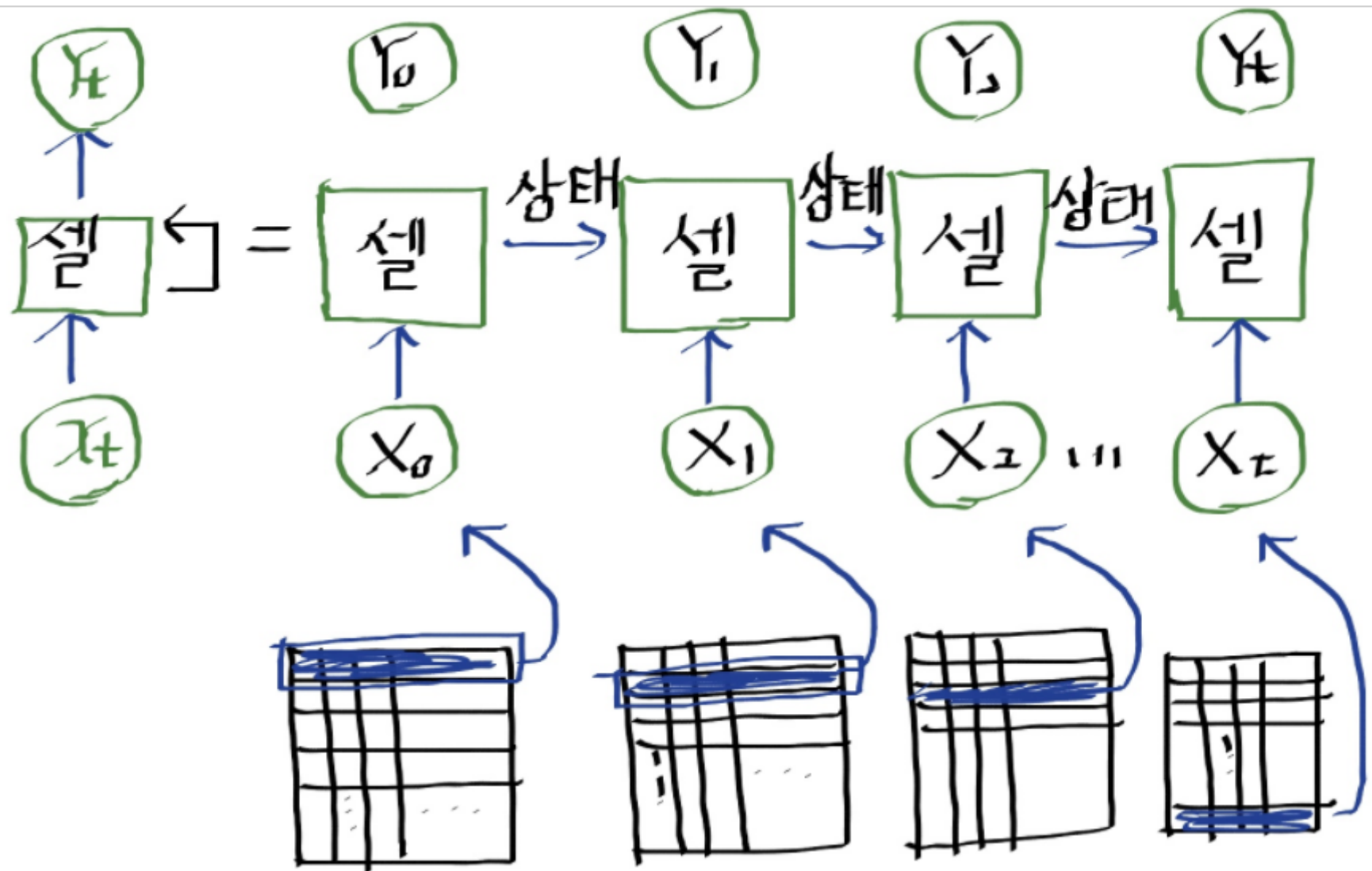


## RNN 이해하기

- RNN이라는 무엇일까?
- RNN의 활용 예
- RNN는 무엇의 약자일까?
- 신경망과 무엇이 다른가?
- MNIST 어떻게 구현할까?

## RNN의 용어 이해

- 순환 신경망(Recurrent Neural Network)이다.
- RNN은 상태가 고정된 데이터를 처리하는 다른 신경망과 달리 자연어 처리나 음성 인식처럼 순서가 있는 데이터를 처리하는 데 강점이 있다.
- 앞이나 뒤의 정보에 따라 전체의 의미가 달라질 때,
- 앞의 정보로 다음에 나오는 정보를 추측하려고 할 때, RNN을 사용하면 좋은 프로그램을 만들 수 있다.
- 2016년 구글의 신경망 기반 기계 번역이 RNN을 이용하여 만든 서비스이다.



```
In [1]: 1 import tensorflow as tf
        2
        3 from tensorflow.examples.tutorials.mnist import input_data
        4 mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
```

C:\Users\WWITHJSW\Anaconda3\lib\site-packages\Wh5py\\_\_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

WARNING:tensorflow:From <ipython-input-1-bed9dddec7b6>:4: read\_data\_sets (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future version.

Instructions for updating:

Please use alternatives such as official/mnist/dataset.py from tensorflow/models.

WARNING:tensorflow:From C:\Users\WWITHJSW\Anaconda3\lib\site-packages\tensorflow\contrib\learn\python\learn\datasets\mnist.py:260: maybe\_download (from tensorflow.contrib.learn.python.learn.datasets.base) is deprecated and will be removed in a future version.

Instructions for updating:

Please write your own downloading logic.

WARNING:tensorflow:From C:\Users\WWITHJSW\Anaconda3\lib\site-packages\tensorflow\contrib\learn\python\learn\datasets\mnist.py:262: extract\_images (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future version.

Instructions for updating:

Please use tf.data to implement this functionality.

Extracting ./mnist/data/train-images-idx3-ubyte.gz

WARNING:tensorflow:From C:\Users\WWITHJSW\Anaconda3\lib\site-packages\tensorflow\contrib\learn\python\learn\datasets\mnist.py:267: extract\_labels (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future version.

Instructions for updating:

Please use tf.data to implement this functionality.

Extracting ./mnist/data/train-labels-idx1-ubyte.gz

WARNING:tensorflow:From C:\Users\WWITHJSW\Anaconda3\lib\site-packages\tensorflow\contrib\learn\python\learn\datasets\mnist.py:110: dense\_to\_one\_hot (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future version.

Instructions for updating:

Please use tf.one\_hot on tensors.

Extracting ./mnist/data/t10k-images-idx3-ubyte.gz

Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz

WARNING:tensorflow:From C:\Users\WWITHJSW\Anaconda3\lib\site-packages\tensorflow\contrib\learn\python\learn\datasets\mnist.py:290: DataSet.\_\_init\_\_ (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future version.

Instructions for updating:

Please use alternatives such as official/mnist/dataset.py from tensorflow/models.

In [2]:

```

1 #####
2 # 옵션 설정
3 #####
4 learning_rate = 0.001
5 total_epoch = 30
6 batch_size = 128
7
8 # RNN 은 순서가 있는 자료를 다루므로,
9 # 한 번에 입력받는 갯수와, 총 몇 단계로 이루어져있는 데이터를 받을지를 설정해야 합니다.
10 # 이를 위해 가로 픽셀수를 n_input 으로, 세로 픽셀수를 입력 단계인 n_step 으로 설정하였습니다.
11 n_input = 28 # 입력
12 n_step = 28 # 28단계 ( 28 X 28)
13 n_hidden = 128 # 은닉층 노드 수 : 128
14 n_class = 10 # 클래스 : 10

```

## 신경망 모델 구성

- RNN은 순서가 있는 데이터를 다룬다. 한번에 입력받을 데이터 개수와 총 몇 단계로 이뤄진 데이터를 받을 지 설정해야 함.
- 가로 픽셀 수 : n\_input, 세로 픽셀 수 : n\_step으로 설정
- 출력값은 원핫 인코딩으로 표현

In [3]:

```

1 X = tf.placeholder(tf.float32, [None, n_step, n_input]) # n_step 차원을 추가
2 Y = tf.placeholder(tf.float32, [None, n_class])
3
4 W = tf.Variable(tf.random_normal([n_hidden, n_class]))
5 b = tf.Variable(tf.random_normal([n_class]))

```

## Tensorflow에서 함수 제공하여 간단하게 RNN 셀 생성

- RNN의 기본 신경망은 다양한 방식의 셀을 사용할 수 있는 함수를 제공
- RNN의 기본 신경망은 긴 단계의 데이터를 학습할 때 맨 뒤에서는 맨 앞의 정보를 잘 기억하지 못함.
- 가장 많이 사용하는 것이 LSTM(Long Short-Term Memory)의 신경망
- GRU(Gated Recurrent Units) 약간 간단한 신경망

```
In [4]: 1 # RNN 에 학습에 사용할 셀을 생성합니다
        2 # 다음 함수들을 사용하면 다른 구조의 셀로 간단하게 변경할 수 있습니다
        3 # BasicRNNCell, BasicLSTMCell, GRUCell
        4 cell = tf.nn.rnn_cell.BasicRNNCell(n_hidden)
```

- `dynamic_rnn` 함수를 이용하여 RNN 신경망 완성
- 한 단계를 학습 후, 상태를 저장, 그 상태를 다음 단계의 입력상태로 하여 다시 학습.
- 이를 반복하여 출력값을 만들어간다. RNN의 기본 구조

```
In [5]: 1 cell
```

```
Out[5]: <tensorflow.python.ops.rnn_cell_impl.BasicRNNCell at 0x20fe1755b38>
```

```
In [6]: 1 # 원래 거쳐야 하는 과정.
        2 # states = tf.zeros(batch_size)
        3 # for i in range(n_step):
        4 #     outputs, states = cell(X[:, i], states)
        5 # ...
        6 # 다음처럼 tf.nn.dynamic_rnn 함수를 사용하면
        7 # CNN 의 tf.nn.conv2d 함수처럼 간단하게 RNN 신경망을 만들어줍니다.
        8 outputs, states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)
```

```
In [7]: 1 print(outputs)
        2 print(states)
```

```
Tensor("rnn/transpose_1:0", shape=(?, 28, 128), dtype=float32)
```

```
Tensor("rnn/while/Exit_3:0", shape=(?, 128), dtype=float32)
```

## 최종 모델 만들기

```
In [6]: 1 # 결과를 Y의 다음 형식과 바꿔야 하기 때문에
        2 # Y : [batch_size, n_class]
        3 # outputs 의 형태를 이에 맞춰 변경해야 합니다.
        4 # outputs : [batch_size, n_step, n_hidden]
        5 # -> [n_step, batch_size, n_hidden]
        6 # tf.transpose 함수를 이용하여 n_step과 batch_size의 차원의 순서를 바꾸고,
        7 # n_step 차원을 제거하여 마지막 단계의 결과값을 취함.
        8 outputs = tf.transpose(outputs, [1, 0, 2])
        9 # -> [batch_size, n_hidden]
        10 outputs = outputs[-1]
        11
        12 # y = x * W + b를 이용하여 최종 결과값.
        13 model = tf.matmul(outputs, W) + b
```

## cost, optimizer 지정

```
In [7]: 1 cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))
        2 optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

## 신경망 모델 학습

```
In [8]: 1 #####
        2 # 신경망 모델 학습
        3 #####
        4 sess = tf.Session()
        5 sess.run(tf.global_variables_initializer())
        6
        7 total_batch = int(mnist.train.num_examples/batch_size)
```

```

In [9]: 1 # total_epoch : 30
        2 # batch_size : 128
        3 # n_step : 28
        4 # n_input : 28
        5 for epoch in range(total_epoch):
        6     total_cost = 0
        7
        8     for i in range(total_batch):
        9         batch_xs, batch_ys = mnist.train.next_batch(batch_size)
       10         # X 데이터를 RNN 입력 데이터에 맞게 [batch_size, n_step, n_input] 형태로 변환합니다.
       11         batch_xs = batch_xs.reshape((batch_size, n_step, n_input))
       12
       13         _, cost_val = sess.run([optimizer, cost],
       14                                feed_dict={X: batch_xs, Y: batch_ys})
       15         total_cost += cost_val
       16
       17     print('Epoch:', '%04d' % (epoch + 1),
       18           'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))
       19
       20 print('최적화 완료!')

```

```

Epoch: 0001 Avg. cost = 0.539
Epoch: 0002 Avg. cost = 0.257
Epoch: 0003 Avg. cost = 0.196
Epoch: 0004 Avg. cost = 0.161
Epoch: 0005 Avg. cost = 0.150
Epoch: 0006 Avg. cost = 0.131
Epoch: 0007 Avg. cost = 0.123
Epoch: 0008 Avg. cost = 0.119
Epoch: 0009 Avg. cost = 0.112
Epoch: 0010 Avg. cost = 0.101
Epoch: 0011 Avg. cost = 0.101
Epoch: 0012 Avg. cost = 0.103
Epoch: 0013 Avg. cost = 0.093
Epoch: 0014 Avg. cost = 0.094
Epoch: 0015 Avg. cost = 0.092
Epoch: 0016 Avg. cost = 0.085
Epoch: 0017 Avg. cost = 0.080
Epoch: 0018 Avg. cost = 0.082
Epoch: 0019 Avg. cost = 0.082
Epoch: 0020 Avg. cost = 0.079

```

Epoch: 0021 Avg. cost = 0.078  
Epoch: 0022 Avg. cost = 0.072  
Epoch: 0023 Avg. cost = 0.072  
Epoch: 0024 Avg. cost = 0.071  
Epoch: 0025 Avg. cost = 0.071  
Epoch: 0026 Avg. cost = 0.063  
Epoch: 0027 Avg. cost = 0.073  
Epoch: 0028 Avg. cost = 0.061  
Epoch: 0029 Avg. cost = 0.071  
Epoch: 0030 Avg. cost = 0.070  
최적화 완료!

## 결과 확인

```
In [10]: 1 #####  
2 # 결과 확인  
3 #####  
4 is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))  
5 accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))  
6 print(is_correct)  
7 print(accuracy)
```

Tensor("Equal:0", shape=(?), dtype=bool)  
Tensor("Mean\_1:0", shape=(), dtype=float32)

```
In [11]: 1 # test_batch_size : 10000, n_step : 28, n_input : 28  
2 test_batch_size = len(mnist.test.images)  
3 test_xs = mnist.test.images.reshape(test_batch_size, n_step, n_input)  
4 test_ys = mnist.test.labels
```



```
In [12]: 1 print(test_batch_size)
          2 print(test_xs)
          3 print(test_ys)
```

```
10000
```

```
[[[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. 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. 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.]
   [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. 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. 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.]
[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. 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. 1. ... 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.]]

```

```

In [13]: 1 print('정확도:', sess.run(accuracy,
      2         feed_dict={X: test_xs, Y: test_ys}))

```

정확도: 0.9727

## REFERENCE : 골빈 해커의 3분 딥러닝