심층 신경망 구성

학습 내용

- MNIST 데이터 셋을 이용하여 심층 신경망을 구현한다.
- 은닉층이 2개인 신경망을 구현한다.

In [13]:

```
import tensorflow as tf
```

In [14]:

```
import os, warnings
# 경고 메시지 무시하거나 숨길때(ignore), 다시보이게(default)
# warnings.filterwarnings(action='default')
warnings.filterwarnings(action='ignore')
```

In [15]:

```
from IPython.display import display, Image
```

In [16]:

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
```

```
WARNING:tensorflow:From <ipython-input-16-4dcbd946c02b>:2: read_data_sets (from tens
orflow.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₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:260: maybe_download (from tensorflow.cont
rib.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₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩base.py:252: _internal_retry.<locals>.wrap.<locals
>.wrapped_fn (from tensorflow.contrib.learn.python.learn.datasets.base) is deprecate
d and will be removed in a future version.
Instructions for updating:
Please use urllib or similar directly.
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
WARNING:tensorflow:From C:₩Users₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:262: extract_images (from tensorflow.cont
rib.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
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
WARNING:tensorflow:From C:₩Users₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:267: extract_labels (from tensorflow.cont
rib.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₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:110: dense_to_one_hot (from tensorflow.co
ntrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Please use tf.one_hot on tensors.
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting ./mnist/data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:₩Users₩peop₩Anaconda3₩lib₩site-packages₩tensorflow_core₩co
ntrib₩learn₩python₩learn₩datasets₩mnist.py:290: DataSet.__init__ (from tensorflow.co
ntrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
```

In [17]:

```
# 데이터의 개수
print(mnist.test.num_examples, mnist.train.num_examples, mnist.validation.num_examples)

# 데이터의 행열 사이즈
print(mnist.train.labels.shape, mnist.train.images.shape)
print(mnist.test.labels.shape, mnist.test.images.shape)
print(mnist.validation.labels.shape, mnist.validation.images.shape)
```

```
10000 55000 5000
(55000, 10) (55000, 784)
(10000, 10) (10000, 784)
(5000, 10) (5000, 784)
```

01 신경망 모델 구성하기

- MNIST의 손글씨는 28 X 28로 구성되어 있다.
- 784개의 특징(픽셀)로 구성되어 있음.
- 레이블은 0~9까지의 10개의 분류

In [19]:

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

- tf.Variable(tf.random_normal([784, 256], stddev=0.01))
- 표준편차가 0.01인 정규 분포를 갖는 값.
- tf.matmul() 각 계층으로 들어오는 입력값에 각각의 가중치를 곱한다.
- tf.nn.relu() 활성화 함수로 ReLU를 사용하는 신경망 계층을 만든다.

In [20]:

```
W1 = tf.Variable(tf.random_normal([784, 256], stddev=0.01))
L1 = tf.nn.relu(tf.matmul(X, W1))

W2 = tf.Variable(tf.random_normal([256, 256], stddev=0.01))
L2 = tf.nn.relu(tf.matmul(L1, W2))

W3 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
model = tf.matmul(L2, W3)

print(W3)
print(model)
```

```
<tf.Variable 'Variable_2:0' shape=(256, 10) dtype=float32_ref>
Tensor("MatMul_2:0", shape=(?, 10), dtype=float32)
```

비용함수, 최적화 함수 지정

- AdamOptimizer (Adaptive Moment Estimation)은 RMSProp와 Momentum방식을 합친 것.

In [22]:

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

세션 생성 및 초기화

In [23]:

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

테스트 데이터와 학습 데이터를 분류하는 이유

- 과적합 해소.

배치 사이즈 지정

- Mini-batch 크기가 전체 트레이닝 셋 데이터 사이즈인 m과 같다면 이것은 Batch gradient descent방법
  - 데이터가 별로 없다면 batch gradient descent를 사용
- Mini-batch 크기가 1이라면 Stochastic gradient descent라고 한다.
  - **적은 메모리**로 동작가능
  - 64,128, 256, 512 사이즈 선택

In [24]:

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

In [25]:

```
# MNIST 데이터 전체를 학습하는 일을 15번 반복함.
# 학습 데이터 전체를 한 바퀴를 도는 일을 에포크(epoch)라 한다.

for epoch in range(15):
    total_cost = 0

    for i in range(total_batch):
        # 배치 사이즈만큼 데이터 가져오기
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)

        # 입력값 : batch_xs, 출력값 : batch_ys
        # 최적화를 수행 후, 손실을 구한다.
        _, cost_val = sess.run([optimizer, cost],
                               feed_dict={X:batch_xs, Y:batch_ys})

        # 총 손실 계산
        total_cost = total_cost + cost_val

    print("Epoch : %4d" %(epoch + 1),
                '평균 Cost = ', "{:.3f}".format(total_cost/total_batch))

print("최적화 완료!")
```

```
Epoch :    1 평균 Cost =  0.422
Epoch :    2 평균 Cost =  0.153
Epoch :    3 평균 Cost =  0.099
Epoch :    4 평균 Cost =  0.071
Epoch :    5 평균 Cost =  0.054
Epoch :    6 평균 Cost =  0.041
Epoch :    7 평균 Cost =  0.032
Epoch :    8 평균 Cost =  0.025
Epoch :    9 평균 Cost =  0.021
Epoch :   10 평균 Cost =  0.020
Epoch :   11 평균 Cost =  0.015
Epoch :   12 평균 Cost =  0.013
Epoch :   13 평균 Cost =  0.014
Epoch :   14 평균 Cost =  0.012
Epoch :   15 평균 Cost =  0.012
최적화 완료!
```

실습해 보기

- 배치 사이즈를 조정해 가면서 확인해 본다.
- 은닉층 노드 및 은닉층수를 늘려가면 확인해 본다.
- 각각의 Cost 값을 확인해 본다.