

4개의 글자를 가진 단어를 학습시켜, 3글자만 주어지면 나머지 한글자를 추천하여 단어를 완성한다.

```
In [1]: 1 import tensorflow as tf
        2 import numpy as np
```

C:\Users\WWITHJ\Anaconda3\lib\site-packages\h5py__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
 from ._conv import register_converters as _register_converters

알파벳

```
In [2]: 1 char_arr = ['a', 'b', 'c', 'd', 'e', 'f', 'g',
                'h', 'i', 'j', 'k', 'l', 'm', 'n',
                'o', 'p', 'q', 'r', 's', 't', 'u',
                'v', 'w', 'x', 'y', 'z']
```

```
In [17]: 1 # one-hot 인코딩 사용 및 디코딩을 하기 위해 연관 배열을 만듭니다.
        2 # {'a': 0, 'b': 1, 'c': 2, ..., 'j': 9, 'k': 10, ...}
        3 num_dic = {n: i for i, n in enumerate(char_arr)}
        4 dic_len = len(num_dic)
        5 print(dic_len)
```

26

```
In [5]: 1 # 입력값과 출력값으로 아래와 같이 사용
        2 # wor -> X, d -> Y
        3 # woo -> X, d -> Y
        4 seq_data = ['word', 'wood', 'deep', 'dive', 'cold', 'cool',
                    'load', 'love', 'kiss', 'kind']
        6
```

```
In [ ]: 1
```

```

In [6]: 1 def make_batch(seq_data):
2         input_batch = []
3         target_batch = []
4
5         for seq in seq_data:
6             # 여기서 생성하는 input_batch 와 target_batch 는
7             # 알파벳 배열의 인덱스 번호 입니다.
8             # [22, 14, 17] [22, 14, 14] [3, 4, 4] [3, 8, 21] ...
9             input = [num_dic[n] for n in seq[:-1]]
10            # 3, 3, 15, 4, 3 ...
11
12            target = num_dic[seq[-1]] # 마지막 글자 인덱스
13
14            # one-hot 인코딩을 합니다.
15            # if input is [0, 1, 2]:
16            # [[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
17            #  [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
18            #  [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]]
19
20            input_batch.append(np.eye(dic_len)[input])
21            # 지금까지 손실함수로 사용하던 softmax_cross_entropy_with_logits 함수는
22            # label 값을 one-hot 인코딩으로 넘겨줘야 하지만,
23            # 이 예제에서 사용할 손실 함수인
24            # sparse_softmax_cross_entropy_with_logits 는
25            # one-hot 인코딩을 사용하지 않으므로 index 를 그냥 넘겨주면 됩니다.
26            target_batch.append(target)
27
28        return input_batch, target_batch
29

```

```

In [20]: 1 ## 잠시 결과 확인
2         a, b= make_batch(seq_data)
3         print(b)

```

```
[3, 3, 15, 4, 3, 11, 3, 4, 18, 3]
```

```
In [7]: 1 #####
2 # 옵션 설정
3 #####
4 learning_rate = 0.01
5 n_hidden = 128
6 total_epoch = 30
7 # 타입 스텝: [1 2 3] => 3
8 # RNN 을 구성하는 시퀀스의 갯수입니다.
9 n_step = 3
10 # 입력값 크기. 알파벳에 대한 one-hot 인코딩이므로 26개가 됩니다.
11 # 예) c => [0 0 1 0 0 0 0 0 0 0 ... 0]
12 # 출력값도 입력값과 마찬가지로 26개의 알파벳으로 분류합니다.
13 n_input = n_class = dic_len
```

신경망 모델 구성

```
In [8]: 1 #####
2 # 신경망 모델 구성
3 #####
4 X = tf.placeholder(tf.float32, [None, n_step, n_input])
5 # 비용함수에 sparse_softmax_cross_entropy_with_logits 을 사용하므로
6 # 출력값과의 계산을 위한 원본값의 형태는 one-hot vector가 아니라 인덱스 숫자를 그대로 사용하기 때문에
7 # 다음처럼 하나의 값만 있는 1차원 배열을 입력값으로 받습니다.
8 # [3] [3] [15] [4] ...
9 # 기존처럼 one-hot 인코딩을 사용한다면 입력값의 형태는 [None, n_class] 여야합니다.
10 Y = tf.placeholder(tf.int32, [None])
11
12 W = tf.Variable(tf.random_normal([n_hidden, n_class]))
13 b = tf.Variable(tf.random_normal([n_class]))
14
```

RNN 셀을 생성

```
In [9]: 1 # RNN 셀을 생성합니다.
2 cell1 = tf.nn.rnn_cell.BasicLSTMCell(n_hidden)
3 # 과적합 방지를 위한 Dropout 기법을 사용합니다.
4 cell1 = tf.nn.rnn_cell.DropoutWrapper(cell1, output_keep_prob=0.5)
5 # 여러개의 셀을 조합해서 사용하기 위해 셀을 추가로 생성합니다.
6 cell2 = tf.nn.rnn_cell.BasicLSTMCell(n_hidden)
```

```
In [10]: 1 # 여러개의 셀을 조합한 RNN 셀을 생성합니다.
2 multi_cell = tf.nn.rnn_cell.MultiRNNCell([cell1, cell2])
3
```

```
In [11]: 1 # tf.nn.dynamic_rnn 함수를 이용해 순환 신경망을 만듭니다.
2 # time_major=True
3 outputs, states = tf.nn.dynamic_rnn(multi_cell, X, dtype=tf.float32)
4
```

```
In [12]: 1 # 최종 결과는 one-hot 인코딩 형식으로 만듭니다
2 outputs = tf.transpose(outputs, [1, 0, 2])
3 outputs = outputs[-1]
4 model = tf.matmul(outputs, W) + b
5
6 cost = tf.reduce_mean(
7     tf.nn.sparse_softmax_cross_entropy_with_logits(
8         logits=model, labels=Y))
9
10 optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

```
In [13]: 1 #####
2 # 신경망 모델 학습
3 #####
4 sess = tf.Session()
5 sess.run(tf.global_variables_initializer())
6
7 input_batch, target_batch = make_batch(seq_data)
```

```
In [14]: 1 for epoch in range(total_epoch):
2         _, loss = sess.run([optimizer, cost],
3                             feed_dict={X: input_batch, Y: target_batch})
4
5         print('Epoch:', '%04d' % (epoch + 1),
6               'cost =', '{:.6f}'.format(loss))
7
8         print('최적화 완료!')
```

```
Epoch: 0001 cost = 3.338523
Epoch: 0002 cost = 2.507684
Epoch: 0003 cost = 1.790445
Epoch: 0004 cost = 1.149668
Epoch: 0005 cost = 0.919347
Epoch: 0006 cost = 0.560896
Epoch: 0007 cost = 0.595992
Epoch: 0008 cost = 0.550712
Epoch: 0009 cost = 0.293604
Epoch: 0010 cost = 0.263494
Epoch: 0011 cost = 0.380400
Epoch: 0012 cost = 0.125504
Epoch: 0013 cost = 0.174322
Epoch: 0014 cost = 0.043103
Epoch: 0015 cost = 0.071541
Epoch: 0016 cost = 0.094335
Epoch: 0017 cost = 0.486667
Epoch: 0018 cost = 0.132997
Epoch: 0019 cost = 0.018971
Epoch: 0020 cost = 0.054595
Epoch: 0021 cost = 0.026499
Epoch: 0022 cost = 0.125011
Epoch: 0023 cost = 0.029892
Epoch: 0024 cost = 0.162986
Epoch: 0025 cost = 0.045094
Epoch: 0026 cost = 0.035695
Epoch: 0027 cost = 0.047871
Epoch: 0028 cost = 0.020316
Epoch: 0029 cost = 0.074238
Epoch: 0030 cost = 0.030395
최적화 완료!
```

```

In [15]: 1 #####
          2 # 결과 확인
          3 #####
          4 # 레이블값이 정수이므로 예측값도 정수로 변경해줍니다.
          5 prediction = tf.cast(tf.argmax(model, 1), tf.int32)
          6
          7 # one-hot 인코딩이 아니므로 입력값을 그대로 비교합니다.
          8 prediction_check = tf.equal(prediction, Y)
          9 accuracy = tf.reduce_mean(tf.cast(prediction_check, tf.float32))
          10
          11 input_batch, target_batch = make_batch(seq_data)
          12
          13 predict, accuracy_val = sess.run([prediction, accuracy],
          14                               feed_dict={X: input_batch, Y: target_batch})

```

```

In [16]: 1 predict_words = []
          2 for idx, val in enumerate(seq_data):
          3     last_char = char_arr[predict[idx]]
          4     predict_words.append(val[:3] + last_char)
          5
          6 print('\n=== 예측 결과 ===')
          7 print('입력값:', [w[:3] + ' ' for w in seq_data])
          8 print('예측값:', predict_words)
          9 print('정확도:', accuracy_val)

```

=== 예측 결과 ===

입력값: ['wor ', 'woo ', 'dee ', 'div ', 'col ', 'coo ', 'loa ', 'lov ', 'kis ', 'kin ']

예측값: ['word', 'wood', 'deep', 'dive', 'coll', 'cool', 'load', 'love', 'kiss', 'kind']

정확도: 0.9

In []:

1