Chapter 12

Keras Deep Neural Network – MNIST

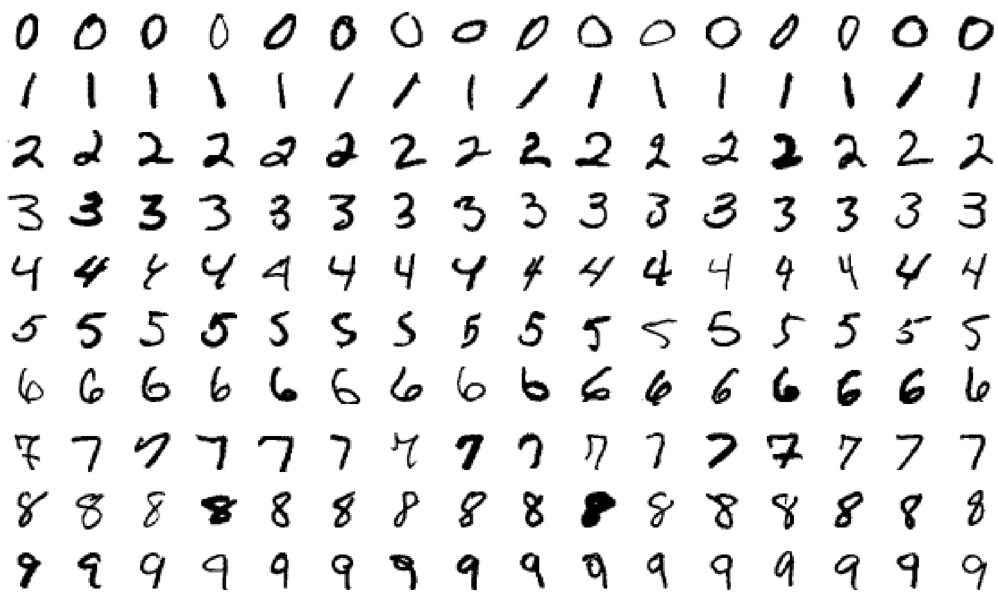
오 세 종

DANKOOK UNIVERSITY

Contents

1. MNIST dataset

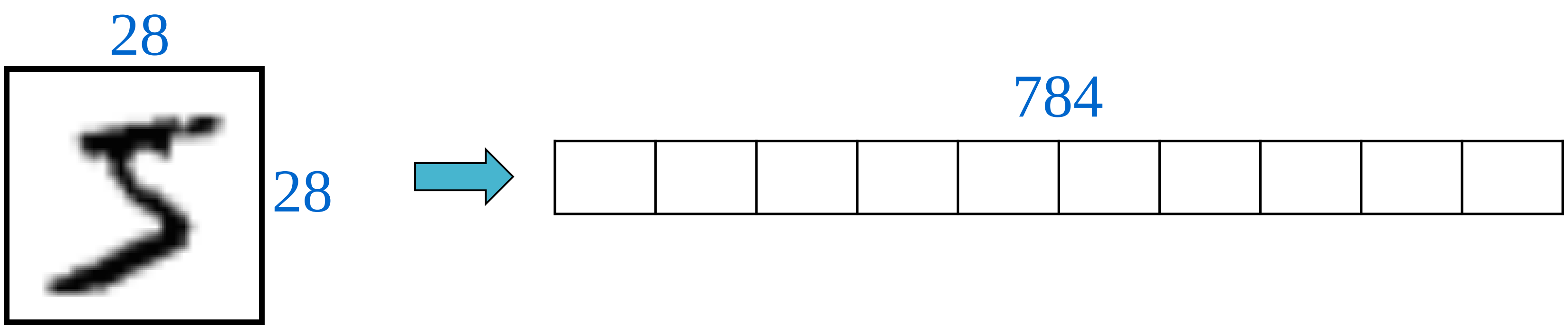
- MNIST database (Modified National Institute of Standards and Technology database)
  - 손으로 쓴 숫자들로 이루어진 대형 데이터베이스이며, 다양한 영상 처리 시스템을 트레이닝하기 위해 일반적으로 사용
  - Training, test 데이터를 별도로 제공
  - http://yann.lecun.com/exdb/mnist/
  - Keras 에도 포함되어 있음



MINST 테스트 데이터셋의 샘플 이미지

1. MNIST dataset

- 데이터 형태
  - 28 x 28 사이즈의 흑백 이미지
  - 1 pixel 은 0~255 의 값 저장
  - 2 차원 형태의 데이터는 학습을 할 수 없으므로 1x784 형태의 1차원 이미지로 변경하여 사용



  - 0~255 사이의 픽셀값은 0~1 사이로 변환하여 사용
  - Class 레이블 개수 : 10개 (0~9)

2. Prepare dataset

```
# load required modules
from keras.datasets import mnist
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.utils import np_utils

import matplotlib.pyplot as plt
import numpy as np
```

2. Prepare dataset

```
# load dataset
(train_X, train_y), (test_X, test_y) = mnist.load_data()
train_X, test_X = train_X / 255.0, test_X / 255.0

# one hot encoding
train_y = np_utils.to_categorical(train_y)
test_y = np_utils.to_categorical(test_y)
```

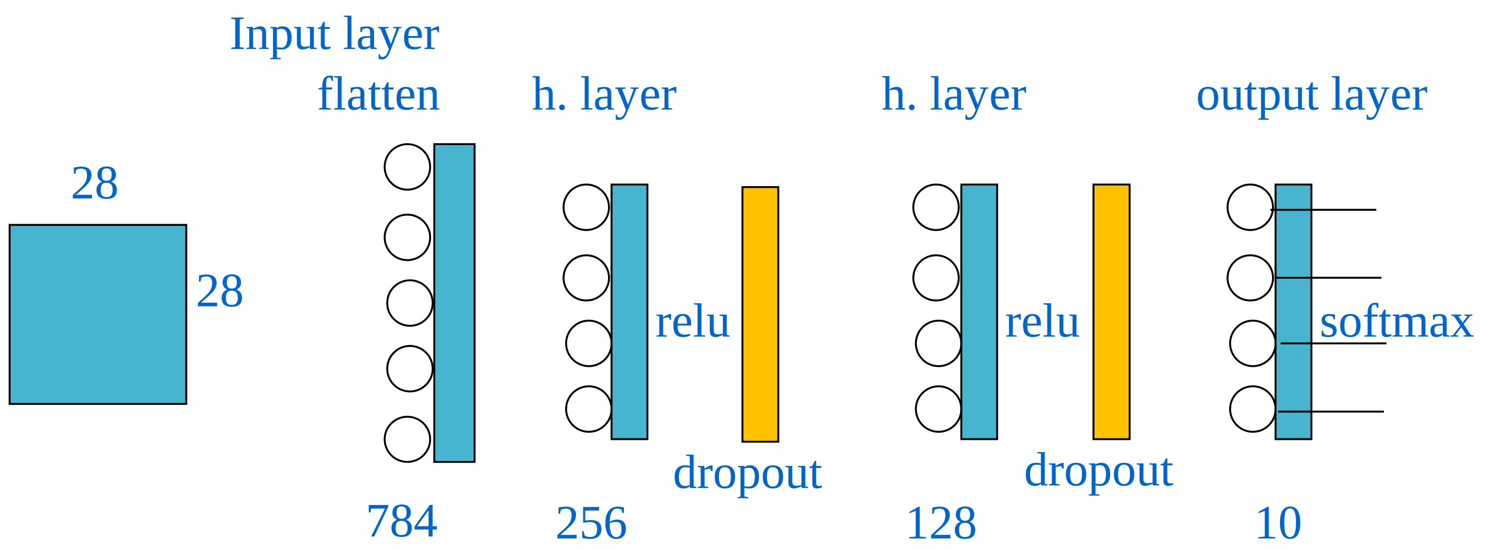
Name	Type	Size	Value
str	list	4	['K9', 'OS', 'OPTIC', 'roi.jpg']
x_test	Array of float64	(10000, 28, 28)	[[[0. 0. 0. ... 0. 0. 0.] [0. 0. 0. ... 0. 0. 0.]
x_train	Array of float64	(60000, 28, 28)	[[[0. 0. 0. ... 0. 0. 0.] [0. 0. 0. ... 0. 0. 0.]
y_classes	list	10000	[7, 2, 1, 0, 4, 1, 4, 9, 5, 9, ...]
y_test	Array of float32	(10000, 10)	[[0. 0. 0. ... 1. 0. 0.] [0. 0. 1. ... 0. 0. 0.]
y_train	Array of float32	(60000, 10)	[[0. 0. 0. ... 0. 0. 0.] [1. 0. 0. ... 0. 0. 0.]

Help | Variable explorer | Plots | Files | Code Analysis

Kite: indexing | conda: base (Python 3.7.6) | Line 23, Col 1 | UTF-8 | CRLF | RW | Mem 79%

3. Model setup

- Network design



3. Model setup

```
# define model (DNN structure)
epochs = 20
batch_size = 128
learning_rate = 0.01

model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate = 0.4))
model.add(Dense(128, activation='relu'))
model.add(Dropout(rate = 0.3))
model.add(Dense(10, activation='softmax'))

model.summary()  # show model structure
```

3. Model setup

```
In [146]: model.summary()  # show model structure
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_3 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_38 (Dense)             (None, 256)               200960    
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_39 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_40 (Dense)             (None, 10)                1290      
=================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
```

4. Model compile & fitting

```
# Compile model
adam = optimizers.adam(lr=learning_rate)
model.compile(loss='categorical_crossentropy',
              optimizer=adam,
              metrics=['accuracy'])

# model fitting (learning)
disp = model.fit(train_X, train_y,
                 batch_size=batch_size,
                 epochs=epochs,
                 verbose=1,       # print fitting process
                 validation_split = 0.2)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/20
48000/48000 [==============================] - 2s 38us/step - loss: 0.2860 - accuracy: 0.9187 -
val_loss: 0.1352 - val_accuracy: 0.9615
Epoch 2/20
48000/48000 [==============================] - 2s 32us/step - loss: 0.2674 - accuracy: 0.9264 -
val_loss: 0.1414 - val_accuracy: 0.9613
Epoch 3/20
48000/48000 [==============================] - 2s 31us/step - loss: 0.2548 - accuracy: 0.9306 -
val_loss: 0.1464 - val_accuracy: 0.9613
Epoch 4/20
48000/48000 [==============================] - 2s 31us/step - loss: 0.2556 - accuracy: 0.9317 -
val_loss: 0.1444 - val_accuracy: 0.9643
Epoch 5/20
48000/48000 [==============================] - 2s 31us/step - loss: 0.2409 - accuracy: 0.9367 -
val_loss: 0.1325 - val_accuracy: 0.9647
Epoch 6/20
48000/48000 [==============================] - 2s 32us/step - loss: 0.2403 - accuracy: 0.9374 -
val_loss: 0.1343 - val_accuracy: 0.9657
Epoch 7/20
48000/48000 [==============================] - 2s 31us/step - loss: 0.2246 - accuracy: 0.9410 -
val_loss: 0.1366 - val_accuracy: 0.9653
Epoch 8/20
48000/48000 [==============================] - 2s 32us/step - loss: 0.2326 - accuracy: 0.9404 -
val_loss: 0.1425 - val_accuracy: 0.9628
```

5. Test

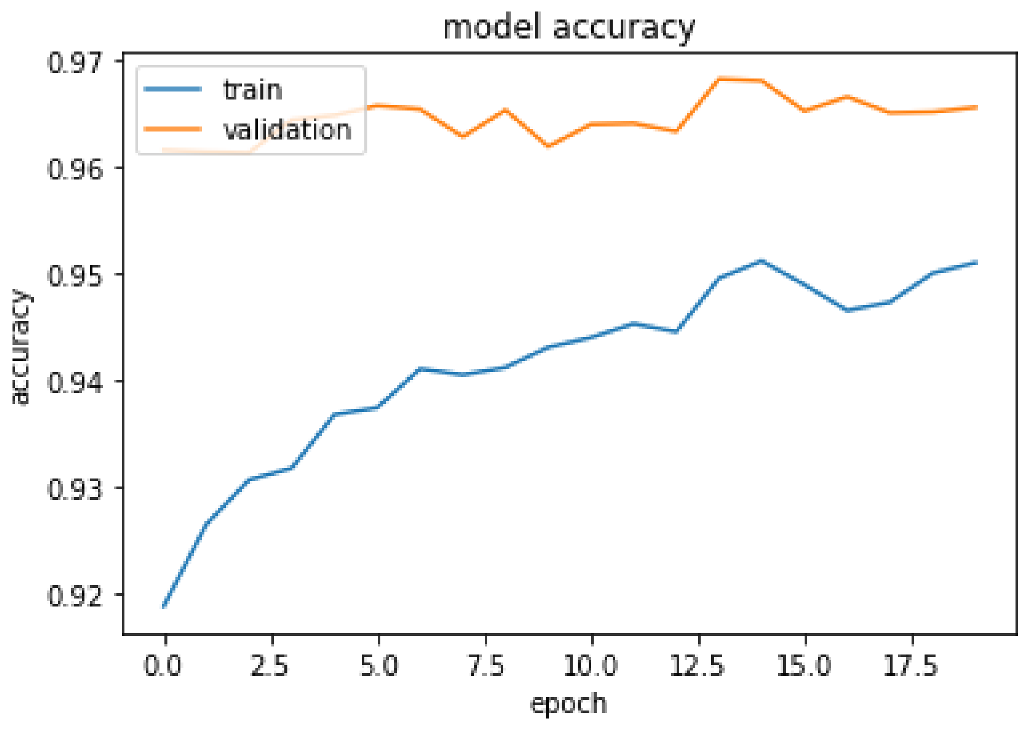
```
# Test model
pred = model.predict(test_X)
print(pred)
y_classes = [np.argmax(y, axis=None, out=None) for y in pred]
print(y_classes)   # result of prediction

# model performance
score = model.evaluate(test_X, test_y, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

# summarize history for accuracy
plt.plot(disp.history['accuracy'])
plt.plot(disp.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

5. Test

```
In [147]: print('Test loss:', score[0])
     ...: print('Test accuracy:', score[1])
Test loss: 0.144199655760663
Test accuracy: 0.9674000144004822
```



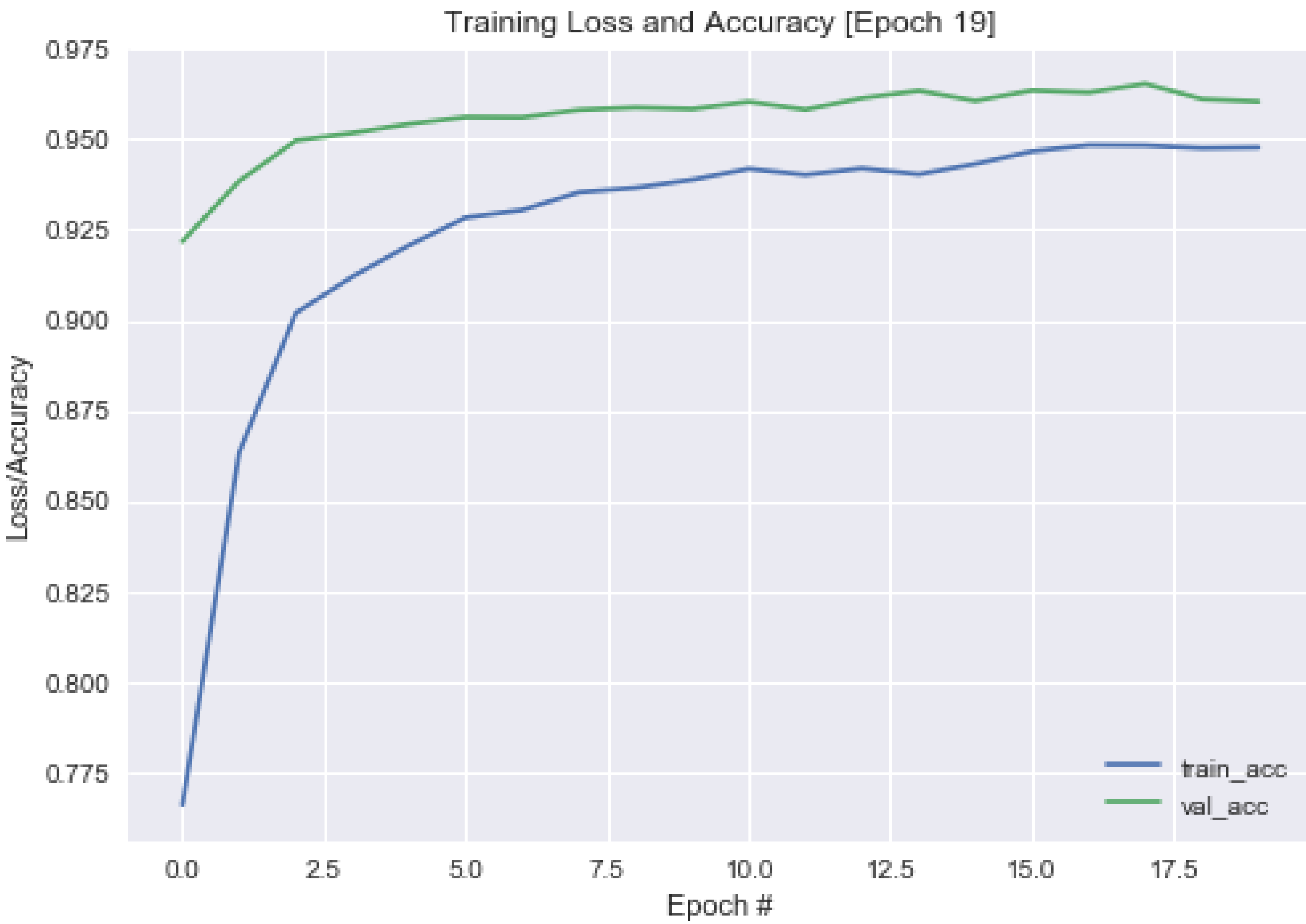
5. Test

```
# summarize history for loss
plt.plot(disp.history['loss'])
plt.plot(disp.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

6. Monitoring fitting process

```
…
import TrainPlot                    # call TrainPlot.py
…
…
# model fitting (learning)
disp = model.fit(train_X, train_y,
                 batch_size=batch_size,
                 epochs=epochs,
                 verbose=1,        # print fitting process
                 validation_split = 0.2,
                 callbacks=[TrainPlot.TrainingPlot()])
```

6. Monitoring fitting process



If you modify `TrainPlot.py`, you also can see loss plot or both acc and loss plots.