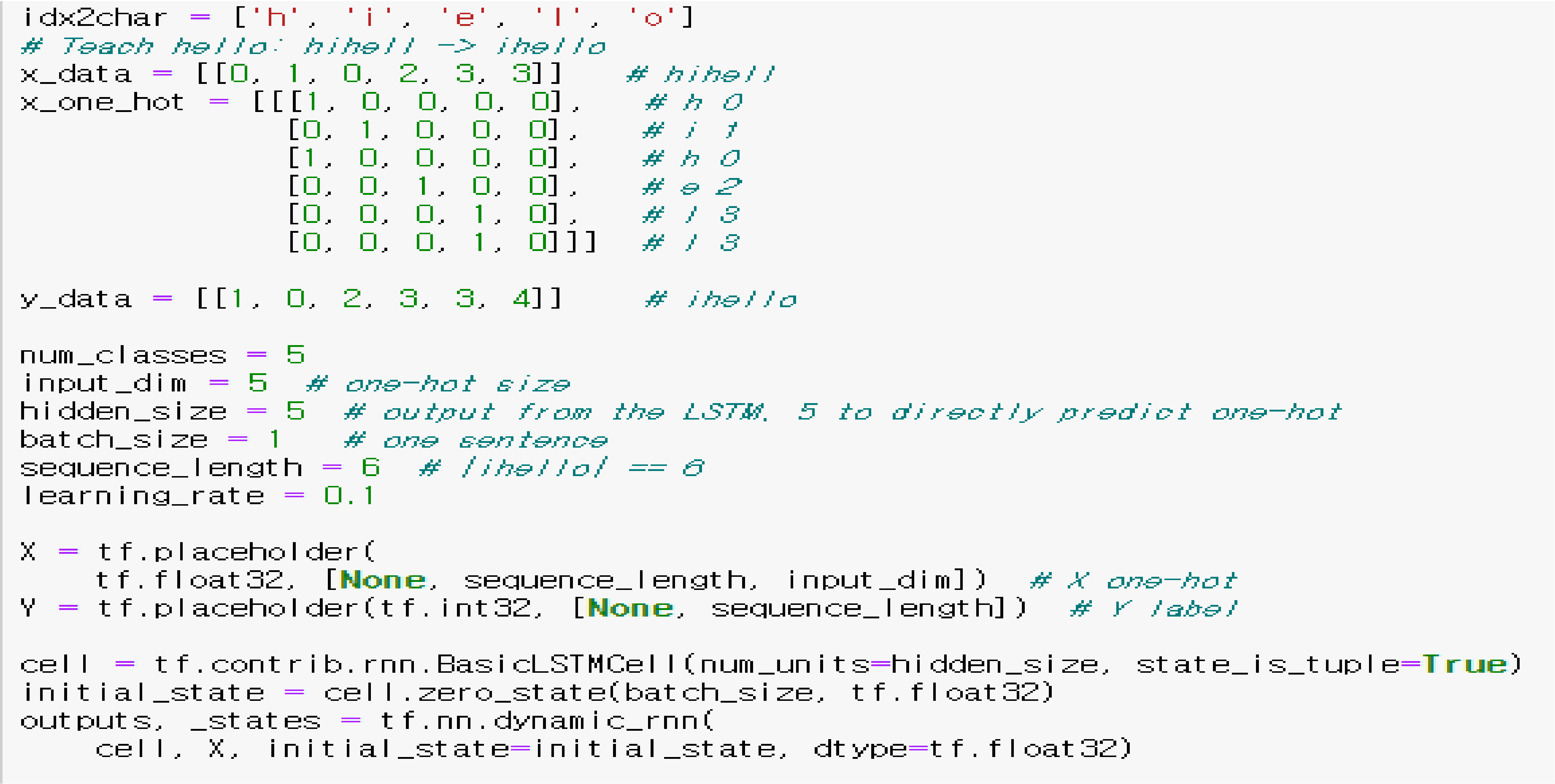
- text: ‘hihello’
- unique chars (vocabulary, voc):
  h, i, e, l, o
- voc index:
  h:0, i:1, e:2, l:3, o:4

One-hot encoding

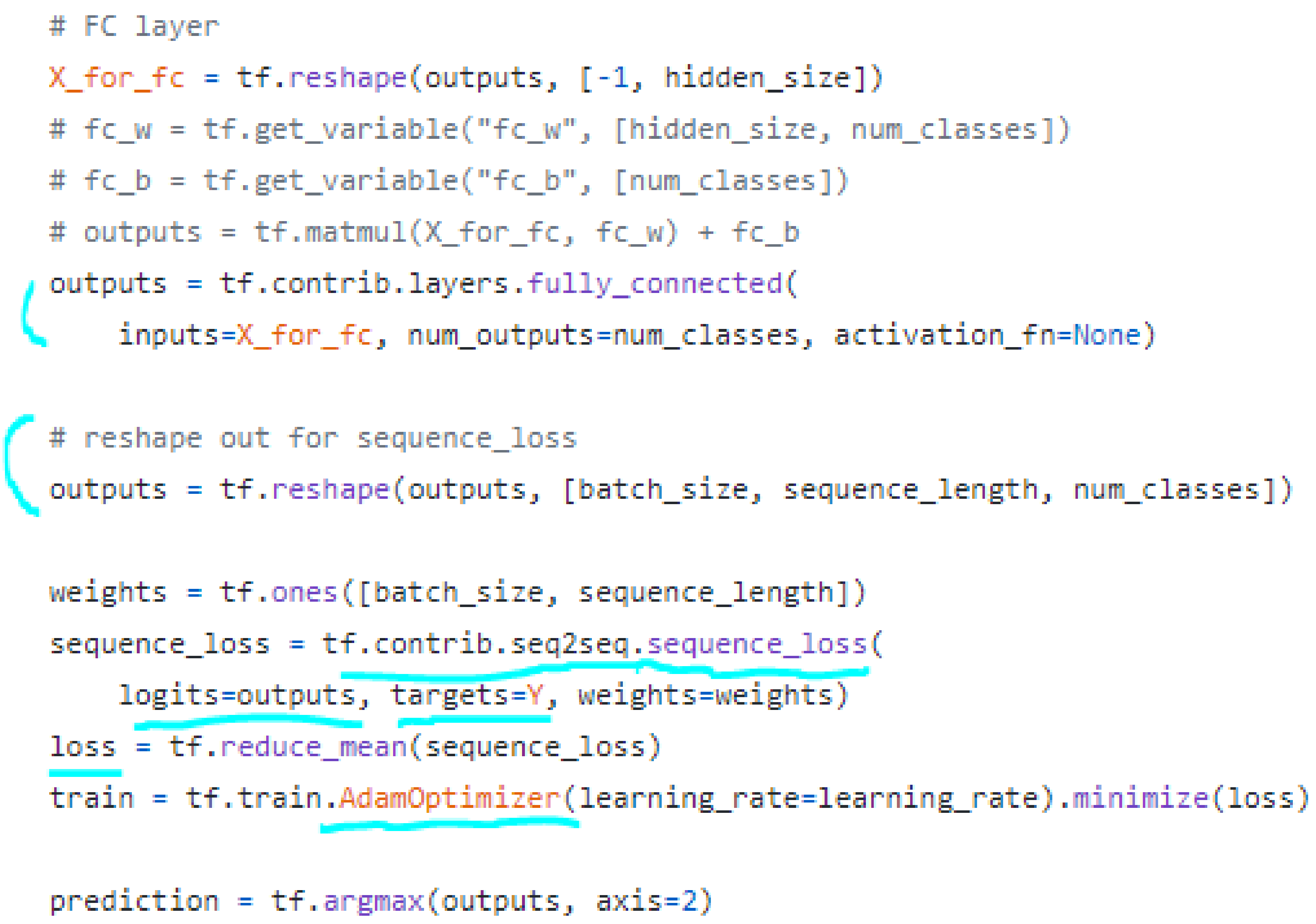
```
[1, 0, 0, 0, 0],    # h 0
[0, 1, 0, 0, 0],    # i 1
[0, 0, 1, 0, 0],    # e 2
[0, 0, 0, 1, 0],    # l 3
[0, 0, 0, 0, 1],    # o 4
```



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



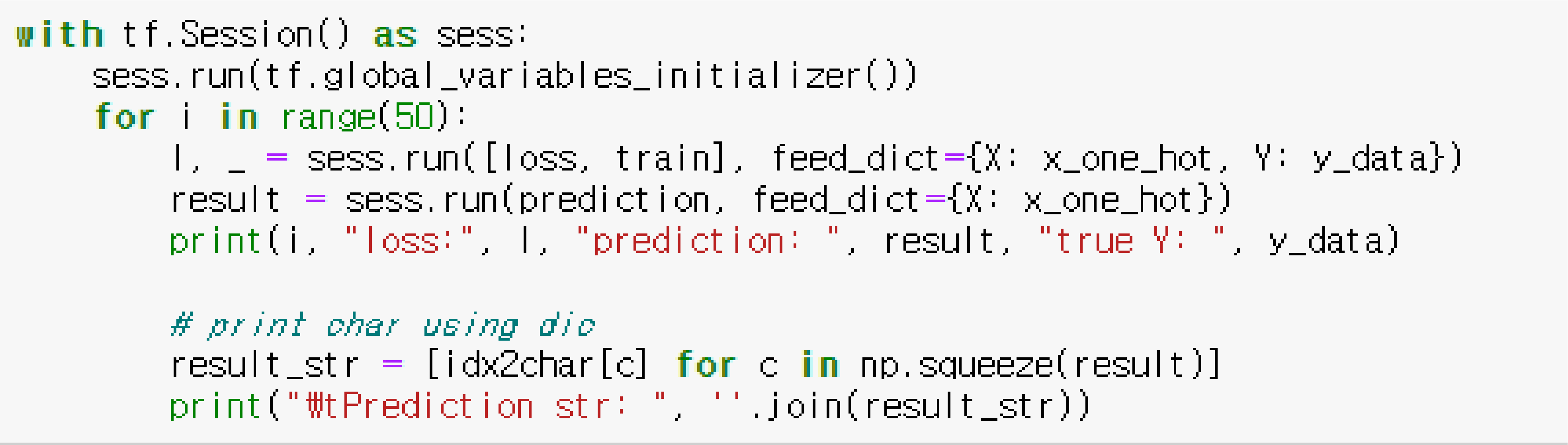
```python
idx2char = ['h', 'i', 'e', 'l', 'o']
# Teach hello: hihell -> ihello
x_data = [[0, 1, 0, 2, 3, 3]]   # hihell
x_one_hot = [[[1, 0, 0, 0, 0],   # h 0
              [0, 1, 0, 0, 0],   # i 1
              [1, 0, 0, 0, 0],   # h 0
              [0, 0, 1, 0, 0],   # e 2
              [0, 0, 0, 1, 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

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



```
# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])
# fc_w = tf.get_variable("fc_w", [hidden_size, num_classes])
# fc_b = tf.get_variable("fc_b", [num_classes])
# outputs = tf.matmul(X_for_fc, fc_w) + fc_b
outputs = tf.contrib.layers.fully_connected(
    inputs=X_for_fc, num_outputs=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)
```

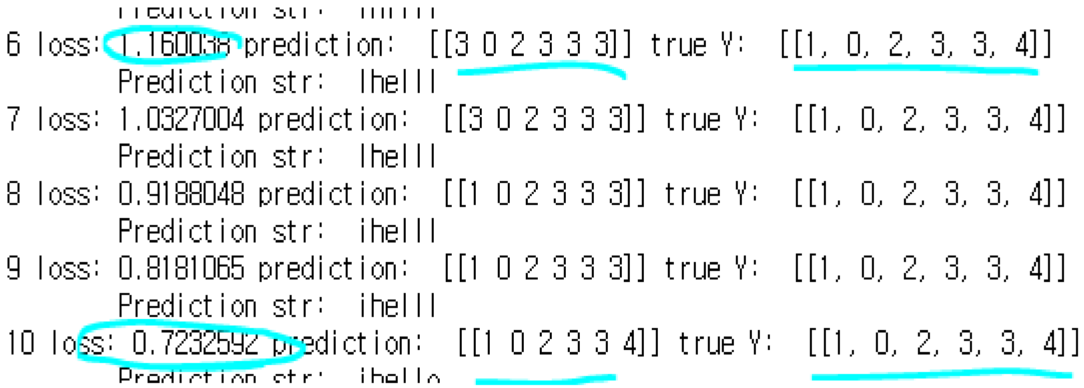
1. RNN에서 나온 output을 바로 학습에 넣는 것은 좋지 않아 재가공.

2. 학습을 위해 loss, optimizer 설정



```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(50):
        l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
        result = sess.run(prediction, feed_dict={X: x_one_hot})
        print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

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

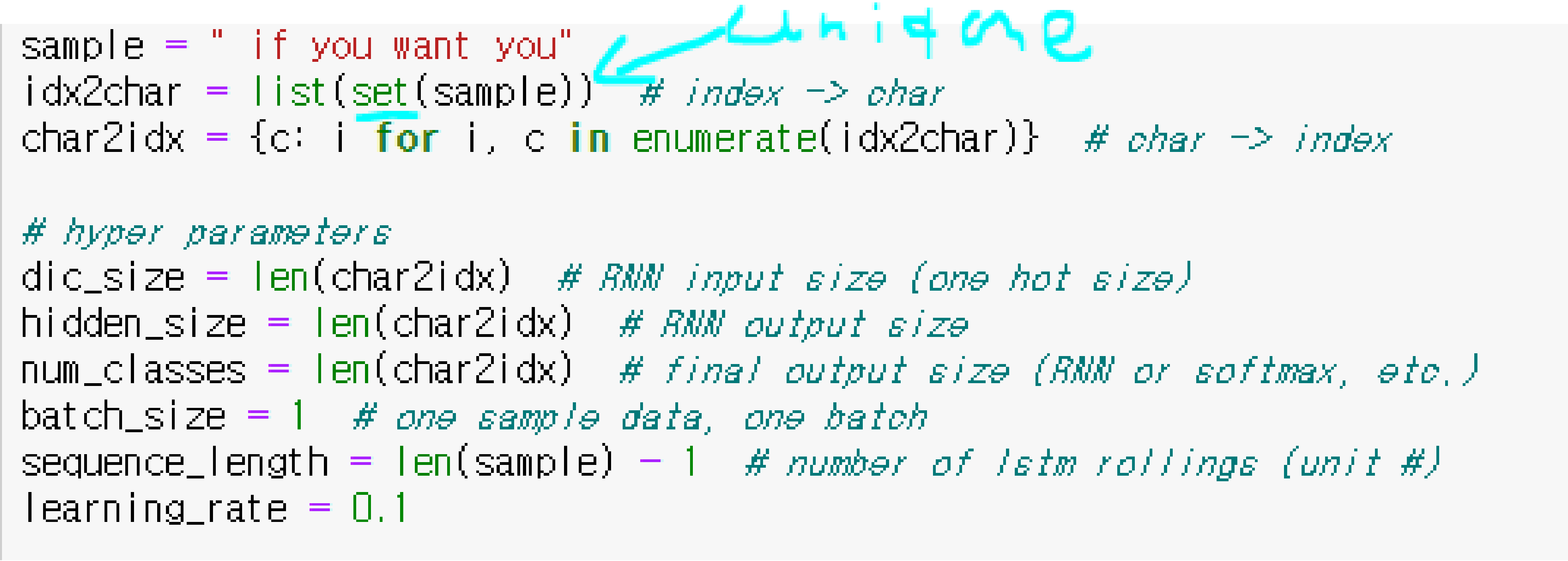
학습 & 예측한 결과값(Prediction) 확인하기



```
         Prediction str:  lhllll
6 loss: 1.160038 prediction:  [[3 0 2 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
         Prediction str:  lhelll
7 loss: 1.0327004 prediction:  [[3 0 2 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
         Prediction str:  lhelll
8 loss: 0.9188048 prediction:  [[1 0 2 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
         Prediction str:  ihelll
9 loss: 0.8181065 prediction:  [[1 0 2 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
         Prediction str:  ihelll
10 loss: 0.7232592 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
         Prediction str:  ihello
```

Loss 값 줄어들고, 예측한 값이
결과 값과 동일하게 된다.

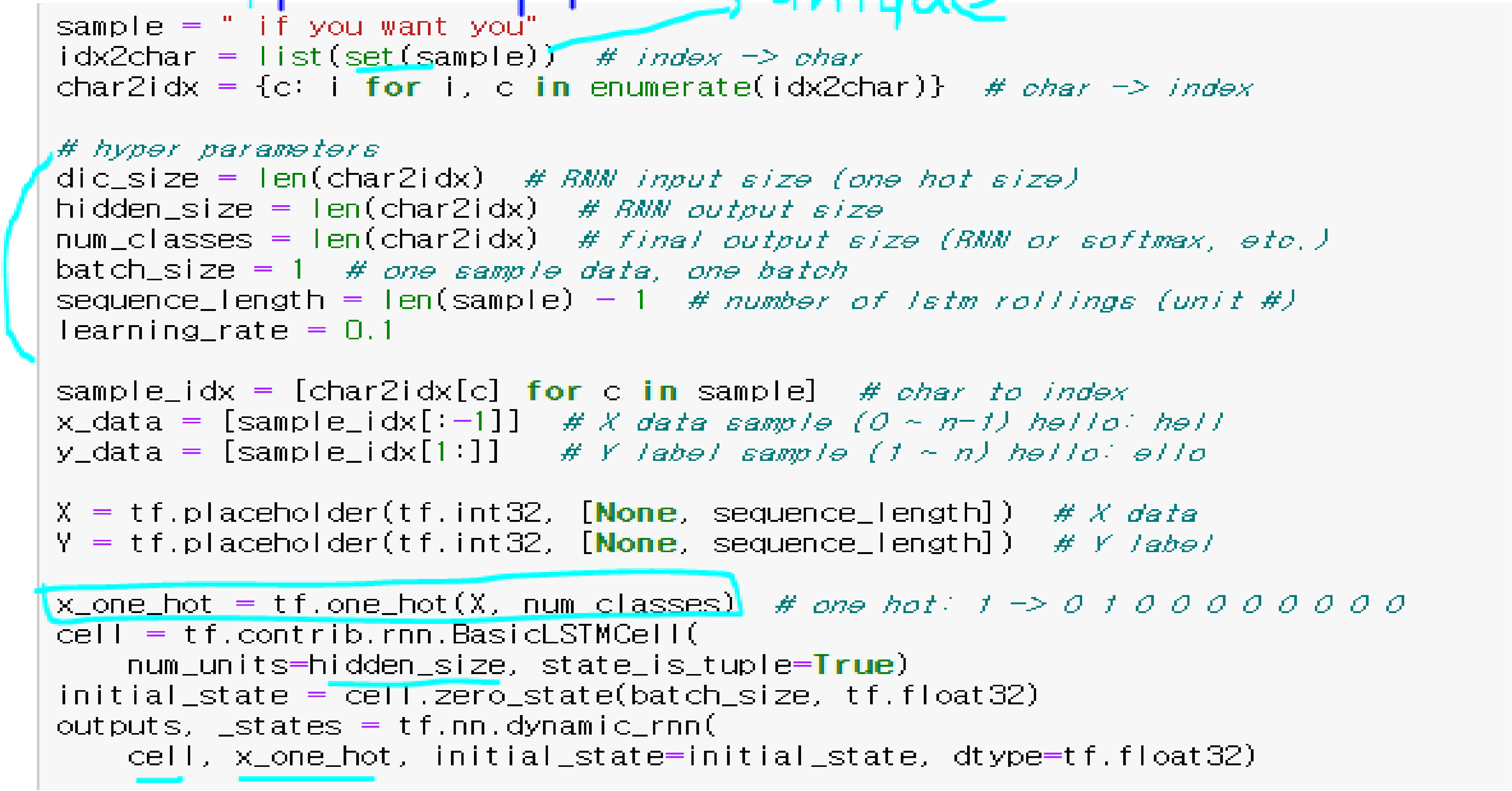
RNN – Long Sequence



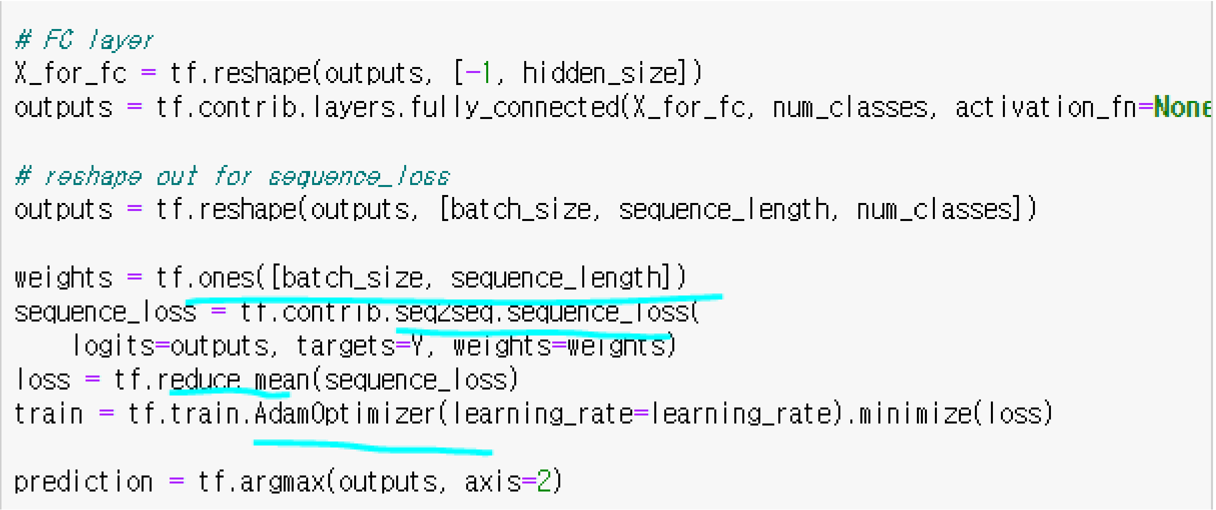
```
sample = " if you want you"
idx2char = list(set(sample))  # index -> char
char2idx = {c: i for i, c in enumerate(idx2char)}  # char -> index

# 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
```

unique

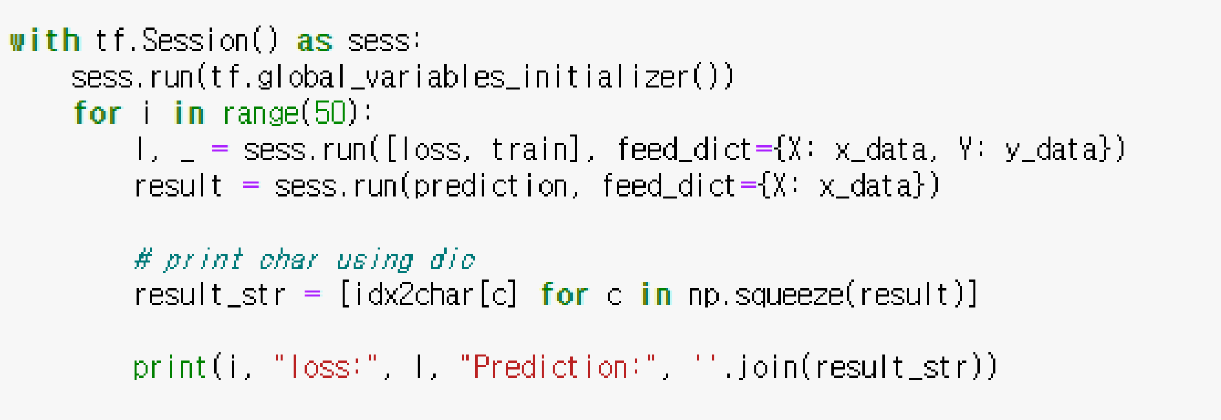
X Y unique



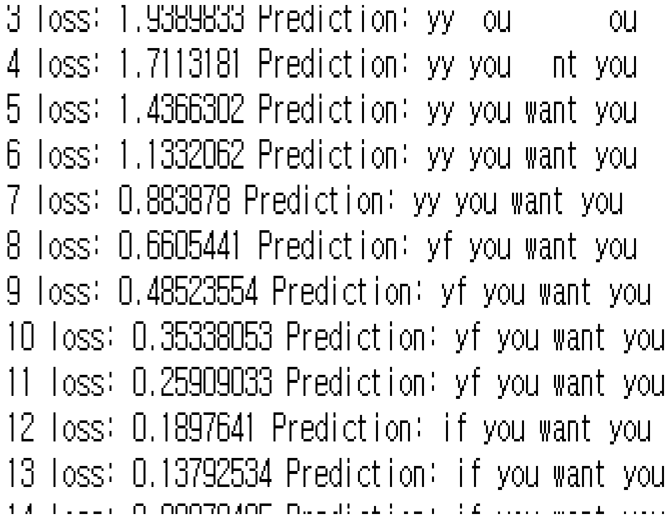
```
sample = " if you want you"
idx2char = list(set(sample))  # index -> char
char2idx = {c: i for i, c in enumerate(idx2char)}  # char -> index

# 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
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)
```



```
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))
```



```
3 loss: 1.9389833 Prediction: yy  ou      ou
4 loss: 1.7113181 Prediction: yy you   nt you
5 loss: 1.4366302 Prediction: yy you want you
6 loss: 1.1332062 Prediction: yy you want you
7 loss: 0.883878 Prediction: yy you want you
8 loss: 0.6605441 Prediction: yf you want you
9 loss: 0.48523554 Prediction: yf you want you
10 loss: 0.35338053 Prediction: yf you want you
11 loss: 0.25909033 Prediction: yf you want you
12 loss: 0.1897641 Prediction: if you want you
13 loss: 0.13792534 Prediction: if you want you
14 loss: 0.09979485 Prediction: if you want you
```