LAB05 MNIST 데이터를 이용한 신경망 모델

01. MNIST 데이터 셋 설명

- 0 ~ 9까지의 숫자를 28 X 28 픽셀 크기의 이미지로 구성.
- 딥러닝 입문의 첫 시작 : Hello World!
- 기본 내장된 mnist 모듈을 이용하여 데이터를 로드

In [4]:

```
import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data
```

In [5]:

```
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
```

```
Extracting ./mnist/data/train-images-idx3-ubyte.gz
Extracting ./mnist/data/train-labels-idx1-ubyte.gz
Extracting ./mnist/data/t10k-images-idx3-ubyte.gz
Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz
```

02. 신경망 모델 구성

- 28 X 28 픽셀(X) -> 각각의 픽셀은 하나의 특징, 즉 784개의 특징
- Label(Y)은 0~9까지 숫자를 가르킴. 10개의 분류
- 입력 X, 출력 Y
- Palceholder를 이용하여 None에는 현재 데이터 개수를 모르기 때문에 None으로 둔다.

In [6]:

```
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])
```

우리가 만들 신경망 (은닉층이 2개)

- 784개의 특징(입력)
- 256 (첫번째 은닉층의 뉴런 개수)
- 256 (두번째 은닉층의 뉴런 개수)
- 10 (결과값 0-9 분류 개수)
- 각 뉴런 노드를 연결하는 가중치는 임의의 값으로 지정한다.
- 활성화 함수는 ReLU함수를 이용한다.

In [7]:

```
# 784개 입력, 256개의 뉴런
# 표준편차가 0.01인 정규 분포를 가지는 임의의 뉴런을 초기화 시킨다.
W1 = tf.Variable(tf.random_normal([784, 256], stddev=0.01))

# X(입력값)에 가중치를 곱하고, 이후 ReLU 함수를 이용하여 레이어를 만든다.
L1 = tf.nn.relu(tf.matmul(X, W1))  # 데이터수 X 784 * 784 X 256 => 데이터수 X 256

W2 = tf.Variable(tf.random_normal([256,256], stddev=0.01))
# L1(입력값)에 가중치를 곱하고, 이후 ReLU 함수를 이용하여 레이어를 만든다.
L2 = tf.nn.relu(tf.matmul(L1, W2)) # 데이터수 X 256 * 256 X 256 => 데이터수 X 256

W3 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
model = tf.matmul(L2, W3) # 데이터수 X 256 * 256 X 10 => 데이터수 X 10
```

03. 손실함수(Loss) 및 최적화 알고리즘

- 손실(Loss) 함수 : Cross_entropy(크로스 엔트로피)
- tf.train.AdamOptimizer 함수 이용 최적화를 수행

In [8]:

```
cost = tf.reduce_mean(
          tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

04. 그래프 실행

In [9]:

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

05. 미니배치를 이용한 학습

- 가중치를 업데이트를 위한 학습시의 데이터의 개수를 일정개수 지정(미니 배치)
- 일반적으로 데이터를 적당한 크기로 잘라서 학습시킴.
- 100개씩의 데이터 단위로 가중치 갱신 수행.
- 1 epoch : 전체 데이터가 한번 학습했음을 의미함.

알아두어야 할 내용 - 데이터 셋을 나누는 이유(학습/테스트)

별도의 테스트 데이터를 사용하는 이유는 학습 데이터로 예측을 하면 예측 정확도가 매우 높게 나오지만, 학습 데이터에 포함되지 않은 새로운 데이터를 예측할 때는 정확도가 매우 떨어지는 경우가 많기 때문.

- 이런 현상을 우리는 과적합이라 한다.

In [10]:

```
batch_size = 100
total_batch = int(mnist.train.num_examples / batch_size)
```

In [12]:

```
for epoch in range(15):
  total_cost = 0
  for i in range(total_batch):
    # 학습할 데이터를 가져온다.
    batch_xs, batch_ys = mnist.train.next_batch(batch_size)

    # 입력 X, 출력 Y에 각각의 데이터 넣고 실행
    _, cost_val = sess.run([optimizer,cost],
                           feed_dict={X:batch_xs, Y:batch_ys})
    total_cost += cost_val
  print(batch_xs.shape, batch_ys.shape)
  print('Epoch {}, Avg. cost = {}'.format(epoch+1, total_cost/total_batch))
```

```
(100, 784) (100, 10)
Epoch 1, Avg. cost = 0.01051193731982163
(100, 784) (100, 10)
Epoch 2, Avg. cost = 0.007532538338802458
(100, 784) (100, 10)
Epoch 3, Avg. cost = 0.011062648312059041
(100, 784) (100, 10)
Epoch 4, Avg. cost = 0.01042535298374291
(100, 784) (100, 10)
Epoch 5, Avg. cost = 0.0059479653918647684
(100, 784) (100, 10)
Epoch 6, Avg. cost = 0.009620980600987174
(100, 784) (100, 10)
Epoch 7, Avg. cost = 0.007457145980214673
(100, 784) (100, 10)
Epoch 8, Avg. cost = 0.005536378025474484
(100, 784) (100, 10)
Epoch 9, Avg. cost = 0.008377153798960948
(100, 784) (100, 10)
Epoch 10, Avg. cost = 0.007472366088188871
(100, 784) (100, 10)
Epoch 11, Avg. cost = 0.0045059184550617914
(100, 784) (100, 10)
Epoch 12, Avg. cost = 0.007065523110156664
(100, 784) (100, 10)
Epoch 13, Avg. cost = 0.0045431002585160675
(100, 784) (100, 10)
Epoch 14, Avg. cost = 0.006000384793542667
(100, 784) (100, 10)
Epoch 15, Avg. cost = 0.006420198379775932
```

06. 학습 후, 결과 확인

In [11]:

```
# tf.argmax(model, 1)는 최대값을 뽑기
# tf.argmax(Y,1)는 1번 인덱스(두번째)값 중에서 최대값 뽑기
# 결과는 10개 레이블중에 확률이 가장 높은 값이 된다.
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y,1))
is_correct
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
accuracy
```

Out[11]:

```
<tf.Tensor 'Mean_1:0' shape=() dtype=float32>
```

In [12]:

```
print(mnist.test.images.shape)
print(mnist.test.labels.shape)
```

```
(10000, 784)
(10000, 10)
```

mnist.test 테스트 데이터 셋을 이용하여 예측 수행

In [13]:

```
print('정확도', sess.run(accuracy, feed_dict={X:mnist.test.images,
                                              Y:mnist.test.labels}))
```

```
정확도 0.9799
```

In [0]:

```

```