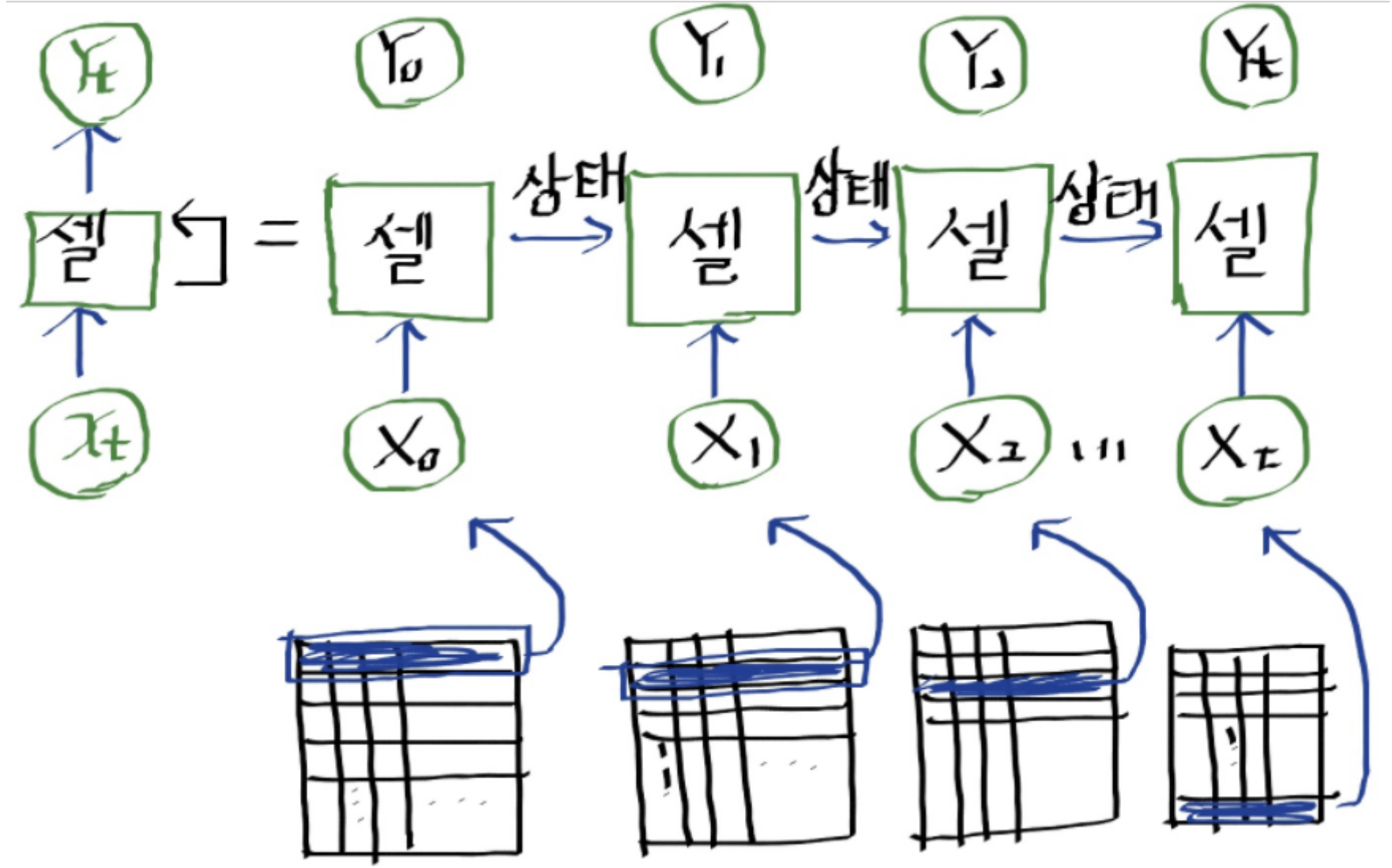
RNN 이해하기

- 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 강점**이 있다.
- 앞이나 뒤의 정보에 따라 전체의 의미가 달라질 때,
- **앞의 정보로 다음에 나오는 정보를 추측**하려고 할 때, RNN을 사용하면 좋은 프로그램을 만들 수 있다.
- 2016년 구글의 신경망 기반 기계 번역이 RNN을 이용하여 만든 서비스이다.



RNN의 구조 이해

- RNN은 셀을 여러개 중첩하여 심층 신경망을 만든다.
- 앞의 학습 결과를 다음 단계의 학습에 이용한다.
- 학습 데이터를 단계별로 구분하여 입력을 한다.
- MNIST를 RNN에 적용한다고 하면, 한 줄단위(28픽셀)을 한 단계의 입력값으로 한다. 총 28단계를 거쳐 입력받음.

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₩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

WARNING:tensorflow:From <ipython-input-1-bed9dddec7b6>:4: read_data_sets (from tensorflow.contrib.learn.python.learn.datasets.mnis
t) 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₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:260: m
aybe_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₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:262: e
xtract_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₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:267: e
xtract_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₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:110: d
ense_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₩WITHJS₩Anaconda3₩lib₩site-packages₩tensorflow₩contrib₩learn₩python₩learn₩datasets₩mnist.py:290: D
ataSet.__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]:
```
########
# 옵션 설정
######
learning_rate = 0.001
total_epoch = 30
batch_size = 128

# RNN 은 순서가 있는 자료를 다루므로,
# 한 번에 입력받는 갯수와, 총 몇 단계로 이루어져있는 데이터를 받을지를 설정해야합니다.
# 이를 위해 가로 픽셀수를 n_input 으로, 세로 픽셀수를 입력 단계인 n_step 으로 설정하였습니다.
n_input = 28   # 입력
n_step = 28    # 28단계 ( 28 X 28)
n_hidden = 128 # 은닉층 노드 수 : 128
n_class = 10   # 클래스 : 10
```

신경망 모델 구성

- RNN은 순서가 있는 데이터를 다룬다.
- 한번에 입력받을 데이터 개수와 총 몇 단계로 이뤄진 데이터를 받을 지 설정(n_step)해야 함.
- 이미지의 경우, 가로 픽셀 수 : n_input, 세로 픽셀 수 : n_step으로 설정
- 출력값은 원핫 인코딩으로 표현

In [3]:
```
X = tf.placeholder(tf.float32, [None, n_step, n_input])  # n_step 차원을 추가
Y = tf.placeholder(tf.float32, [None, n_class])
```

Tensorflow에서 함수 제공하여 간단하게 RNN 셀 생성

- RNN의 기본 신경망은 다양한 방식의 셀을 사용할 수 있는 함수를 제공
- RNN의 기본 신경망은 단점이 있음.
  - 긴 단계의 데이터를 학습할 때 맨 뒤에서는 맨 앞의 정보를 잘 기억하지 못함.
  - 단점을 어느정도 보완하여 많이 사용하는 것이 LSTM(Long Short-Term Memory)의 신경망
- GRU(Gated Recurrent Units) 신경망 : LSTM과 비슷하지만 구조가 더 간단한 신경망

신경망 구성을 위한 셀을 구성

In [4]:
```
# BasicRNNCell,BasicLSTMCell,GRUCell
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]:
```
cell
```

Out[5]: <tensorflow.python.ops.rnn_cell_impl.BasicRNNCell at 0x120aaaa5cc0>

반복과정

```
states = tf.zeros(batch_size)
for i in range(n_step):
    outputs, states = cell(X[[:, i]], states)
...
다음처럼 tf.nn.dynamic_rnn 함수를 사용하면
CNN 의 tf.nn.conv2d 함수처럼 간단하게 RNN 신경망을 만들어줍니다.
```

- 한 단계를 학습한 뒤, 상태를 저장한다.
- 그 상태를 다음 단계의 입력 상태로 하여 다시 학습한다.
- 주어진 단계만큼 반복하면서 상태를 전파하며 출력값을 만들어간다. RNN의 기본 구조
- 이에 대한 전 단계를 고려하며 RNN의 모델의 핵심 구조를 만들 수 있는 것이 dynamic_rnn함수이다.

In [6]:
```
outputs, states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)
```

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

```
Tensor("rnn/transpose_1:0", shape=(?, 28, 128), dtype=float32)
Tensor("rnn/while/Exit_3:0", shape=(?, 128), dtype=float32)
```

In [8]:
```
W = tf.Variable(tf.random_normal([n_hidden, n_class]))
b = tf.Variable(tf.random_normal([n_class]))
```

최종 모델 만들기

- RNN 신경망이 출력값의 형태 -> [batch_size, n_step, n_hidden]
  - batch_size : 데이터의 개수
  - n_step : RNN의 단계
  - n_hidden : 은닉층의 노드 수
- 우리는 위의 형태를 가중치(W)의 형태로 변경해 주어야 한다.
  - (가) 순서 바꾸기 : [batch_size, n_step, n_hidden] -> [n_step, batch_size, n_hidden]
  - (나) n_step의 차원을 제거 [batch_size, n_hidden]

In [9]:
```
# 결과를 Y의 다음 형식과 바꿔야 하기 때문에
# Y : [batch_size, n_class]
# outputs 의 형태를 이에 맞춰 변경해야합니다.
# outputs : [batch_size, n_step, n_hidden]
# -> [n_step, batch_size, n_hidden]
# tf.transpose 함수를 이용하여 n_step과 batch_size의 차원의 순서를 바꾸고,
# n_step 차원을 제거하여 마지막 단계의 결과값을 취함.
print(outputs)
outputs1 = tf.transpose(outputs, [1, 0, 2])
print(outputs1)
#  -> [batch_size, n_hidden]
outputs2 = outputs1[-1]   # 맨 마지막의 결과값만 취한다.
print(outputs2)
# y = x * W + b를 이용하여 최종 결과값.
model = tf.matmul(outputs2, W) + b
```

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

cost(비용), optimizer(최적화 알고리즘) 지정

- 손실(Loss) 함수 : cross_entropy

- 최적화 알고리즘 : AdamOptimizer

In [10]:
```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

신경망 모델 학습

In [11]:
```
#########
# 신경망 모델 학습
######
sess = tf.Session()
sess.run(tf.global_variables_initializer())

total_batch = int(mnist.train.num_examples/batch_size)
```

In [12]:
```
# total_epoch : 30
# batch_size : 128
# n_step : 28
# n_input : 28
for epoch in range(total_epoch):
    total_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # X 데이터를 RNN 입력 데이터에 맞게 [batch_size, n_step, n_input] 형태로 변환합니다.
        batch_xs = batch_xs.reshape((batch_size, n_step, n_input))

        _, cost_val = sess.run([optimizer, cost],
                               feed_dict={X: batch_xs, Y: batch_ys})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))

print('최적화 완료!')
```

```
Epoch: 0001 Avg. cost = 0.568
Epoch: 0002 Avg. cost = 0.243
Epoch: 0003 Avg. cost = 0.188
Epoch: 0004 Avg. cost = 0.160
Epoch: 0005 Avg. cost = 0.139
Epoch: 0006 Avg. cost = 0.136
Epoch: 0007 Avg. cost = 0.123
Epoch: 0008 Avg. cost = 0.114
Epoch: 0009 Avg. cost = 0.111
Epoch: 0010 Avg. cost = 0.099
Epoch: 0011 Avg. cost = 0.103
Epoch: 0012 Avg. cost = 0.093
Epoch: 0013 Avg. cost = 0.090
Epoch: 0014 Avg. cost = 0.087
Epoch: 0015 Avg. cost = 0.087
Epoch: 0016 Avg. cost = 0.083
Epoch: 0017 Avg. cost = 0.083
Epoch: 0018 Avg. cost = 0.078
Epoch: 0019 Avg. cost = 0.082
Epoch: 0020 Avg. cost = 0.077
```

```
Epoch: 0021 Avg. cost = 0.073
Epoch: 0022 Avg. cost = 0.079
Epoch: 0023 Avg. cost = 0.069
Epoch: 0024 Avg. cost = 0.074
Epoch: 0025 Avg. cost = 0.067
Epoch: 0026 Avg. cost = 0.063
Epoch: 0027 Avg. cost = 0.064
Epoch: 0028 Avg. cost = 0.062
Epoch: 0029 Avg. cost = 0.065
Epoch: 0030 Avg. cost = 0.060
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]:
```
print(test_batch_size)
print(test_xs)
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]:

```
print('정확도:', sess.run(accuracy,
                       feed_dict={X: test_xs, Y: test_ys}))
```

```
정확도: 0.9727
```

REFERENCE : 골빈 해커의 3분 딥러닝