MNIST - Categorical Classification

Overfitting Issue

Import Tensorflow & Keras

```
import warnings
warnings.filterwarnings('ignore')
```

- import TensorFlow

```
import tensorflow as tf

tf.__version__
```

```
'2.4.1'
```

- GPU 설정 확인

```
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```

- import Keras

```
import keras

keras.__version__
```

```
'2.4.3'
```

I. MNIST Data_Set Load & Review

1) Load MNIST Data_Set

```
from keras.datasets import mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
```

```
11493376/11490434 [==============================] - 0s 0us/step
```

- Train_Data Information

```
print(len(X_train))
print(X_train.shape)

print(len(y_train))
print(y_train[0:5])
```

```
60000
(60000, 28, 28)
60000
[5 0 4 1 9]
```

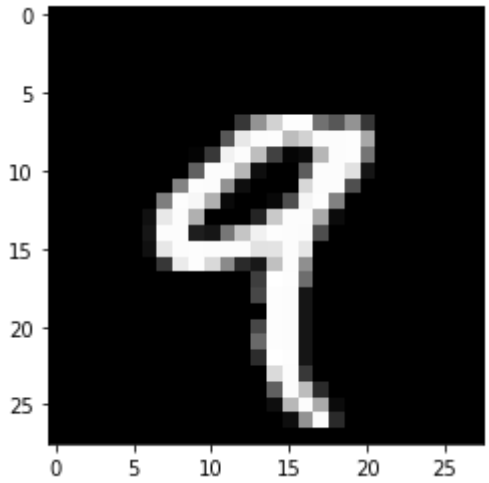
- Test_Data Information

```
print(len(X_test))
print(X_test.shape)

print(len(y_test))
print(y_test[0:5])
```

```
10000
(10000, 28, 28)
10000
[7 2 1 0 4]
```

2) Visualization

```
import matplotlib.pyplot as plt

digit = X_train[4]
plt.imshow(digit, cmap = 'gray')
plt.show()
```



```
import numpy as np
np.set_printoptions(linewidth = 150)

print(X_train[4])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0  55 148 210 253 253 113  87 148  55   0
 [  0   0   0   0   0   0   0   0   0   0   0  87 232 252 253 189 210 252 252 253 168   0
 [  0   0   0   0   0   0   0   0   0   4  57 242 252 190  65   5  12 182 252 253 116   0
 [  0   0   0   0   0   0   0   0   0  96 252 252 183  14   0   0  92 252 252 225  21   0
 [  0   0   0   0   0   0   0   0 132 253 252 146  14   0   0   0 215 252 252  79   0   0
 [  0   0   0   0   0   0   0 126 253 247 176   9   0   0   8  78 245 253 129   0   0   0
 [  0   0   0   0   0   0  16 232 252 176   0   0   0  36 201 252 252 169  11   0   0   0
 [  0   0   0   0   0   0  22 252 252  30  22 119 197 241 253 252 251  77   0   0   0   0
 [  0   0   0   0   0   0  16 231 252 253 252 252 252 226 227 252 231   0   0   0   0   0
 [  0   0   0   0   0   0   0  55 235 253 217 138  42  24 192 252 143   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  62 255 253 109   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  71 253 252  21   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 253 252  21   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  71 253 252  21   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 106 253 252  21   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  45 255 253  21   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 218 252  56   0   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  96 252 189  42   0   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  14 184 252 170  11   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  14 147 252  42   0   0   0
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

II. Data Preprocessing

1) Reshape and Normalization

- reshape
    - (60000, 28, 28) to (60000, 784)

```
X_train = X_train.reshape((60000, 28 * 28))
X_test = X_test.reshape((10000, 28 * 28))

X_train.shape, X_test.shape
```

```
((60000, 784), (10000, 784))
```

- Normalization

```
X_train = X_train.astype(float) / 255
X_test = X_test.astype(float) / 255
```

```
print(X_train[4])
```

```
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.21568627 0.58039216 0.82352941 0.99215686 0.99215686 0.44313725 0.34117647 0.58039216
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.34117647 0.90980392 0.98823529 0.99215686 0.74117647 0.82352941 0.98823529
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.01568627 0.22352941 0.94901961 0.98823529 0.74509804 0.25490196 0.01960784
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.37647059 0.98823529 0.98823529 0.71764706 0.05490196
 0.88235294 0.08235294 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.51764706 0.99215686 0.98823529 0.57254902
 0.98823529 0.98823529 0.30980392 0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.49411765 0.99215686 0.96862745
 0.30588235 0.96078431 0.99215686 0.50588235 0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.0627451  0.90980392
 0.14117647 0.78823529 0.98823529 0.98823529 0.6627451  0.04313725 0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.46666667 0.77254902 0.94509804 0.99215686 0.98823529 0.98431373 0.30196078 0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.99215686 0.98823529 0.98823529 0.98823529 0.88627451 0.89019608 0.98823529 0.90588235
 0.          0.          0.          0.          0.          0.          0.          0.
 0.21568627 0.92156863 0.99215686 0.85098039 0.54117647 0.16470588 0.09411765 0.75294118
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.08235294 0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.99215686 0.98823529 0.08235294 0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.27843137 0.99215686 0.98823529 0.08235294 0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.41568627 0.99215686 0.98823529 0.08235294 0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.17647059 1.         0.99215686
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
 0.16470588 0.          0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.          0.          0.
```

```
0.72156863 0.98823529 0.66666667 0.04313725 0.         0.         0.         0.         0
0.         0.         0.         0.         0.         0.         0.         0.         0
0.         0.         0.05490196 0.57647059 0.98823529 0.16470588 0.         0.         0
0.         0.         0.         0.         0.         0.         0.         0.         0
0.         0.         0.         0.         0.         0.         0.         0.         0
```

2) One Hot Encoding

```
from keras.utils import to_categorical

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
print(y_train[:5])
```

```
[[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.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

III. MNIST Keras Modeling

1) Model Define

- 모델 신경망 구조 정의
  - 2개의 Hidden Layers & 768개의 Nodes
  - 복잡한 Model Capacity로 인한 Overfitting

```
from keras import models
from keras import layers

mnist = models.Sequential()
mnist.add(layers.Dense(512, activation = 'relu', input_shape = (28 * 28,)))
mnist.add(layers.Dense(256, activation = 'relu'))
mnist.add(layers.Dense(10, activation = 'softmax'))
```

- 모델 구조 확인

```
mnist.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
```

2) Model Compile

- 모델 학습방법 설정

```
mnist.compile(loss = 'categorical_crossentropy',
              optimizer = 'rmsprop',
              metrics = ['accuracy'])
```

3) Model Fit

- 약 3분

```
%%time

Hist_mnist = mnist.fit(X_train, y_train,
                       epochs = 100,
                       batch_size = 128,
                       validation_split = 0.2)
```

```
375/375 [==============================] - 1s 3ms/step - loss: 0.0017 - accuracy: 0.9996
Epoch 74/100
375/375 [==============================] - 1s 3ms/step - loss: 3.8782e-04 - accuracy: 0.99
Epoch 75/100
375/375 [==============================] - 1s 3ms/step - loss: 2.1999e-04 - accuracy: 0.99
Epoch 76/100
375/375 [==============================] - 1s 3ms/step - loss: 1.6864e-04 - accuracy: 1.00
Epoch 77/100
375/375 [==============================] - 1s 4ms/step - loss: 6.8045e-04 - accuracy: 0.99
Epoch 78/100
375/375 [==============================] - 1s 3ms/step - loss: 6.0276e-04 - accuracy: 0.99
Epoch 79/100
375/375 [==============================] - 1s 3ms/step - loss: 9.8995e-05 - accuracy: 1.00
Epoch 80/100
375/375 [==============================] - 1s 3ms/step - loss: 4.5880e-04 - accuracy: 0.99
Epoch 81/100
375/375 [==============================] - 1s 3ms/step - loss: 8.4441e-04 - accuracy: 0.99
Epoch 82/100
375/375 [==============================] - 1s 3ms/step - loss: 0.0012 - accuracy: 0.9999 -
Epoch 83/100
```

```
375/375 [==============================] - 1s 3ms/step - loss: 6.1047e-04 - accuracy: 0.99
Epoch 84/100
375/375 [==============================] - 1s 3ms/step - loss: 6.0354e-04 - accuracy: 0.99
Epoch 85/100
375/375 [==============================] - 1s 3ms/step - loss: 0.0011 - accuracy: 0.9997 -
Epoch 86/100
375/375 [==============================] - 1s 3ms/step - loss: 2.6732e-04 - accuracy: 0.99
Epoch 87/100
375/375 [==============================] - 1s 3ms/step - loss: 6.0930e-04 - accuracy: 0.99
Epoch 88/100
375/375 [==============================] - 1s 3ms/step - loss: 2.3086e-04 - accuracy: 1.00
Epoch 89/100
375/375 [==============================] - 1s 3ms/step - loss: 0.0017 - accuracy: 0.9998 -
Epoch 90/100
375/375 [==============================] - 1s 3ms/step - loss: 1.6866e-04 - accuracy: 0.99
Epoch 91/100
375/375 [==============================] - 1s 3ms/step - loss: 6.7574e-04 - accuracy: 0.99
Epoch 92/100
375/375 [==============================] - 1s 3ms/step - loss: 2.9096e-04 - accuracy: 0.99
Epoch 93/100
375/375 [==============================] - 1s 3ms/step - loss: 1.4330e-04 - accuracy: 0.99
Epoch 94/100
375/375 [==============================] - 1s 3ms/step - loss: 7.1412e-04 - accuracy: 0.99
Epoch 95/100
375/375 [==============================] - 1s 3ms/step - loss: 2.8642e-04 - accuracy: 1.00
Epoch 96/100
375/375 [==============================] - 1s 3ms/step - loss: 6.8896e-04 - accuracy: 0.99
Epoch 97/100
375/375 [==============================] - 1s 3ms/step - loss: 8.1925e-05 - accuracy: 1.00
Epoch 98/100
375/375 [==============================] - 1s 3ms/step - loss: 3.8081e-06 - accuracy: 1.00
Epoch 99/100
375/375 [==============================] - 1s 3ms/step - loss: 1.0084e-08 - accuracy: 1.00
Epoch 100/100
375/375 [==============================] - 1s 3ms/step - loss: 1.8408e-10 - accuracy: 1.00

CPU times: user 2min 11s, sys: 19.1 s, total: 2min 31s
Wall time: 2min 4s
```

4) 학습 결과 시각화 - Overfitting

- Loss Visualization

```
import matplotlib.pyplot as plt

epochs = range(1, len(Hist_mnist.history['loss']) + 1)

plt.figure(figsize = (9, 6))
plt.plot(epochs, Hist_mnist.history['loss'])
plt.plot(epochs, Hist_mnist.history['val_loss'])
# plt.ylim(0, 0.25)
plt.title('Training & Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.ylabel('Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.grid()
plt.show()
```

5) Model Evaluate

- Loss & Accuracy

```
loss, accuracy = mnist.evaluate(X_test, y_test)

print('Loss = {:.5f}'.format(loss))
print('Accuracy = {:.5f}'.format(accuracy))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3297 - accuracy: 0.9840
Loss = 0.32968
Accuracy = 0.98400
```

6) Model Predict

- Probability

```
np.set_printoptions(suppress = True, precision = 9)

print(mnist.predict(X_test[:1,:]))
```

```
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
```

- Class

```
print(mnist.predict_classes(X_test[:1,:]))
```

```
[7]
```

#

#

#

The End

#

#

#