MNIST 분류 모델 만들기 - 신경망

학습 내용

- 데이터 전처리 후, 딥러닝 모델 돌려보기

In [1]:

```
from keras.datasets import mnist
from keras.utils import np_utils
```

In [2]:

```
import numpy
import sys
import tensorflow as tf
```

In [3]:

```
seed = 0
numpy.random.seed(seed)
```

데이터 다운로드

In [11]:

```
# 처음 다운일 경우, 데이터 다운로드 시간이 걸릴 수 있음.
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

In [12]:

```
import matplotlib.pyplot as plt
```

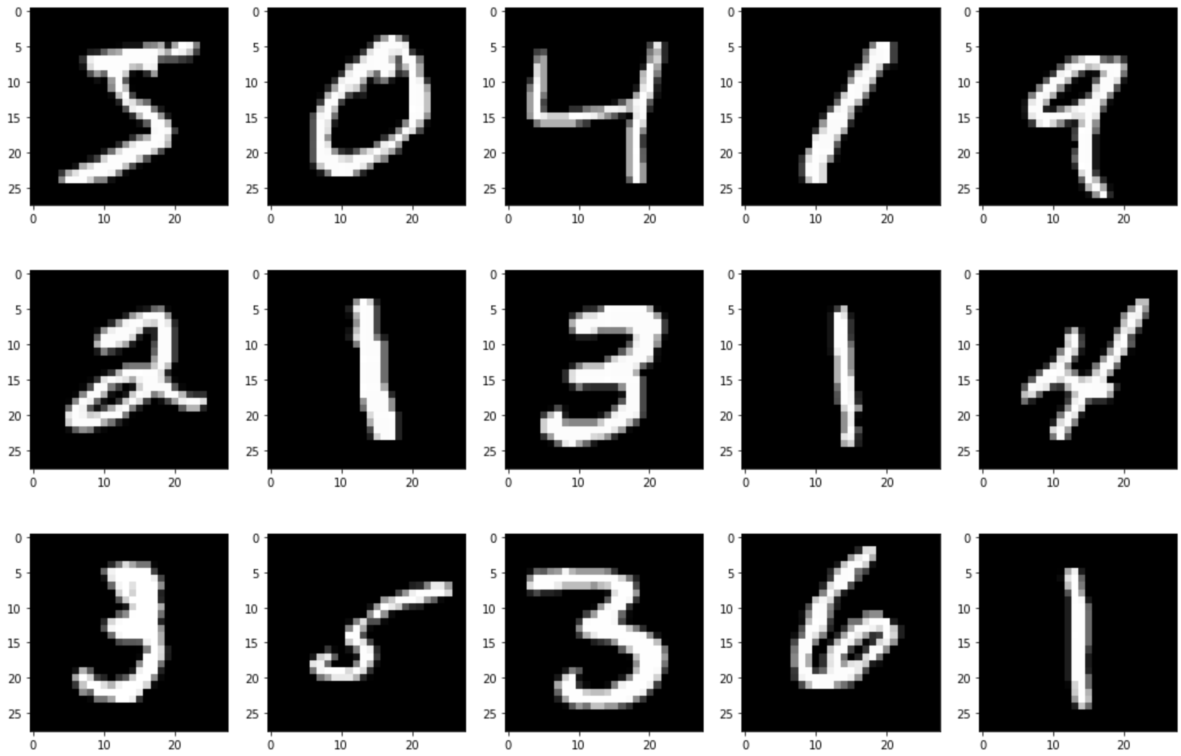
In [13]:

```
fig, axes = plt.subplots(3, 5, figsize=(18,12) )

print("label={}".format(y_train[0:15]))   # x데이터 0~14개 가져오기

for image, ax in zip( X_train, axes.ravel() ):
    ax.imshow(image) # 이미지 표시
```

```
label=[5 0 4 1 9 2 1 3 1 4 3 5 3 6 1]
```



X_train의 데이터 정보를 하나 보기

In [14]:

```
print(X_train.shape)  # 60000 만개, 28행, 28열
X_train[0].shape
```

(60000, 28, 28)

Out[14]:

(28, 28)

신경망에 맞추어 주기 위해 데이터 전처리

- 학습 데이터
- 테스트 데이터

In [15]:

```
# X_train = X_train.reshape(X_train.shape[0],784)   # 60000, 28, 28 -> 60000, 784로 변경
# 데이터 값의 범위 0~255 -> 0~1
# X_train.astype('float64')
# X_train = X_train/255

# 이렇게도 가능
X_train = X_train.reshape(X_train.shape[0],784).astype('float64') / 255
```

In [16]:

```
import numpy as np
```

In [17]:

```
print(X_train.shape)                # 데이터 크기
np.min(X_train), np.max(X_train)   # 값의 범위
```

(60000, 784)

Out[17]:

(0.0, 1.0)

In [18]:

```
# 테스트 데이터 전처리
X_test = X_test.reshape(X_test.shape[0],784)
X_test.astype('float64')
X_test = X_test/255
```

In [19]:

```
print(X_test.shape)              # 데이터 크기
np.min(X_test), np.max(X_test)   # 값의 범위
```

(10000, 784)

Out[19]:

(0.0, 1.0)

출력데이터 검증을 위해 10진수의 값을 One-Hot Encoding을 수행

In [20]:

```
# OneHotEncoding - 10진수의 값을 0, 1의 값을 갖는 벡터로 표현
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
```

변환 전과 후

In [21]:

```
y_train[0:4]
```

Out[21]:

array([5, 0, 4, 1], dtype=uint8)

In [22]:

```
Y_train[0:4]
```

Out[22]:

```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

딥러닝 만들어 보기

In [23]:

```
from keras.models import Sequential
from keras.layers import Dense
```

In [26]:

```
m = Sequential()
m.add(Dense(512,input_dim=784, activation='relu'))
m.add(Dense(128, activation='relu') )
m.add(Dense(10,activation='softmax'))  #softmax
```

오차함수 :categorical_crossentropy, 최적화 함수 : adam

In [27]:

```
m.compile(loss="categorical_crossentropy",
         optimizer='adam',
         metrics=['accuracy'])
```

In [28]:

```
### 배치 사이즈 200, epochs 30회 실행,
history = m.fit(X_train, Y_train, validation_data=(X_test, Y_test),
                epochs=30,
                batch_size=200,
                verbose=1)
```

```
Epoch 1/30
300/300 [==============================] - 5s 15ms/step - loss: 0.2680 - accuracy:
0.9225 - val_loss: 0.1230 - val_accuracy: 0.9625
Epoch 2/30
300/300 [==============================] - 3s 11ms/step - loss: 0.1002 - accuracy:
0.9698 - val_loss: 0.0836 - val_accuracy: 0.9733
Epoch 3/30
300/300 [==============================] - 3s 11ms/step - loss: 0.0621 - accuracy:
0.9811 - val_loss: 0.0811 - val_accuracy: 0.9741
Epoch 4/30
300/300 [==============================] - 4s 14ms/step - loss: 0.0421 - accuracy:
0.9873 - val_loss: 0.0815 - val_accuracy: 0.9731
Epoch 5/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0303 - accuracy:
0.9908 - val_loss: 0.0661 - val_accuracy: 0.9799
Epoch 6/30
300/300 [==============================] - 3s 12ms/step - loss: 0.0237 - accuracy:
0.9927 - val_loss: 0.0710 - val_accuracy: 0.9787
Epoch 7/30
300/300 [==============================] - 4s 13ms/step - loss: 0.0157 - accuracy:
0.9951 - val_loss: 0.0732 - val_accuracy: 0.9792
Epoch 8/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0134 - accuracy:
0.9959 - val_loss: 0.0717 - val_accuracy: 0.9795
Epoch 9/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0107 - accuracy:
0.9970 - val_loss: 0.0757 - val_accuracy: 0.9787
Epoch 10/30
300/300 [==============================] - 4s 14ms/step - loss: 0.0112 - accuracy:
0.9965 - val_loss: 0.0719 - val_accuracy: 0.9800
Epoch 11/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0104 - accuracy:
0.9969 - val_loss: 0.0728 - val_accuracy: 0.9794
Epoch 12/30
300/300 [==============================] - 4s 13ms/step - loss: 0.0094 - accuracy:
0.9969 - val_loss: 0.0995 - val_accuracy: 0.9766
Epoch 13/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0098 - accuracy:
0.9966 - val_loss: 0.0786 - val_accuracy: 0.9797
Epoch 14/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0052 - accuracy:
0.9984 - val_loss: 0.0882 - val_accuracy: 0.9796
Epoch 15/30
300/300 [==============================] - 4s 13ms/step - loss: 0.0088 - accuracy:
0.9970 - val_loss: 0.0830 - val_accuracy: 0.9808
Epoch 16/30
300/300 [==============================] - 4s 13ms/step - loss: 0.0084 - accuracy:
0.9970 - val_loss: 0.0881 - val_accuracy: 0.9802
Epoch 17/30
300/300 [==============================] - 3s 11ms/step - loss: 0.0075 - accuracy:
0.9976 - val_loss: 0.0948 - val_accuracy: 0.9796
Epoch 18/30
```

```
300/300 [==============================] - 4s 15ms/step - loss: 0.0053 - accuracy:
0.9984 - val_loss: 0.0831 - val_accuracy: 0.9823
Epoch 19/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0057 - accuracy:
0.9981 - val_loss: 0.0859 - val_accuracy: 0.9824
Epoch 20/30
300/300 [==============================] - 3s 11ms/step - loss: 0.0081 - accuracy:
0.9975 - val_loss: 0.0971 - val_accuracy: 0.9796
Epoch 21/30
300/300 [==============================] - 3s 11ms/step - loss: 0.0044 - accuracy:
0.9986 - val_loss: 0.1118 - val_accuracy: 0.9792
Epoch 22/30
300/300 [==============================] - 3s 11ms/step - loss: 0.0058 - accuracy:
0.9981 - val_loss: 0.1036 - val_accuracy: 0.9790
Epoch 23/30
300/300 [==============================] - 3s 11ms/step - loss: 0.0058 - accuracy:
0.9981 - val_loss: 0.1073 - val_accuracy: 0.9803
Epoch 24/30
300/300 [==============================] - 3s 11ms/step - loss: 0.0020 - accuracy:
0.9993 - val_loss: 0.1047 - val_accuracy: 0.9826
Epoch 25/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0027 - accuracy:
0.9992 - val_loss: 0.0963 - val_accuracy: 0.9832
Epoch 26/30
300/300 [==============================] - 3s 11ms/step - loss: 0.0019 - accuracy:
0.9994 - val_loss: 0.0980 - val_accuracy: 0.9815
Epoch 27/30
300/300 [==============================] - 4s 14ms/step - loss: 0.0114 - accuracy:
0.9965 - val_loss: 0.1091 - val_accuracy: 0.9778
Epoch 28/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0073 - accuracy:
0.9976 - val_loss: 0.0969 - val_accuracy: 0.9813
Epoch 29/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0028 - accuracy:
0.9991 - val_loss: 0.0981 - val_accuracy: 0.9824
Epoch 30/30
300/300 [==============================] - 4s 12ms/step - loss: 0.0021 - accuracy:
0.9994 - val_loss: 0.0969 - val_accuracy: 0.9826
```

In [30]:

```
print("Test Accuracy : %.4f" %(m.evaluate(X_test, Y_test)[1]))
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.0969 - accuracy: 0.
9826
Test Accuracy : 0.9826
```

In [31]:

```
pred = m.predict(X_test)
```

In [32]:

```
print( pred.shape )
print( pred[1] )
```

```
(10000, 10)
[1.0948021e-20 3.0981642e-15 1.0000000e+00 9.5306487e-15 1.3041178e-28
 2.3261916e-19 1.0367870e-20 2.5786484e-23 9.9193638e-19 1.6119041e-24]
```