RNN을 이용한 단어 자동 완성

- 4개의 글자를 가진 단어를 학습시켜, 3글자만 주어지면 나머지 한글자를 추천하여 단어를 완성한다.
- dynamic_rnn의 sequence_length 옵션을 사용하면 가변 길이의 단어를 학습 시킬 수 있다.
- 학습시킬 데이터는 영문자로 구성된 임의의 단어를 사용함.
- 한 글자 한글자가 한 단계의 입력값, 총 글자 수가 전체 단계가 된다.
- word : 4글자, w, o, r, d - 총 4단계

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

```
C:\Users\WITHJS\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype fr
om `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

- 각각의 알파벳에 대해 index를 구한다.

In [10]:
```
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 [11]:
```
# one-hot 인코딩 사용 및 디코딩을 하기 위해 연관 배열을 만듭니다.
# {'a': 0, 'b': 1, 'c': 2, ..., 'j': 9, 'k', 10, ...}
num_dic = {n: i for i, n in enumerate(char_arr)}
dic_len = len(num_dic)
print(dic_len)
```

```
26
```

In [12]:
```
# 입력값과 출력값으로 아래와 같이 사용
# wor -> X, d -> Y
# woo -> X, d -> Y
seq_data = ['word', 'wood', 'deep', 'dive', 'cold', 'cool',
            'load', 'love', 'kiss', 'kind']
```

In [ ]:
```

```

In [13]:
```
def make_batch(seq_data):
    input_batch = []
    target_batch = []

    for seq in seq_data:
        # 여기서 생성하는 input_batch 와 target_batch 는
        # 알파벳 배열의 인덱스 번호 입니다.
        # [22, 14, 17] [22, 14, 14] [3, 4, 4] [3, 8, 21] ...
        input = [num_dic[n] for n in seq[:-1]]
        # 3, 3, 15, 4, 3 ...

        target = num_dic[seq[-1]]  # 마지막 글자 인덱스

        # one-hot 인코딩을 합니다.
        # if input is [0, 1, 2]:
        # [[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
        #  [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
        #  [ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]]

        input_batch.append(np.eye(dic_len)[input])
        # 지금까지 손실함수로 사용하던 softmax_cross_entropy_with_logits 함수는
        # label 값을 one-hot 인코딩으로 넘겨줘야 하지만,
        # 이 예제에서 사용할 손실 함수인
        # sparse_softmax_cross_entropy_with_logits 는
        # one-hot 인코딩을 사용하지 않으므로 index 를 그냥 넘겨주면 됩니다.
        target_batch.append(target)

    return input_batch, target_batch

```

In [19]:
```
## 잠시 결과 확인
## word
a, b= make_batch(seq_data)
print(a[0])  # w, o, r
print(b[0])  # d
print(b)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
  0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
  0. 0.]]
3
[3, 3, 15, 4, 3, 11, 3, 4, 18, 3]
```

In [20]:
```
#########
# 옵션 설정
######
learning_rate = 0.01
n_hidden = 128
total_epoch = 30
# 타입 스텝: [1 2 3] => 3
# RNN 을 구성하는 시퀀스의 갯수입니다.
n_step = 3
# 입력값 크기. 알파벳에 대한 one-hot 인코딩이므로 26개가 됩니다.
# 예) c => [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... 0]
# 출력값도 입력값과 마찬가지로 26개의 알파벳으로 분류합니다.
n_input = n_class = dic_len
```

신경망 모델 구성

In [24]:
```
#########
# 신경망 모델 구성
######
X = tf.placeholder(tf.float32, [None, n_step, n_input])
# 비용함수에 sparse_softmax_cross_entropy_with_logits 을 사용하므로
# 출력값과의 계산을 위한 원본값의 형태는 one-hot vector가 아니라 인덱스 숫자를 그대로 사용하기 때문에
# 다음처럼 하나의 값만 있는 1차원 배열을 입력값으로 받습니다.
# [3] [3] [15] [4] ...
# 기존처럼 one-hot 인코딩을 사용한다면 입력값의 형태는 [None, n_class] 여야합니다.
Y = tf.placeholder(tf.int32, [None])

W = tf.Variable(tf.random_normal([n_hidden, n_class]))
b = tf.Variable(tf.random_normal([n_class]))

```

RNN 셀을 생성

- DropoutWrapper 함수 : RNN에도 과적합 방지를 위한 DropoutWrapper 함수 적용이 가능하다.

In [25]:
```
# RNN 셀을 생성합니다.
cell1 = tf.nn.rnn_cell.BasicLSTMCell(n_hidden)
# 과적합 방지를 위한 Dropout 기법을 사용합니다.
cell1 = tf.nn.rnn_cell.DropoutWrapper(cell1, output_keep_prob=0.5)
# 여러개의 셀을 조합해서 사용하기 위해 셀을 추가로 생성합니다.
cell2 = tf.nn.rnn_cell.BasicLSTMCell(n_hidden)
```

In [26]:
```
# 여러개의 셀을 조합한 RNN 셀을 생성합니다.
multi_cell = tf.nn.rnn_cell.MultiRNNCell([cell1, cell2])
```

In [27]:
```
# tf.nn.dynamic_rnn 함수를 이용해 순환 신경망을 만듭니다.
# time_major=True
outputs, states = tf.nn.dynamic_rnn(multi_cell, X, dtype=tf.float32)
```

손실함수, 최적화 함수 적용

In [28]:
```
# 최종 결과는 one-hot 인코딩 형식으로 만듭니다
outputs = tf.transpose(outputs, [1, 0, 2])
outputs = outputs[-1]
model = tf.matmul(outputs, W) + b

cost = tf.reduce_mean(
            tf.nn.sparse_softmax_cross_entropy_with_logits(
                logits=model, labels=Y))

optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

신경망 실행 - 그래프 실행

In [30]:
```
#########
# 신경망 모델 학습
######
sess = tf.Session()
sess.run(tf.global_variables_initializer())

input_batch, target_batch = make_batch(seq_data)
```

In [31]:
```
for epoch in range(total_epoch):
    _, loss = sess.run([optimizer, cost],
                       feed_dict={X: input_batch, Y: target_batch})

    print('Epoch:', '%04d' % (epoch + 1),
          'cost =', '{:.6f}'.format(loss))

print('최적화 완료!')
```

```
Epoch: 0001 cost = 3.791007
Epoch: 0002 cost = 2.767154
Epoch: 0003 cost = 1.581693
Epoch: 0004 cost = 1.724090
Epoch: 0005 cost = 1.077464
Epoch: 0006 cost = 0.499617
Epoch: 0007 cost = 0.576309
Epoch: 0008 cost = 0.579886
Epoch: 0009 cost = 0.338431
Epoch: 0010 cost = 0.268343
Epoch: 0011 cost = 0.278959
Epoch: 0012 cost = 0.189432
Epoch: 0013 cost = 0.343576
Epoch: 0014 cost = 0.112635
Epoch: 0015 cost = 0.101098
Epoch: 0016 cost = 0.100363
Epoch: 0017 cost = 0.197772
Epoch: 0018 cost = 0.091607
Epoch: 0019 cost = 0.417259
Epoch: 0020 cost = 0.097675
Epoch: 0021 cost = 0.100898
Epoch: 0022 cost = 0.119844
Epoch: 0023 cost = 0.067566
Epoch: 0024 cost = 0.024673
Epoch: 0025 cost = 0.036463
Epoch: 0026 cost = 0.041144
Epoch: 0027 cost = 0.006614
Epoch: 0028 cost = 0.032478
Epoch: 0029 cost = 0.063141
Epoch: 0030 cost = 0.077687
최적화 완료!
```

결과 확인

In [32]:
```
# 레이블값이 정수이므로 예측값도 정수로 변경해줍니다.
prediction = tf.cast(tf.argmax(model, 1), tf.int32)

# one-hot 인코딩이 아니므로 입력값을 그대로 비교합니다.
prediction_check = tf.equal(prediction, Y)
accuracy = tf.reduce_mean(tf.cast(prediction_check, tf.float32))

input_batch, target_batch = make_batch(seq_data)

predict, accuracy_val = sess.run([prediction, accuracy],
                                 feed_dict={X: input_batch, Y: target_batch})
```

In [33]:
```
predict_words = []
for idx, val in enumerate(seq_data):
    last_char = char_arr[predict[idx]]
    predict_words.append(val[:3] + last_char)

print('Wn=== 예측 결과 ===')
print('입력값:', [w[:3] + ' ' for w in seq_data])
print('예측값:', predict_words)
print('정확도:', accuracy_val)
```

```
=== 예측 결과 ===
입력값: ['wor ', 'woo ', 'dee ', 'div ', 'col ', 'coo ', 'loa ', 'lov ', 'kis ', 'kin ']
예측값: ['word', 'wood', 'deep', 'dive', 'cold', 'cool', 'load', 'love', 'kiss', 'kind']
정확도: 1.0
```

In [ ]:
```

```